

# A Conceptual Framework for Integrating Conversational Agents in Digital Education

Présentée le 16 juin 2023

Faculté des sciences et techniques de l'ingénieur  
Groupe SCI STI DG  
Programme doctoral en robotique, contrôle et systèmes intelligents

pour l'obtention du grade de Docteur ès Sciences

par

**Juan Carlos FARAHA**

Acceptée sur proposition du jury

Prof. F. Mondada, président du jury  
Dr D. Gillet, directeur de thèse  
Prof. M. Notari, rapporteur  
Prof. E. L.-C. Law, rapporteuse  
Dr E. Amico, rapporteur



Me gusta andar,  
pero no sigo el camino,  
pues lo seguro ya no tiene misterio.

— Facundo Cabral

To my parents, Rocío and Juan Manuel, for their unconditional support throughout the years.

To my wife, Clarisse, for joining me on this journey.

And to my son, Sandro, for helping me put everything into perspective.





# Acknowledgements

It goes without saying that I could not have completed this thesis without the support of many, many people. A full list would require more pages than those comprising the content of this thesis. In my attempt at brevity, I might have forgotten some of you. Please accept my most heartfelt apologies. I owe you a drink.

First of all, I would like to thank my thesis advisor, Denis Gillet, for his guidance, patience, and flexibility as I homed in on the topic for this thesis, developed it, and carried out the work over the past few years. I do not think I could have imagined a better place to undertake this project than the Interaction Systems Group (REACT) at the École Polytechnique Fédérale de Lausanne. Thank you for making this thesis possible. I would also like to thank the members of the jury, Francesco Mondada, Enrico Amico, Effie L.-C. Law, and Michele Notari. Your feedback has made this thesis stronger and has helped me reflect on the road that lies ahead.

A huge thank you goes out to those who have mentored me throughout the years. To Pedro Mediano, for (re)igniting my passion for research and for helping me map out the years to come. To María Jesús Rodríguez-Triana, for her kind words and pertinent feedback as I dipped my toes into a new research domain. And to Sandy Ingram and Adrian Holzer, for always being available to brainstorm ideas and for hosting the multiple field experiments featured in this thesis. I am also especially grateful to Basile Spaenlehauer, who has been my main collaborator in this endeavor and without whom I would not have managed to conduct all this work on time. Thank you for approaching every crazy idea with enthusiasm. Your support has been essential.

I have also been lucky to work alongside wonderful colleagues at REACT: Andrii Vozniuk, Alex Wild, Wissam Halimi, Isabelle Vonèche-Cardia, Yves Piguët, Matin Macktoobian, Adeline Ung, Graciana Aad, Shenyi Wang, Julien Torrent, Maria Gaci, Nour Ghalia Abassi, Rania Islam-bouli, Jérémy La Scala, Alexandre Chau, Kim Lan Phan Hoang, Christine Vuichoud. Thank you for lightening up the physical and virtual workplace over the years. Another big thank you goes out to colleagues beyond REACT, who have been incredibly helpful as I explored different research directions: Jessica Pidoux, Nathalie Meyer, Hanna Tolle, Nicolas Gninenko, Andrea Santoro, Sara Stampacchia, Pamela Andrade Sevillano, Nuno R. C. Gomes, Stian Håklev, Andrea Luppi, Fernando Rosas, Chiara Mazzocchi, Thierry Chaminade, Sabrina Lenzoni, Aditya K. Purohit, Kristoffer Bergram, Arielle Moro, Alessio De Santo, Katerina Mangaroska.

## Acknowledgements

---

I look forward to continuing our collaborations in the postdoctoral future. And thank you to the students with whom I have had the pleasure to work through various projects, theses, and internships: Joana Soares Machado, Uchendu Nwachukwu, Hassan Abdul Ghaffar, Pedro Torres da Cunha, Sola Olateju, Vandit Sharma, Xinyang Lu, Hadi Khairallah, Abdallah Al Chami, Roland Salloum, Víctor González, Álvaro Bautista, Fanny Lasne, Malin Svenberg. Working with you has been a key part of this experience.

I would also like to thank my friends since forever: Christos Kaplanis, Daniel Rodrigo, Ryon Hart, Sergio Saba, Tony Acuña-Rohter, Jack Fishburn, Julian Bonierbale, Victor Trokoudes, Alexander de Carvalho, Petros Andreou, Karim Hakimzadeh, Pablo Tsutsumi, Bruno Alberti. An extra special thanks to Martín Otero Knott for the endless flow of wise words and to Hagop Taminian for actually joining REACT and collaborating with me on some of the work included in this thesis. Moving to Lausanne has been quite an adventure and I do not think this adventure would have been the same without the friends I made and reconnected with: Eduardo Saldaña Piovanneti, Christophe Huguenin, Michele Kotiuga, Ezequiel González Debada, Raquel Zambrano, André Nogueira, Jonathan Blazek, Louise Skinnari, Renato Zanetti, Giulia Bommarito. Thank you for helping me make Lausanne my home.

My most heartfelt *thank you* goes to my family for their unwavering support. To my siblings, Juan Diego, Vanessa, Juan Manuel, and Úrsula, and to their respective families. Thank you for always being there notwithstanding the distance and the odd pandemic. To my in-laws, Hélène and Patrick, for welcoming me with arms wide open into their family. To my uncle Diego, for inspiring me to write and to focus on what matters the most. To my aunt Rosmarie, for making Switzerland feel like home. Finally, to my grandparents, who are no longer with me, but who I think would be proud—each in his/her own way—of this accomplishment. And to my extended family in Peru, Portugal, France, Switzerland, and beyond. Gracias, obrigado, merci, and danke.

Most importantly, I would like to thank my parents, Rocío and Juan Manuel, for providing me with the opportunity to pursue my dreams and for encouraging me to embark on countless adventures in uncharted waters. Thank you for the love and support.

Of course, this thesis would not have been possible without my wife, Clarisse. She provided me with the inspiration to undertake this challenge, with the courage to set sail, and with the much-needed motivation when the winds were against us. Thank you for the love and support.

And to my son, Sandro. For making this achievement so much more fulfilling and special. I hope that someday these words will inspire you to embark on your own adventure. Thank you for the love and support.

*Lausanne, 31 May 2023*

Juan Carlos Farah

# Résumé

La présence d'agents conversationnels — ou *chatbots* — dans des contextes éducatifs n'a cessé d'augmenter au cours des dernières années. Des enquêtes récentes ont montré un intérêt généralisé pour l'utilisation des chatbots dans l'éducation, à la fois pour la recherche et la pratique. Bien que ces enquêtes mettent en évidence des avantages tangibles et des applications futures prometteuses des chatbots éducatifs, plusieurs défis limitent notre capacité à les intégrer dans des contextes éducatifs réels, notamment des défis *technologiques*, *pédagogiques*, d'*interaction* et de *conception*. Dans cette thèse, nous motivons notre approche pour relever ces défis en formulant une question de recherche globale : *Comment pouvons-nous accompagner l'intégration des chatbots dans des contextes d'apprentissage spécifiques à un domaine*? Pour répondre à cette question — et s'attaquer à certains des défis identifiés dans la littérature spécialisée — nous proposons un cadre conceptuel couvrant quatre dimensions différentes.

Notre enquête commence par aborder les *fondements technologiques* de notre cadre à travers une architecture de développement et un pipeline d'analyse d'apprentissage visant à soutenir la création d'applications interactives pour les plateformes d'éducation numérique et à donner accès aux données générées lorsque les apprenants interagissent avec ces applications. À l'aide de notre architecture, nous nous focalisons ensuite sur un domaine — la formation en génie logiciel — et développons les outils numériques nécessaires pour échafauder des *scénarios pédagogiques* dans lesquels ces chatbots pourraient interagir avec les apprenants. Plus précisément, nous proposons le cahier de revue de code, un modèle de construction de scénarios technopédagogiques pour soutenir l'enseignement des meilleures pratiques de programmation. Les cahiers de révision de code ressemblent aux interactions que les développeurs ont sur les plateformes sociales de codage et, compte tenu de la popularité des chatbots sur ces plateformes, ils sont a fortiori adaptés aux chatbots éducatifs.

Dans une série d'études en ligne, d'observation et de retours de terrain, nous explorons ensuite différentes *stratégies d'interaction* qui pourraient être exploitées par les chatbots éducatifs dans leurs échanges avec les apprenants. Les résultats de ces études sont pertinents pour les enseignants qui cherchent à intégrer les chatbots éducatifs dans leur pratique quotidienne et ont servi à éclairer deux contributions finales proposées dans cette thèse. Ces contributions portent sur le *processus de conception* et comprennent un modèle pour guider la conception participative de chatbots éducatifs, ainsi qu'un schéma technique pour définir comment ces

## Résumé

---

chatbots pourraient être intégrés dans des applications d'apprentissage numérique. En faisant un zoom arrière les deux contributions finales visent à généraliser nos résultats à d'autres domaines éducatifs.

Alors que les interfaces de programmation d'applications vers de puissants modèles de langage génératif deviennent largement accessibles, nous ne pouvons que nous attendre à ce que des chatbots éducatifs de plus en plus complexes deviennent omniprésents dans les années à venir. Comprendre comment les apprenants interagissent avec ces chatbots et fournir le soutien nécessaire pour encadrer leur développement est donc primordial. Notre travail s'oriente sur cet axe de recherche.

## Mots clés

agents conversationnels, chatbots, éducation numérique, génie logiciel, interaction homme-machine, intelligence artificielle, plateformes d'expérience d'apprentissage, formation en génie logiciel, études empiriques, conception participative

# Abstract

The presence of conversational agents—or *chatbots*—in educational contexts has been steadily increasing over the past few years. Recent surveys have shown widespread interest in the use of chatbots in education, both for research and practice. Although these surveys highlight tangible benefits and promising future applications of educational chatbots, several challenges limit our ability to integrate these chatbots into educational contexts, including *technological*, *pedagogical*, *interaction*, and *design* challenges. In this thesis, we motivate our approach to these challenges by formulating one overarching research question: *How can we guide the integration of chatbots into domain-specific learning contexts?* To address this question and, in turn, tackle some of the challenges identified in the literature, we propose a conceptual framework spanning four different dimensions. Following the design-based research methodology, we address each dimension in a corresponding phase of our design process, undertaking multiple iterations of the design cycle within each phase.

Our investigation starts by addressing the *technological foundations* of our framework through an application development architecture and a learning analytics pipeline aimed at supporting the creation of interactive applications for digital education platforms and providing access to the data generated when learners interact with these applications. Using our architecture, we then zoom in on one domain—software engineering education—and develop the applications needed to scaffold *pedagogical scenarios* in which these chatbots could interact with learners. Specifically, we propose the code review notebook, a template for building technopedagogical scenarios to support teaching programming best practices. Code review notebooks resemble the interactions developers have on social coding platforms and—given the popularity of chatbots on these platforms—are especially suitable for educational chatbots.

In a series of online, observational, and field studies, we then explore different *interaction strategies* that could be harnessed by educational chatbots in their conversations with learners. In particular, we conducted three field studies to assess the effects that educational chatbots following (i) Wizard of Oz, (ii) rule-based, and (iii) large language model-based conversational strategies could have on different aspects of the learning experience. Findings from these studies are relevant to instructors looking to integrate educational chatbots into their practice and served to inform two final contributions proposed in this thesis. These contributions focus on *design processes* and comprise a model to guide the participatory design of educational chatbots, as well as a technical blueprint to define how these chatbots could be integrated into

## **Abstract**

---

digital learning applications. Zooming out from the software engineering education use case, the two final contributions aim to generalize our findings to other educational domains.

As application programming interfaces to powerful generative language models become widely accessible, we can only expect that increasingly complex educational chatbots will become ubiquitous in the years to come. Understanding how learners interact with these chatbots and providing the support necessary to guide their development is therefore of paramount importance. Our framework aligns itself closely with this line of research.

## **Keywords**

conversational agents, chatbots, digital education, software engineering, human-computer interaction, artificial intelligence, learning experience platforms, software engineering education, empirical studies, participatory design

# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract (Français/English)</b>	<b>iii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xxi</b>
<b>Acronyms</b>	<b>xxiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research Context . . . . .	3
1.1.1 Digital Education . . . . .	4
1.1.2 Software Engineering . . . . .	5
1.1.3 Human-Computer Interaction . . . . .	6
1.1.4 Artificial Intelligence . . . . .	7
1.2 Problem Statement . . . . .	9
1.3 Research Questions and Objectives . . . . .	11
1.4 Research Methodology . . . . .	12
1.5 Contributions . . . . .	14
1.6 Structure . . . . .	19
<b>I Technological Foundations</b>	<b>21</b>
Related Work . . . . .	23
<b>2 Application Development Architecture</b>	<b>27</b>
2.1 Introduction . . . . .	28
2.2 Design . . . . .	28
2.2.1 Components . . . . .	29
2.2.2 Process . . . . .	31
2.3 Implementation . . . . .	33
2.3.1 Graasp . . . . .	33
2.3.2 Templates . . . . .	34
2.3.3 Command-Line Interface . . . . .	35

## Contents

---

2.3.4	Local Development . . . . .	36
2.3.5	Repository . . . . .	36
2.4	Methodology . . . . .	38
2.4.1	Participants . . . . .	39
2.4.2	Instruments . . . . .	39
2.4.3	Data Analysis . . . . .	39
2.5	Results . . . . .	39
2.5.1	Technical Aspect . . . . .	39
2.5.2	Educational Aspect . . . . .	40
2.5.3	Licensing Aspect . . . . .	40
2.6	Discussion and Implications . . . . .	40
2.7	Conclusion . . . . .	41
<b>3</b>	<b>Learning Analytics Pipeline</b>	<b>43</b>
3.1	Introduction . . . . .	44
3.2	Design Considerations . . . . .	44
3.3	Architecture . . . . .	46
3.3.1	Request for Experiment . . . . .	47
3.3.2	Consent Gathering . . . . .	48
3.3.3	Data Gathering . . . . .	48
3.3.4	Data Management . . . . .	49
3.3.5	Collaboration . . . . .	54
3.4	Discussion and Implications . . . . .	55
3.5	Conclusion . . . . .	56
<b>II</b>	<b>Pedagogical Scenarios</b>	<b>59</b>
	Related Work . . . . .	61
<b>4</b>	<b>Integrated Computational Notebooks</b>	<b>65</b>
4.1	Introduction . . . . .	66
4.2	Design . . . . .	67
4.3	Methodology . . . . .	69
4.3.1	Scenario . . . . .	70
4.3.2	Participants . . . . .	71
4.3.3	Instruments . . . . .	71
4.3.4	Data Analysis . . . . .	72
4.4	Results . . . . .	72
4.4.1	Engagement . . . . .	72
4.4.2	Usability . . . . .	73
4.4.3	Modality . . . . .	74
4.5	Discussion and Implications . . . . .	77
4.6	Conclusion . . . . .	78



<b>5</b>	<b>Code Review Notebooks</b>	<b>79</b>
5.1	Introduction . . . . .	81
5.2	Design . . . . .	81
5.2.1	Use Cases . . . . .	82
5.2.2	Interfaces . . . . .	82
5.2.3	Code Review Notebooks . . . . .	84
5.3	Methodology . . . . .	86
5.3.1	Scenario . . . . .	86
5.3.2	Participants . . . . .	88
5.3.3	Procedure . . . . .	88
5.3.4	Instruments . . . . .	89
5.3.5	Data Analysis . . . . .	90
5.4	Results . . . . .	90
5.4.1	Usability . . . . .	90
5.4.2	Learning Gains . . . . .	90
5.4.3	Self-Reflection . . . . .	91
5.4.4	Feedback . . . . .	92
5.5	Discussion and Implications . . . . .	93
5.6	Conclusion . . . . .	96
<b>III</b>	<b>Interaction Strategies</b>	<b>99</b>
	Related Work . . . . .	102
<b>6</b>	<b>Perception of Humor</b>	<b>105</b>
6.1	Introduction . . . . .	106
6.2	Design . . . . .	108
6.2.1	Chatbot Interface . . . . .	108
6.2.2	Misplaced Modifiers . . . . .	109
6.3	Methodology . . . . .	110
6.3.1	Participants . . . . .	111
6.3.2	Procedure . . . . .	112
6.3.3	Instruments . . . . .	112
6.3.4	Data Analysis . . . . .	113
6.4	Results . . . . .	114
6.5	Discussion and Implications . . . . .	116
6.6	Conclusion . . . . .	117
<b>7</b>	<b>Emoji Reactions</b>	<b>119</b>
7.1	Introduction . . . . .	120
7.2	Methodology . . . . .	121
7.2.1	Data Acquisition . . . . .	121
7.2.2	Data Analysis . . . . .	121

## Contents

---

7.3	Results . . . . .	122
7.3.1	Human versus Bot . . . . .	122
7.3.2	Popular Bots . . . . .	123
7.3.3	Qualitative Analysis . . . . .	125
7.4	Discussion and Implications . . . . .	125
7.5	Conclusion . . . . .	127
<b>8</b>	<b>Wizard of Oz</b>	<b>129</b>
8.1	Introduction . . . . .	130
8.2	Design . . . . .	131
8.2.1	Chatbot Integration . . . . .	131
8.2.2	Lint Bot . . . . .	132
8.3	Methodology . . . . .	133
8.3.1	Pedagogical Scenario . . . . .	134
8.3.2	Participants . . . . .	135
8.3.3	Procedure . . . . .	136
8.3.4	Instruments . . . . .	136
8.3.5	Data Analysis . . . . .	137
8.4	Results . . . . .	137
8.4.1	Usability . . . . .	137
8.4.2	Engagement . . . . .	139
8.4.3	Learning Gains . . . . .	139
8.4.4	Qualitative Feedback . . . . .	139
8.5	Discussion and Implications . . . . .	139
8.6	Conclusion . . . . .	140
<b>9</b>	<b>Rule-Based Scripts</b>	<b>143</b>
9.1	Introduction . . . . .	144
9.2	Design . . . . .	144
9.2.1	Automated Chatbot Responses . . . . .	145
9.2.2	Dialog Engine . . . . .	145
9.2.3	Selective Audience . . . . .	146
9.3	Methodology . . . . .	147
9.3.1	Pilot . . . . .	148
9.3.2	Pedagogical Scenario . . . . .	148
9.3.3	Procedure . . . . .	150
9.3.4	Participants . . . . .	150
9.3.5	Instruments . . . . .	150
9.3.6	Data Analysis . . . . .	151
9.4	Results . . . . .	151
9.4.1	Learning Gains . . . . .	151
9.4.2	Perceived Relevance . . . . .	152
9.4.3	Usability . . . . .	152

9.4.4	Feedback . . . . .	152
9.5	Discussion and Implications . . . . .	153
9.6	Conclusion . . . . .	155
<b>10</b>	<b>Large Language Models</b>	<b>157</b>
10.1	Introduction . . . . .	158
10.2	Design . . . . .	159
10.2.1	Architecture . . . . .	159
10.2.2	Configuration Model . . . . .	161
10.3	Methodology . . . . .	164
10.3.1	Scenario . . . . .	164
10.3.2	Participants . . . . .	167
10.3.3	Procedure . . . . .	168
10.3.4	Instruments . . . . .	168
10.3.5	Data Analysis . . . . .	168
10.4	Results . . . . .	169
10.4.1	Learning Gains . . . . .	169
10.4.2	Engagement . . . . .	169
10.4.3	Self-Reflection . . . . .	170
10.4.4	Feedback . . . . .	171
10.4.5	Usability . . . . .	172
10.5	Discussion and Implications . . . . .	173
10.6	Conclusion . . . . .	175
<b>IV</b>	<b>Design Processes</b>	<b>177</b>
	Related Work . . . . .	179
<b>11</b>	<b>Conceptual Model</b>	<b>183</b>
11.1	Introduction . . . . .	184
11.2	Design . . . . .	185
11.2.1	Backdrop . . . . .	185
11.2.2	Stakeholders . . . . .	186
11.2.3	Components . . . . .	186
11.2.4	Process . . . . .	190
11.3	Methodology . . . . .	193
11.3.1	Pilot Studies . . . . .	193
11.3.2	Illustrative Study . . . . .	196
11.3.3	Participants . . . . .	196
11.3.4	Procedure . . . . .	197
11.3.5	Instruments . . . . .	197
11.3.6	Data Analysis . . . . .	198
11.4	Results . . . . .	198

## Contents

---

11.5 Discussion and Implications . . . . .	199
11.6 Conclusion . . . . .	201
<b>12 Technical Blueprint</b>	<b>203</b>
12.1 Introduction . . . . .	204
12.2 Design Considerations . . . . .	205
12.2.1 Integrated . . . . .	205
12.2.2 Task-Oriented . . . . .	205
12.3 Architecture . . . . .	206
12.3.1 Building Blocks . . . . .	206
12.3.2 Components . . . . .	208
12.3.3 Processes . . . . .	210
12.4 Implementation . . . . .	213
12.4.1 Pedagogical Scenario . . . . .	213
12.4.2 Design . . . . .	213
12.4.3 Technical Implementation . . . . .	214
12.5 Discussion and Implications . . . . .	216
12.6 Conclusion . . . . .	219
<b>13 Conclusion</b>	<b>221</b>
13.1 Contributions . . . . .	223
13.1.1 Technological Foundations . . . . .	224
13.1.2 Pedagogical Scenarios . . . . .	224
13.1.3 Interaction Strategies . . . . .	225
13.1.4 Design Processes . . . . .	226
13.1.5 Conceptual Framework . . . . .	227
13.2 Limitations and Future Work . . . . .	229
<b>Bibliography</b>	<b>231</b>

# List of Figures

1.1	Our work is underpinned by specific areas of research spanning four disciplines. The intersections of these areas give rise to four more specific domains of research (shown in <b>bold</b> ) that guide the approach undertaken in this thesis. The four guiding research opportunities identified ( <b>RO1</b> , <b>RO2</b> , <b>RO3</b> , <b>RO4</b> ) span the four research domains underpinning our work. Further opportunities emerge in the intersections of these guiding opportunities. . . . .	3
1.2	Our initial problem statement can be broken down into four types of challenges that in turn guide the formulation of our overarching problem statement. . .	9
1.3	Our overarching research question (RQ★) was broken down into four questions that mapped onto four partial objectives, which in turn informed our overarching objective (O★). . . . .	11
1.4	Iteration type I in the holistic DBR process refers to the full cycle comprising the five semantic fields. (Adapted from Reinmann (2020).) . . . . .	14
1.5	Iteration type II in the holistic DBR process breaks a full DBR cycle into five fields of action, which consist of shorter cycles between two semantic fields. (Adapted from Reinmann (2020).) . . . . .	14
1.6	Iteration type III in the holistic DBR process breaks the full DBR cycle into playing fields, which refer to triads of adjacent semantic fields. The three iteration type III cycles used in this thesis are (i) <i>creative conceiving</i> (goal setting ↔ conception ↔ development), <i>concrete developing</i> (conception ↔ development ↔ testing), and (iii) <i>practical testing</i> (development ↔ testing ↔ analysis). (Adapted from Reinmann (2020).) . . . . .	14
1.7	The main DBR process followed in this thesis comprised four cycles, each focusing on one of the objectives highlighted in Section 1.3. Our work mainly progressed incrementally from left to right (solid arrows), but outcomes from each phase also served to refine the work that had been conducted in previous phases (dashed arrows). . . . .	15
1.8	This thesis puts forth one overarching contribution (in <b>bold</b> ) that comprises seven individual contributions. . . . .	15
1.9	An Overview of Our Research Context, Research Opportunities, Problem Statement, Research Questions, Objectives, and Contributions . . . . .	18

## List of Figures

---

2.1	Our application development architecture defines key components and a software engineering process to facilitate the development of apps specifically aimed at deployment within a digital learning environment. . . . .	31
2.2	The Graasp ecosystem consists of four interfaces, each addressing a different aspect of the digital learning experience. (Adapted from Gillet, Vonèche-Cardia, et al. (2022).) . . . . .	33
2.3	Graasp Builder is aimed at educators looking to create learning activities with resources from across the web. . . . .	34
2.4	Once an activity is ready, a link to access it via Graasp Player can be shared with learners. (Comic strip by Munroe (2015).) . . . . .	34
2.5	A simple template can serve to bootstrap an app on Graasp. . . . .	35
2.6	By default, a more advanced template provides developers with insight into how data is generated and stored on Graasp. . . . .	35
2.7	Creating an app is facilitated by the CLI's new command, which fetches a template and bootstraps a local development environment. . . . .	36
2.8	An app created with the Graasp Application Development Kit—which is an implementation of our application development architecture—can be featured on the Graasp Library, a repository of open educational resources. . . . .	37
2.9	Four example apps (clockwise from top left): <i>Sticky Notes</i> for design thinking, <i>Task Management</i> for collaboration (La Scala et al., 2022), <i>Greenhouse Effect</i> for climate change awareness, and <i>Light Pollution Simulator</i> to visualize the effects of light pollution (Gomes et al., 2019). . . . .	38
3.1	The user flow in our architecture involves five main stages. First, there is a request for experiment, followed by consent gathering, data gathering, data management, and finally collaboration. The tools that we propose support stakeholders across all stages and are highlighted in blue. . . . .	47
3.2	The <i>Experiment Tool</i> and the <i>Consent Management Tool</i> of our architecture were implemented within the <i>Membership via Consent</i> panel on Graasp. . . .	48
3.3	A web dashboard serves to visualize the learning analytics data collected by the educational platform, providing high-level insights into a learning activity's usage for technical and nontechnical users alike. . . . .	50
3.4	The <i>Datasets</i> tab, where users load the datasets they want to work with. A schema has been generated for each of these datasets, which are tagged with that schema's label. . . . .	51
3.5	The <i>Algorithms</i> tab, where users can add, edit, and delete pre-composed or custom anonymization algorithms. . . . .	53
3.6	The <i>Executions</i> tab, where users can select the datasets they have imported and run pre-composed or custom algorithms against them. . . . .	53
3.7	A built-in JSON editor allows users to view and edit original and anonymized datasets within the application. Datasets can also be exported to be viewed and edited using external tools. . . . .	53

3.8	The <i>Validations</i> tab, where users can run anonymity verification algorithms against the resulting datasets and see the outcomes of the tests (success, warning, or failure). . . . .	53
3.9	Users can add their own custom algorithms using the built-in code editor. Algorithms can also be loaded from a file. . . . .	54
4.1	A computational notebook learning activity on Graasp. Students can write and execute code, navigate a virtual file system, and view graphics. . . . .	67
4.2	Feedback feature in the learning platform. (1) Learners write code in the app. (2) Educator gives feedback to the learners. (3) Learners can view the feedback received. . . . .	68
4.3	A learning analytics dashboard to track how learners interact with an activity.	69
4.4	Self-reported programming experience at the beginning of the study was not a predictor of the code interaction metric. . . . .	73
4.5	There were no significant differences in the distribution of the code interaction metric by gender. . . . .	73
4.6	Comparing blended (A, B) and distance (C) learning scenarios. The educator's activity is shown on a thick line above and each thin line represents a student.	74
4.7	Days in which there was recorded activity on Code. Each horizontal line represents a student and each block represents a day in which the student recorded an interaction. A change in use is evident after the switch to distance learning.	75
4.8	How much a student interacted with Code in the blended learning scenario predicted how the student would interact in the remote learning scenario. . .	76
4.9	The code interaction metric and time spent watching videos metric presented a strong positive correlation. . . . .	76
5.1	Educators can configure Code Review to display syntax-highlighted code in four different programming languages. . . . .	83
5.2	The educator interface of the app provides an overview of how learners have interacted with the code snippet. . . . .	83
5.3	Code Review includes a Markdown editor that allows both educators and learners to annotate the code snippet with rich text and images. . . . .	84
5.4	Comments can be edited, deleted, replied to, or hidden. . . . .	84
5.5	In this example code review notebook, the lesson consists of seven phases that can be explored using a vertical navigation bar. This figure highlights the <i>Getting Started</i> phase, which uses Code Review to present students with an initial exercise. . . . .	87
5.6	Code quality issues were explained using text accompanied by a code snippet highlighting the issue and another illustrating a possible solution. . . . .	88
5.7	Boxplots showing the differences between the pre- and post-test scores. . . .	91
5.8	Histogram depicting the distribution of learning gains achieved. Students are grouped by percentage achieved in steps of 10%. . . . .	91

## List of Figures

---

5.9	While the distribution of students' self-reflection sentiment scores was wide, it shows a tendency towards negative scores. . . . .	92
5.10	There is a negative relationship between learning gains and self-reflection sentiment scores, but learning gains did not significantly predict sentiment scores. . . . .	92
5.11	The distribution of sentiment scores for the feedback provided was predominantly positive. . . . .	93
5.12	While there is a slightly positive relationship between learning gains and feedback sentiment scores, learning gains did not significantly predict sentiment scores. . . . .	93
5.13	Features requested for Code Review ordered by mean score (top to bottom). .	94
6.1	Our chatbot interface was configured to resemble an interactive grammar exercise. . . . .	108
6.2	A sample interaction with the chatbot for the control condition using the funniest phrase (CP2). Users provided their responses via the buttons at the bottom of the chat window. . . . .	113
6.3	A sample interaction with the chatbot for the verbal and emoji treatment using the funniest phrase (T3P2). Note that the chatbot's responses are not affected by the user's input. . . . .	113
7.1	There are eight emoji reactions available on GitHub. . . . .	120
7.2	Proportion of reactions received—by reaction type—for both human and bot comments. . . . .	123
7.3	Proportion of reactions received—by reaction type—for a selection of popular bots. . . . .	124
7.4	Example of the laugh reaction to a bot comment being caused by an incongruity. Although the issue is clearly spam, vscodebot [bot] suggests that it could be a duplicate of something completely unrelated, possibly due to the presence of the word <i>support</i> . . . . .	126
8.1	The new Code Review app allows students to reply to comments seeded by the educator, leading to thread-like exchanges. As depicted here, the educator can impersonate a chatbot when annotating the code snippet. . . . .	131
8.2	Educators can create different bot identities that they can then impersonate when interacting with students. . . . .	132
8.3	Educators can impersonate chatbots to seed a code snippet with comments or to provide feedback to students. . . . .	133
8.4	We prepared a lesson introducing students to the concept of code linting in JavaScript. . . . .	134



8.5	The <i>Examples</i> phase of the learning scenario varied across experimental conditions. This figure depicts the interface for the chatbot condition, showcasing an example code snippet with linting errors that have been highlighted by comments from an instructor impersonating <i>Lint Bot</i> . . . . .	135
8.6	Box plots summarizing the descriptive statistics across conditions for the three aspects considered by our quantitative analysis: usability (top), engagement (middle), and learning gains (bottom). . . . .	138
9.1	Through the <i>Auto Bot Setting</i> feature, educators can upload a script defining the rules that the chatbot would follow or edit these rules directly using a built-in editor. . . . .	145
9.2	Our chatbot interaction script was configured using the JSON format (left) and supported a rule-based conversational flow comprising several potential pathways (right). . . . .	146
9.3	Educators could also select to show/hide the chatbot to/from a particular set of learners. . . . .	147
9.4	Our chatbot was integrated into a code review application that was embedded in a code review notebook aimed at teaching Python programming. . . . .	149
9.5	Some of the chatbot's comments featured animated gifs (left) and emojis (right) to elicit humor and express surprise, respectively. . . . .	150
9.6	Learning gains were positive for both groups, but there were no significant differences across conditions. . . . .	151
9.7	On average, students rated how relevant they found each guideline and the overall relevance of the code style exercises positively (above the median rating of 4), but there were no significant differences across groups. . . . .	152
9.8	User Experience Questionnaire (UEQ) ratings were consistently higher for the treatment group, especially for the efficiency ( $p = 0.0127$ ), dependability ( $p = 0.0565$ ), and perspicuity ( $p = 0.0807$ ) dimensions. . . . .	153
10.1	Our architecture comprises six loosely-coupled components that interact through three main processes. . . . .	159
10.2	The cue is used to invite learners to interact with the chatbot. . . . .	163
10.3	The Code Capsule app can be used to write and execute (left) or to review code (right). The code used in these examples has been adapted from the Python Matplotlib library's documentation (Hunter, 2007). . . . .	165
10.4	The cue consisted of a perk explaining the issue present in the code snippet and a hook asking students whether they agreed with the solution proposed. . . . .	166
10.5	The lesson used for this study consisted of an improved version of the code review notebook used in the studies presented in Chapters 5 and 9. (Comic strip by Munroe (2015).) . . . . .	167
10.6	Learning Gains . . . . .	169
10.7	Overall Time Spent to Complete the Lesson . . . . .	169
10.8	Time Spent per Phase . . . . .	170

## List of Figures

---

10.9	Sentiment Analysis on Responses Capturing Self-Reflection . . . . .	171
10.10	Sentiment Analysis on Responses Capturing Feedback . . . . .	171
10.11	Results of the User Experience Questionnaire (UEQ) . . . . .	173
11.1	Overview of the TRACE model including its components and the tasks associated with each stakeholder. The thick line represents how the stakeholders can then <i>trace</i> a line through the components they select for their design. . . . .	185
11.2	Before being exposed to the model, participants were asked to describe how they would integrate a chatbot into a learning activity presenting the history of their place of birth. . . . .	194
11.3	Participants in the pilot workshop study were invited to share how they had defined the different components of our model using the Sticky Notes app on Graasp. . . . .	195
11.4	TRACE was used to brainstorm the design of educational chatbots during a semester-long pilot case study. Outcomes from these sessions included diagrams similar to the one depicted above, which was produced by an undergraduate student completing a semester project on designing educational chatbots. . . . .	196
11.5	Responses regarding how groups defined the different components of the TRACE model can be illustrated with word clouds to highlight important keywords. . . . .	199
12.1	The learning application hosts the learning resource (pink) that will be the focus of the interaction and serves as the context for our blueprint. . . . .	206
12.2	When summoned, the chatbot performs a function and opens channels (light purple) through which it can hold an interaction with the learner. . . . .	206
12.3	A learner can respond (orange) to the chatbot's comments to partake in the interaction that the chatbot has selected for a given channel. . . . .	206
12.4	Our blueprint consists of multiple loosely coupled components, comprising both engines, which are tasked with executing processes, and models, which keep track of the system's state. Here, we illustrate how the different components and processes come together to support the interaction between a learner and a chatbot in the context of a learning activity. . . . .	211
12.5	Before any interaction takes place, the app features a text snippet selected by the educator. . . . .	214
12.6	When summoned, the chatbot highlights the keywords in the text. . . . .	214
12.7	Clicking on a highlighted keyword opens a channel in which a learner can hold an interaction with the chatbot. . . . .	214
12.8	The educator can configure the app and the chatbot interaction using Graasp's educator interface. . . . .	215
12.9	The learning task focuses on a snippet of text that is provided by the educator. . . . .	216

12.10	When the learner clicks on the <i>Highlight Keywords</i> button, the summoning process is executed and the chatbot highlights the keywords selected by the educator. . . . .	217
12.11	Once the channels are open, the learner and the chatbot can hold interactions about the keywords. These interactions are powered by OpenAI's API. . . . .	218
13.1	An Overview of Our Research Context, Research Opportunities, Problem Statement, Research Questions, Objectives, and Contributions . . . . .	222
13.2	A Visual Representation of Our Conceptual Framework . . . . .	228



## List of Tables

3.1	Features Requested to Support Research on Digital Education Platforms (Adapted from Machado et al. (2019).)	45
4.1	Structure of the Information Technologies Course	70
6.1	Phrases Selected for the Experiment	110
6.2	Experimental Setup	111
6.3	Ten Traits Rated by Participants in the Post-Questionnaire	114
6.4	Means and Standard Deviations for Five-Point Likert Scores by Trait and Phrase	115
7.1	GitHub Bots that Received Reactions from the Highest Number of Distinct Users	124
8.1	Descriptive Statistics for Each Condition and ANOVA Results Across Conditions	137
9.1	Weekly lecture topics included in the Python programming component of the course alongside their corresponding lab session and code style guidelines covered therein.	148
10.1	UEQ Results Compared to Benchmark Usability Scores	173
12.1	An Overview of Our Blueprint's Key Components and Building Blocks Alongside the Functionalities They Support	208



# Acronyms

**ADA** Application Development Architecture. xiv, 13, 14, 19, 23, 24, 27–29, 31, 34, 37, 38, 41, 159, 161, 179, 218, 219, 224

**ADK** Application Development Kit. 15, 27–31, 34, 39, 40, 43, 224

**AI** Artificial Intelligence. 3, 4, 7, 8, 193, 221

**ANOVA** Analysis of Variance. 111, 137, 139

**API** Application Programming Interface. 29, 30, 36, 49, 67, 86, 159–162, 166, 175, 195, 213, 215

**CASA** Computers Are Social Actors. 106, 107, 189

**CLI** Command-Line Interface. 28–32, 35, 36, 40, 67, 136

**CMC** Computer-Mediated Communication. 108, 111

**DBR** Design-Based Research. 12–14, 19

**ESL** English as a Second Language. 107, 118

**Graasp ADK** Graasp Application Development Kit. xiv, 34–40, 66, 79, 82, 101, 159, 195, 214, 218, 224

**GUI** Graphical User Interface. 46, 207, 209, 212

**HCI** Human-Computer Interaction. 3, 6, 8, 106, 116, 204, 221

**ID** Identifier. 30, 52, 146

**JSON** JavaScript Object Notation. 49–52, 145

**LA** Learning Analytics. xiv, 4, 15, 23–26, 30, 40, 43, 44, 46–50, 55, 56, 65–67, 86, 90, 94, 136, 160, 161, 179, 224

**LACE** Learning Analytics Community Exchange. 24

## Acronyms

---

- LLM** Large Language Model. 7, 38, 101, 104, 118, 155, 157–159, 161, 162, 174–176, 179, 200, 221, 225, 226, 229
- LMS** Learning Management System. 4, 23, 71
- LXP** Learning Experience Platform. 4–6, 9, 11–16, 23, 24, 27, 28, 30–33, 43, 46, 64–68, 79, 81, 84, 93, 109, 117, 131, 157–161, 173, 175, 195, 203, 213, 221, 223, 224, 226, 230
- NLP** Natural Language Processing. 7, 8, 158, 209
- OER** Open Educational Resource. xiv, 8, 24, 31, 32, 34, 37, 160
- ROFL** Rolling on the Floor Laughing. 110, 112
- SemVer** Semantic Versioning. 31
- STEM** Science, Technology, Engineering, and Mathematics. 1, 62, 69
- SUS** System Usability Scale. 72, 89, 90, 136
- UEQ** User Experience Questionnaire. xviii, xxi, 151, 152, 154, 155, 167, 168, 172–174
- UEQ-S** Short Version of the User Experience Questionnaire. 89, 90, 151
- UX** User Experience. 94



# 1 Introduction

CONVERSATIONAL agents, commonly known as *chatbots*, are computer programs that can interact using natural language (Abu Shawar et al., 2007a). Conceptualized by Alan Turing (1950), chatbots have evolved from early keyword matching implementations such as *ELIZA* (Weizenbaum, 1966), to pattern matching agents such as *A.L.I.C.E.* (R. S. Wallace, 2009), to systems powered by large language models trained on corpora comprising billions of words, including Google’s *Meena* (Adiwardana et al., 2020), Facebook’s *BlenderBot* (Shuster et al., 2022), and—most notably—OpenAI’s *ChatGPT* (OpenAI, 2022). Nowadays, chatbots come in a variety of architectures (Cahn, 2017) and permeate a wide range of application domains, including e-commerce (Cui et al., 2017), banking (Trivedi, 2019), and healthcare (Laranjo et al., 2018).

The use of chatbots in educational settings has also been rising steadily over the past decade. Recent surveys have shown active research on the development and use of chatbots for education (Quiroga Pérez et al., 2020). These chatbots aimed at educational settings—also referred to as *educational chatbots*—have been incorporated into applications aimed at several subjects, spanning science, technology, engineering, and mathematics (STEM), as well as languages, business, and the arts (Hwang et al., 2021). Able to assume different pedagogical roles (Wollny et al., 2021), educational chatbots have the potential to support educators with various tasks, including answering frequently asked questions (Quiroga Pérez et al., 2020), encouraging learning (Graesser et al., 2005), and providing personalized tutoring (Clarizia et al., 2018).

To assess the value that educational chatbots can add to the learning experience, several empirical studies have been conducted, often employing questionnaires to evaluate learners’ perceptions of these chatbots (Quiroga Pérez et al., 2020). The results of these empirical studies are promising, suggesting that educational chatbots can help teach basic concepts and provide educational resources to students (Chen et al., 2022), encourage students’ social presence online (Huang et al., 2022), and serve to scaffold programming exercises (Winkler, Hobert, et al., 2020). Nevertheless, there is ample need for more field research to ground the use of chatbots in education on solid results. Indeed—as recently as 2021—researchers have

raised the issue that there are “few empirical studies investigating the use of effective learning designs or learning strategies with chatbots” (Hwang et al., 2021). Conducting these empirical studies, however, is not trivial. Designing, implementing, deploying, and evaluating a chatbot within an educational context requires expertise in multiple domains (Okonkwo et al., 2021), often involving several stakeholders (Durall Gazulla et al., 2023), including software developers, educators, educational researchers, and—of course—learners.

One strategy to address this challenge has been to adopt a participatory design methodology (Kensing et al., 1993) for designing educational chatbots (Durall et al., 2020; Gabrielli et al., 2020). Participatory design advocates for the inclusion of end users in the design process and involves collaboration between all stakeholders, which is often facilitated through workshops (Robertson et al., 2012). In education, participatory design has been harnessed to create digital technologies and learning spaces, as well as to explore issues of student agency and empowerment (Cumbo et al., 2022). However, as highlighted by Durall Gazulla et al. (2023), the collaborative design (or *co-design*) process for educational chatbots—and educational technologies in general—entails a particular set of challenges, including the diverse needs of different stakeholders and the difficulties associated with going from abstract ideas to concrete implementations. In light of these challenges, Durall Gazulla et al. “advocate that researchers, designers and developers engage in a collective reflection on big and small decisions in collaborative design, and co-create strategies to overcome the obstacles that hinder collaboration and mutual understanding when designing learning technologies”.

Moreover, although technical frameworks for integrating chatbots into educational contexts exist (Griol et al., 2013; Sjöström et al., 2018; Villegas-Ch et al., 2020), there is no standard approach to building and deploying chatbots for these contexts. As highlighted by Quiroga Pérez et al. (2020), “there exists as much technology used in the development of chatbots as there are educational chatbots”. This lack of standard adds a layer of complexity when translating the outcomes of design sessions into software solutions that can be deployed and evaluated. This challenge was also echoed by Følstad, Araujo, et al. (2021), who highlighted the need for solutions to support chatbot development.

Therefore, while there is a tangible need to study the impact of integrating chatbots into educational contexts, we currently lack the support tools to efficiently and systematically design and deploy these chatbots. This need was neatly captured by a recent survey of the principles grounding the design of educational chatbots, which emphasized that “researchers should explore devising frameworks for designing and developing educational chatbots to guide educators to build usable and effective chatbots” (Kuhail et al., 2023). Our work aims to address this gap by developing a conceptual framework that can serve to scaffold the integration of educational chatbots into digital education platforms.

The remainder of this chapter is structured as follows. Section 1.1 introduces the research context that informs our problem statement, namely that there is a lack of conceptual and technical design tools for integrating educational chatbots into learning contexts. We motivate

and formally define this problem statement in Section 1.2. In Section 1.3 we formulate the research questions that emerge from our problem statement and map them to the corresponding objectives that we address in this thesis. Section 1.4 presents the research methodology we followed to approach these questions and objectives, while Section 1.5 provides a summary of our contributions, as well as how they were validated and disseminated to the research community. Finally, in Section 1.6, we outline the overall structure of this thesis.

## 1.1 Research Context

The topics covered in this thesis span different disciplines, including (i) *digital education*, (ii) *software engineering*, (iii) *human-computer interaction (HCI)*, and (iv) *artificial intelligence (AI)*. Our work stems from specific areas within each of these disciplines—learning experience platforms, social coding platforms, conversational agents, and large language models, respectively—and focuses on the intersections between these areas. As shown in Figure 1.1, these four disciplines and their intersections serve as the pillars that underpin our focus on four more specific research domains: (i) *educational application development*, (ii) *software engineering education*, (iii) *educational chatbots*, and (iv) *chatbot design*. Related work conducted in each of these four specific domains will be presented at the beginning of each of the following four parts of the thesis, which respectively address the parts of our research corresponding to the domain in question.

In this section, we provide an overview of how the four underlying pillars serve as the basis for the research conducted in this thesis. For each pillar, we highlight the research opportunities that ultimately informed our problem statement, which we will present in Section 1.2.

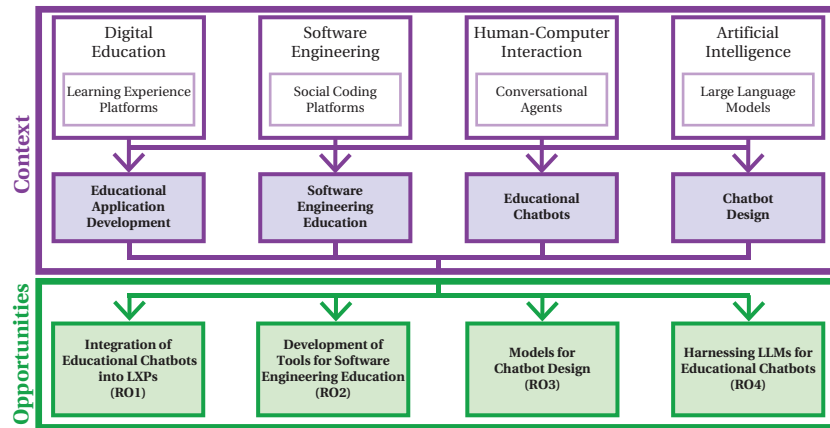


Figure 1.1: Our work is underpinned by specific areas of research spanning four disciplines. The intersections of these areas give rise to four more specific domains of research (shown in **bold**) that guide the approach undertaken in this thesis. The four guiding research opportunities identified (**RO1**, **RO2**, **RO3**, **RO4**) span the four research domains underpinning our work. Further opportunities emerge in the intersections of these guiding opportunities.

### 1.1.1 Digital Education

Our research is primarily framed by recent advances in digital education, where a myriad of technological solutions has been proposed in response to novel teaching practices. One of the key trends in education over the last decade has been the shift to using blended learning models, where traditional face-to-face learning is complemented by digital interactions, in class or at distance (Adams Becker et al., 2017). The importance of these interactions is mainly informed by the success of *active learning* in pedagogical practices (Prince, 2004). Active learning calls for engaging students in the learning process through strategies that go beyond merely a one-way transmission of information. While there is no “identifiable origin or a common definition” for the term *active learning*, these strategies often involve placing an emphasis on developing skills, higher-order thinking, integrating activities, and encouraging self-reflection (Bonwell et al., 1991).

The rise of blended and online learning has also influenced the availability and types of digital platforms aimed at supporting active learning through technological solutions. One advantage of using digital technologies for blended learning is the opportunity to integrate *learning analytics (LA)* into the instructor’s awareness and reflection processes. LA has been defined by the Society for Learning Analytics Research (SoLAR) as “the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs” (Ferguson, 2012). Indeed, LA has been shown to have concrete practical applications in the learning process, including as potential early predictors of student success or failure (Van Goidsenhoven et al., 2020). These practical applications have motivated LA-driven approaches to designing learning activities, which most often take place on digital learning platforms (Mangaroska et al., 2019),

One type of platform, the *learning experience platform (LXP)*, has embodied this data-driven approach and gained considerable traction in recent years. Learning experience platforms depart from the traditional structure of *learning management systems (LMSs)* and online courses to provide more flexibility to both learners and educators. As such, learning experience platforms (LXPs) are more focused on the personalization of the learning experience, allowing the integration of nontraditional learning material and acting as content aggregators for both instructors and students (Valdiviezo et al., 2020). This emphasis on personalization makes LXPs an ideal environment for recommender systems and other AI techniques that can tailor the content to the needs and expectations of different learners (Ritz et al., 2022). Educational chatbots align themselves closely with this approach to the learning experience, as they have the potential to support learners through personalized conversational scripts.

Another strength attributed to LXPs is their support for interactive content. This interactivity is often achieved through the inclusion of embeddable web applications. Through these interfaces, learners can actively interact with both their peers and their educators, contributing to “a social media type stream of information” that also supports the content creation process (Cockrill, 2021). The fact that chatbots interact with learners through natural language—a

key element in social interaction—could also ease their integration into LXPs.

### Research Opportunity

Our first research opportunity emerges in the context of digital education, namely the possibility of studying the integration of educational chatbots into LXPs (**RO1**).

#### 1.1.2 Software Engineering

A second domain of research framing our work is the domain of software engineering. More specifically, our work builds on software engineering best practices, how they are supported by conversational agents, and how they can be incorporated into educational settings. Although there are several definitions of what *best practices* entail in the context of software engineering (Marques et al., 2018), conducting peer reviews for software verification and validation is one of the skills highlighted by the IEEE Computer Society and Association for Computing Machinery's Joint Task Force on Computing Curricula (2015). Indeed, code reviews are an integral part of the software development process (Wiegers, 2002) and while there are several tools to support code reviews (Baum et al., 2016; Hedberg, 2004), most collaborative software development involving the code review process currently takes place on social coding platforms.

Well established in the software engineering industry, social coding platforms (e.g., GitHub, GitLab, Bitbucket) are collaborative workspaces focused on software development (Dabbish et al., 2012). These platforms are based on version control software and provide bespoke interfaces to support reviewing code, leaving comments, visualizing changes, and carrying out other related tasks. Chatbots can help with some of these tasks and have thus become a common feature of the social software development process (Lebeuf et al., 2018), helping to reduce manual labor, improve code quality, and increase productivity (Abdellatif et al., 2020; Peng et al., 2019; Wessel, Serebrenik, et al., 2020a, 2020b). A study by Wessel, de Souza, et al. (2018) revealed that nearly 26% of popular open-source projects on GitHub used chatbots. Moreover, a significant proportion of these chatbots were involved in tasks related to the code review process. That is, among the 48 chatbots identified in the study, 14 chatbots (29.2%) were used to review code and pull requests, highlighting the suitability of chatbots for code review tasks on social coding platforms. Although these chatbots are principally designed to support development tasks, they are nonetheless interactive, communicating with human users via the various discussion interfaces available on GitHub.

### Research Opportunity

Building on **RO1**, a second research opportunity that emerges at the intersection of digital education and software engineering is the possibility of developing tools to support software engineering education (**RO2**). More specifically, there is an opportunity at the intersection of

**RO1** and **RO2** to make these tools compatible with LXPs and capable of integrating educational chatbots.

### 1.1.3 Human-Computer Interaction

A third pillar underpinning our research is the domain of human-computer interaction (HCI). Given that chatbots are interactive, user-facing systems, researchers agree that the design and evaluation of such systems involve both their technical and social aspects (Feine, Gnewuch, et al., 2019). The study of how these social aspects affect the way users interact with and perceive chatbots is usually framed by the *Computers Are Social Actors* (CASA) paradigm (Nass, Steuer, et al., 1994). According to CASA, user interactions with computers are fundamentally social in nature and, therefore, many social conventions that guide interpersonal behavior are also evident in HCI (Fogg et al., 1997; Nass, Moon, et al., 1997).

The CASA framework posits that interactive computer interfaces—of which chatbots are a prime example—are regarded as social actors and that this effect becomes even more pronounced when anthropomorphic traits are conveyed by the interface (Moon, 2000). As highlighted by Feine, Gnewuch, et al. (2019), chatbot design is often aligned with the implications put forth by CASA, with chatbots harnessing a wide array of social cues (e.g., eye movement, photorealism, small talk) in their interactions. In fact, several studies (Moon, 2000; Schanke et al., 2021) have discussed the impact of anthropomorphic cues on HCI. A recent study by Go et al. (2019) specifically investigated the effects of visual, identity, and conversational cues on humanness perceptions of chatbots, and analyzed the correlations between the cues.

In the context of conversational agents, De Angeli et al. (2001) defined personality as a stable set of traits that determines the agent's interaction style, describes its character, and allows the end user to understand its general behavior. Shum et al. (2018) claimed that social chatbots need to present a consistent personality to gain the confidence and trust of users. Furthermore, in a study conducted by Jain et al. (2018), several participants found humorous chatbots with distinct personalities more engaging and enjoyable, while Brandtzaeg et al. (2017) argued that entertainment and socialization may be seen as aspects of the relationship between humans and chatbots, since nearly 20% of their study's participants reported using chatbots for entertainment. These results highlight how a chatbot's design and interaction style can have an impact on user experience, making user interactions more enjoyable and fulfilling.

Nevertheless, computers expressing human-like traits can arouse anthropomorphic perceptions that could lead to unintended consequences (Duffy, 2003), such as setting unattainable expectations in their users (Schanke et al., 2021) or triggering the uncanny valley effect. This effect—identified by Mori (2012)—predicts that as artificial agents approach human-like resemblance, imperfections in this resemblance might evoke negative reactions. A recent psychophysiological study by Ciechanowski et al. (2019) observed the uncanny valley effect when comparing a text-based chatbot to a human-like avatar. Their results link this effect to more intense emotional reactions, as evidenced by data from electromyography, instant-

neous heart rate, and electrodermal activity. The effect was also negatively correlated with how competent participants found the chatbot to be, further highlighting the perils of the uncanny valley on users' perceptions of chatbots. In light of these findings, the authors proposed that as “bots become increasingly more popular in the professional and personal sphere, the task of understanding how they are perceived and what drives [these perceptions] becomes urgent and necessary” (Ciechanowski et al., 2019). Assessing the extent and effect of such perceptions—especially in educational settings that often include young learners—is therefore imperative and is a key motivating factor behind our empirical studies and the development of our framework.

### Research Opportunity

The need to better understand human-chatbot interaction highlights a third research opportunity that builds on **RO1** and **RO2**, namely the possibility of creating models to guide a chatbot's design (**RO3**) and evaluate how different designs can affect users' experiences with chatbots in educational contexts, software engineering, and software engineering education. The intersection of these three key research opportunities provided a central motivating factor for the approach undertaken in this thesis.

#### 1.1.4 Artificial Intelligence

The final pillar grounding our work is the domain of artificial intelligence (AI). Although the scope of AI is very broad, our research focuses on advances in methods from natural language processing (NLP), and particularly on the success of large language models (LLMs) in a variety of language-based tasks.

LLMs fall into a general class of AI models referred to as *foundation models*, which are “trained on broad data at scale and can be adapted (e.g., fine-tuned) to a wide range of downstream tasks” (Bommasani et al., 2021). The latest generation of these models is based on the Transformer architecture (Vaswani et al., 2017), a deep learning architecture that allows—most crucially—for increased parallelization during training. The process of training these models has also benefited from advances in self-supervised learning, whereby the training tasks are derived from the training data itself and do not require any annotations. An example of this type of task would be to mask a word within a sequence and ask the model to predict the missing word (e.g., *it is raining \_\_\_\_\_ and dogs*). The ability to harness unannotated data alongside powerful parallelization that can make use of cutting-edge hardware has paved the way for language models to be trained on increasingly larger datasets and incorporate a larger number of parameters. The GPT-3 model, for instance, comprises 175 billion parameters (T. Brown et al., 2020).

While LLMs are used for a wide array of language tasks, they can also be used to power chatbots. As mentioned at the beginning of this introduction, notable examples of LLM-powered

chatbots include Google's *Meena* (Adiwardana et al., 2020), Facebook's *BlenderBot* (Shuster et al., 2022), and OpenAI's *ChatGPT* (OpenAI, 2022). Without any further configuration, these open-domain chatbots are able to interact with users about diverse topics through unrestrained conversation. Nevertheless, understanding how these models can be harnessed to create *task-oriented* chatbots aimed at specific domains is still ongoing.

A recent position paper by Kasneci et al. (2023) outlined opportunities and challenges of incorporating LLMs in education. Among the challenges identified, the authors highlight (i) the possible biases that LLMs can perpetuate and amplify, (ii) the need for open educational resources (OERs) to guide educators on how to access and use these models, (iii) the need to ensure data privacy and security, and (iv) the lack of adaptability to align the models with the objectives of individual learners and educators. Addressing these challenges could open the door to more powerful applications of LLMs in education. Finally, Kasneci et al. call for further research at the intersection of AI and HCI, to ensure that the interfaces used to interact with these models are aligned with the needs of different types of learners (e.g., age-related differences, accessibility requirements).

Another important factor to consider when assessing the applicability of these LLMs to education is the process through which they can be configured for more specific (*downstream*) tasks, such as generating the responses that an educational chatbot can use in its interactions. This process can be supported by *prompting*, which consists in providing the model with a few examples of how it should respond to a query. Finding the most appropriate prompts is a complex task and using inefficient prompting strategies can result in worse results than using no prompt at all (Reynolds et al., 2021). Identifying optimal prompts—or *prompt engineering*—is an active area of research in NLP. Recent work has focused on automating the generation of these prompts (Z. Jiang et al., 2020), understanding the biases that prompting can be susceptible to (T. Z. Zhao et al., 2021), and providing the appropriate infrastructure to generate prompts (Bach et al., 2022). Nevertheless, there is little guidance for performing prompt engineering with domain-specific applications in mind, as is the case with educational chatbots.

### Research Opportunity

An opportunity to integrate AI into our research arises at the intersection of **RO2** and **RO3**. That is, the possibility of exploring how to incorporate the functionalities offered by LLMs in the design of educational chatbots **RO4**, with a focus on how to appropriately configure these models through efficient prompting strategies. This opportunity also extends to the domain-specific case of designing chatbots for software engineering education, completing the full landscape of research opportunities that emerged from our four pillars, as depicted in Figure 1.1.



## 1.2 Problem Statement

The research context highlighted in Section 1.1 motivated our approach to the study of human-chatbot interaction in digital education by leading us to the intersections of the research opportunities illustrated in Figure 1.1. These research opportunities serve as a starting point for our investigations, which first focus on the case of integrating chatbots for software engineering education—and more specifically to support the code review process in educational settings—before generalizing to the domain of educational chatbot design.

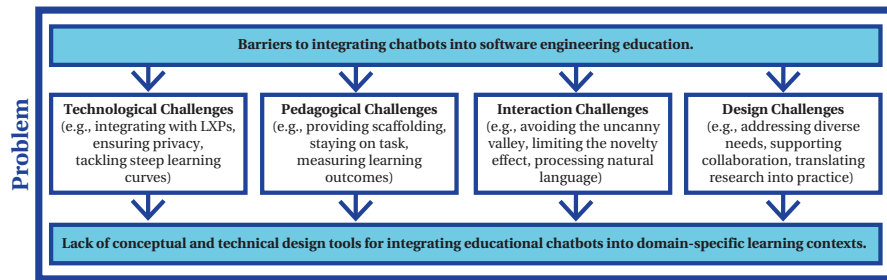


Figure 1.2: Our initial problem statement can be broken down into four types of challenges that in turn guide the formulation of our overarching problem statement.

As depicted in Figure 1.2, we start by considering the challenges that limit the development, deployment, and use of educational chatbots for software engineering education. Guided by the intersections of the research opportunities we identified, our preliminary problem statement can be formulated as the existence of *barriers to integrating chatbots into software engineering education*. Addressing this problem requires tackling some of the challenges highlighted in the literature. These challenges can initially be grouped into three categories: (i) technological challenges, (ii) pedagogical challenges, and (iii) interaction challenges.

Technological challenges encompass the lack of development frameworks and standards aimed at integrating applications and educational chatbots into LXPs. Building a custom chatbot architecture, for example, requires complex computer programming skills (Luo et al., 2019). If external platforms—such as social coding platforms—are used, there are also privacy considerations to address (Feliciano et al., 2016), as well as these platforms’ steep learning curves (Fiksel et al., 2019).

Pedagogical challenges are most evident when general-purpose chatbots are used for learning. To avoid the challenge of building custom chatbots from scratch, educational chatbots are often powered by technologies not necessarily aimed at education, such as *Textit* (Pottier, 2013), *Pandorabots* (R. Wallace, 2002), *Messenger* (Facebook, 2008), or *Dialogflow* (Google, 2012). While chatbots built with these technologies have been successfully applied in educational settings (Smutny et al., 2020), they often lack the pedagogical scaffolding required to easily design context-dependent dialog flows that can be integrated into formal learning. This lack of pedagogical scaffolding makes it difficult to ensure that the interaction between the chatbot

and the learner is focused on achieving a particular learning goal (Kumar, 2021) and stays on topic (Fryer et al., 2019)—a task that is particularly complex when using free-form dialog and natural language. Furthermore, whether chatbots are general-purpose or custom-built, there is always the challenge of extrapolating from the conversations held between a learner and a chatbot to empirical measures of learning gains or outcomes (Winkler & Soellner, 2018).

Finally, interaction challenges include avoiding the uncanny valley (Mori, 2012), addressing the novelty effect (Huang et al., 2022), and ensuring that chatbots exhibit consistent personality traits (Shum et al., 2018). There are also challenges associated with the different techniques that chatbots can use to harness natural language in their interactions. On the one hand, chatbots limited by rule-based scripts could be perceived as repetitive or be unable to provide relevant responses (Haristiani, 2019). On the other hand, chatbots harnessing large generative language models such as those powering ChatGPT (OpenAI, 2022) might provide a more natural user experience, but risk going off-topic (J. Edwards et al., 2021; Johansson, 2021) as well as replicating biases present in the data they are trained on (Bender et al., 2021). Finding the right balance between focus and naturalness requires consideration of the pedagogical and technological contexts in which the chatbot is interacting.

By breaking down our initial problem statement into these categories of challenges, we can generalize from the specific case concerning educational chatbots for software engineering education to the more general case of educational chatbot design. Indeed, while each of these challenges—or categories of challenges—could be independently tackled, an overarching problem is how to address these challenges holistically and in a way that satisfies the needs of all stakeholders involved. By *all stakeholders*, we refer to *educators*, *developers* and *researchers* in education, as well as *learners*. In considering this overarching problem, we identify a fourth category of challenges—(iv) design challenges—concerning the process of designing educational technologies.

As highlighted by Durall Gazulla et al. (2023), challenges related to the co-design process in education include—but are not limited to—(i) addressing the stakeholders' diverse needs, (ii) ensuring a certain level of technological literacy among all stakeholders, (iii) moving from abstract ideas to concrete implementations, and (iv) translating research into practice. These challenges span across the three aforementioned categories and are directly related to the need for a conceptual framework to guide how these stakeholders collaborate in the design of educational chatbots, as encapsulated in **RO3**. More concretely, a generalized problem statement can be formulated as the *lack of conceptual and technical design tools for integrating educational chatbots into domain-specific learning contexts*. This problem hinders the adoption of chatbots by educators, complicates the development of technical solutions by developers, and limits the access learners have to this technology. The aim of this thesis is to tackle this problem.

### 1.3 Research Questions and Objectives

In the previous section, we noted how the lack of conceptual and technical design tools to integrate chatbots into educational contexts poses a problem for the adoption of educational chatbots. In this section, we translate this problem into our overarching research question (**RQ★**) and then break this research question down into the four questions (**RQ1**, **RQ2**, **RQ3**, **RQ4**) that guided our approach to tackling this problem. We then work our way back to our overarching objective (**O★**) by first defining the partial objectives (**O1**, **O2**, **O3**, **O4**) that arise from each of the guiding research questions and then combining them to tackle **RQ★**. These research questions and objectives are outlined in Figure 1.3.

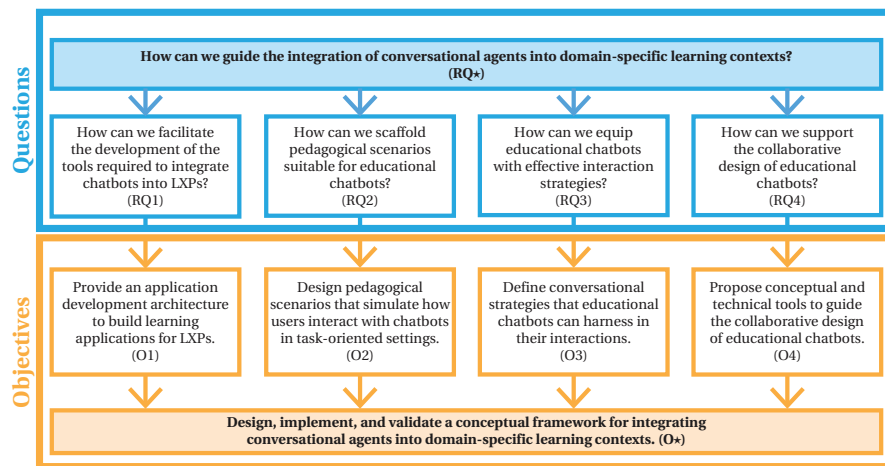


Figure 1.3: Our overarching research question (**RQ★**) was broken down into four questions that mapped onto four partial objectives, which in turn informed our overarching objective (**O★**).

Our approach to the aforementioned problem statement leads us to formulate our guiding research question.

- How can we guide the integration of chatbots into domain-specific learning contexts? (**RQ★**)

To approach this overarching research question, we first break it down into four—more specific—research questions. Each of these questions corresponds to one of the categories of challenges highlighted in Section 1.2.

- *Technological Challenges*: How can we facilitate the development of the tools required to integrate chatbots into LXPs? (**RQ1**)
- *Pedagogical Challenges*: How can we scaffold pedagogical scenarios suitable for educational chatbots? (**RQ2**)

- *Interaction Challenges*: How can we equip educational chatbots with effective interaction strategies? **(RQ3)**
- *Design Challenges*: How can we support the collaborative design of educational chatbots? **(RQ4)**

These research questions map, respectively, to the four partial objectives tackled in this thesis, which constitute the four dimensions addressed by our conceptual framework.

- Provide an application development architecture to build learning applications for LXPs. **(O1)**
- Design pedagogical scenarios that simulate how users interact with chatbots in task-oriented settings. **(O2)**
- Define conversational strategies that educational chatbots can harness in their interactions. **(O3)**
- Propose conceptual and technical tools to guide the collaborative design of educational chatbots. **(O4)**

By addressing these guiding research questions and objectives, we iteratively built and tested the technological, pedagogical, and interaction design tools that we could harness to create, deploy, and evaluate educational chatbots. Together, these four partial objectives served our overarching objective:

- Design, implement, and validate a conceptual framework for integrating conversational agents into domain-specific learning contexts. **(O★)**

The outcome of this objective is a complex design object comprising conceptual and technical tools that address four dimensions of educational chatbot design: (i) *technological foundations*, (ii) *pedagogical scenarios*, (iii) *interaction strategies*, and (iv) *design processes*. This conceptual framework aims to address the needs of all stakeholders and is particularly useful for educators looking to integrate educational chatbots into their practice, as well as for developers in education who need to provide the technical solutions to allow such integrations.

### 1.4 Research Methodology

The work carried out in the context of this thesis followed the methodology proposed by design-based research (DBR). DBR has been defined as “a systematic but flexible methodology aimed to improve educational practices through iterative analysis, design, development, and

implementation, based on collaboration among researchers and practitioners in real-world settings, and leading to contextually-sensitive design principles and theories” (F. Wang et al., 2005). This methodology supports the use of mixed methods over multiple iterations and has been particularly successful in framing the development of technological tools for learning across academic disciplines (Anderson et al., 2012).

While there are several approaches to DBR, we specifically followed Reinmann’s holistic DBR model, which was proposed for research and teaching in higher education (Reinmann, 2020). Reinmann’s model defines five *semantic fields* that constitute the base of the holistic DBR model: (i) *goal-setting (GS)*, (ii) *conceptualization (CO)*, (iii) *development (DE)*, (iv) *testing (TE)*, and (v) *analysis (AN)*. These fields are presented as a cycle, as shown in Figure 1.4.

Reinmann then defines three types of iterations of the DBR cycle. *Iteration type I* (see Figure 1.4) refers to the full cycle of the DBR process. *Iteration type II* (see Figure 1.5) breaks the cycle down into *fields of action*, defined as the segments between each pair of semantic fields. Within each of the five fields of action, smaller cycles can take place, capturing how there can be faster feedback processes between semantic fields. Finally, *iteration type III* considers not only pairs of adjacent semantic fields but triads, which Reinmann refers to as *playing fields*. Three *iteration type III* triads were used in this thesis: (i) *creative conceiving* (goal-setting ↔ conception ↔ development), (ii) *concrete developing* (conception ↔ development ↔ testing), and (iii) *practical testing* (development ↔ testing ↔ analysis). These cycles are depicted in Figure 1.6.

In short, the holistic model elegantly captures the intricacies of the DBR process when designing complex objects. Designing complex objects can require multiple cycles of the DBR process, each cycle tackling different parts that together constitute—or lead to—a primary design object (Tammeleht, 2022). In this thesis, the primary design object was defined by our overarching objective, namely to *design, implement, and validate a conceptual framework for integrating conversational agents into domain-specific learning contexts*. As outlined in Section 1.3, the approach to our overarching objective was to first tackle four partial objectives, each focused on a dimension of our primary design object. Hence, this approach required four iteration type I cycles. We label each cycle according to the dimension addressed: (i) *technological foundations*, (ii) *pedagogical scenarios*, (iii) *interaction strategies*, and (iv) *design processes*.

The technological foundations cycle tackled **O1**, aimed at providing an application development architecture (ADA) to build online learning applications for LXPs. The pedagogical scenarios cycle tackled **O2**, namely to design a pedagogical scenario that simulates how users interact with chatbots in a task-oriented setting. The interaction strategies cycle tackled **O3**, which consisted in defining different conversational strategies that educational chatbots can harness in their interactions. Finally, the design processes cycle tackled **O4**, proposing conceptual and technical tools to guide the collaborative design of educational chatbots.

Figure 1.7 illustrates how these four cycles constitute the main DBR process followed in this

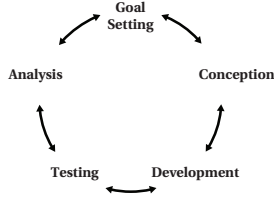


Figure 1.4: Iteration type I in the holistic DBR process refers to the full cycle comprising the five semantic fields. (Adapted from Reinmann (2020).)

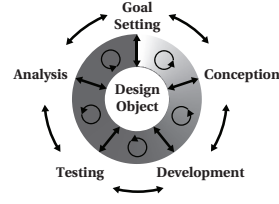


Figure 1.5: Iteration type II in the holistic DBR process breaks a full DBR cycle into five fields of action, which consist of shorter cycles between two semantic fields. (Adapted from Reinmann (2020).)

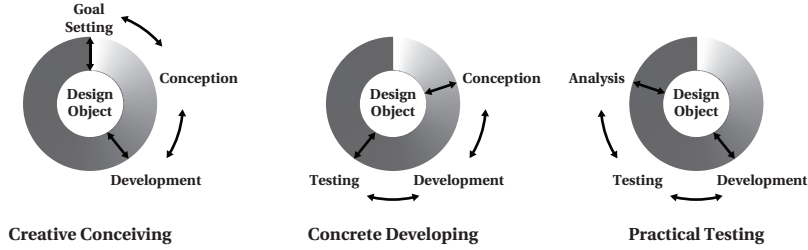


Figure 1.6: Iteration type III in the holistic DBR process breaks the full DBR cycle into playing fields, which refer to triads of adjacent semantic fields. The three iteration type III cycles used in this thesis are (i) *creative conceiving* (goal setting ↔ conception ↔ development), *concrete developing* (conception ↔ development ↔ testing), and (iii) *practical testing* (development ↔ testing ↔ analysis). (Adapted from Reinmann (2020).)

thesis. Within each of these cycles, depending on the objective, shorter iteration type II or iteration type III cycles served to tackle different aspects of the dimension in question. In the next section, we provide an outline of the contributions that were output throughout this process, as well as the evaluations through which these contributions were validated.

## 1.5 Contributions

Although our work was aimed at delivering our framework as an overarching contribution, each iteration of the DBR cycle resulted in one or more contributions that could also, independently, be of interest to the research community. In this section, we summarize the contributions put forth by this thesis. These contributions are also highlighted in Figure 1.8.

The main contribution (**C★**) of this thesis is a conceptual framework for integrating task-oriented conversational agents into digital education platforms. This framework comprises seven individual contributions (**C1**, **C2**, **C3**, **C4**, **C5**, **C6**, **C7**), which we describe below.

First, we designed an application development architecture to support the implementation of online applications that can be integrated into LXPs (**C1**). To evaluate this architecture, we

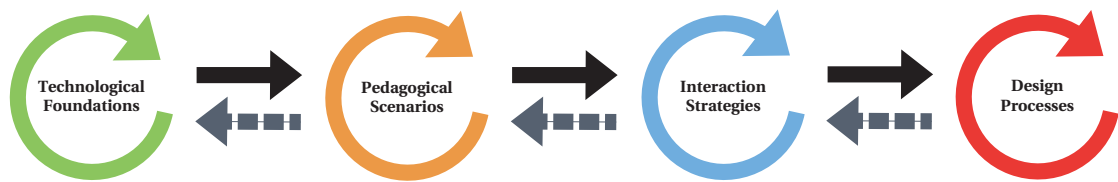


Figure 1.7: The main DBR process followed in this thesis comprised four cycles, each focusing on one of the objectives highlighted in Section 1.3. Our work mainly progressed incrementally from left to right (solid arrows), but outcomes from each phase also served to refine the work that had been conducted in previous phases (dashed arrows).

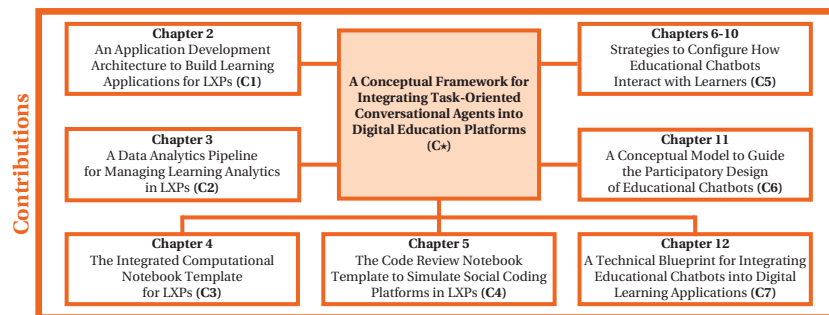


Figure 1.8: This thesis puts forth one overarching contribution (in **bold**) that comprises seven individual contributions.

implemented an application development kit (ADK) based on our design and conducted a qualitative study comprising semi-structured interviews with 12 developers who used the ADK. This architecture is presented in Chapter 2 and is featured in the following publication:

- Farah, J. C., Ingram, S., & Gillet, D. (2022). Supporting Developers in Creating Web Apps for Education via an App Development Framework. *HEAD'22 Conference Proceedings*, 893–890. <https://doi.org/10.4995/HEAD22.2022.15652>

Second, we outlined an end-to-end learning analytics pipeline to support the full data management cycle for research in education (**C2**). To validate the feasibility of integrating this pipeline into an LXP, we implemented a proof of concept and harnessed it to access the data produced by the applications used in our empirical studies. This data pipeline is presented in Chapter 3 and is featured in the following publication:

- Farah, J. C., Soares Machado, J., Torres da Cunha, P., Ingram, S., & Gillet, D. (2021). An End-to-End Data Pipeline for Managing Learning Analytics. *2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*. <https://doi.org/10.1109/ITHET50392.2021.9759783>

Third, we developed *Code*, an application to write and execute JavaScript and Python code directly in the browser. We used this application to scaffold learning activities following the *integrated computational notebook* template, a template aimed at providing browser-based computational notebook solutions within LXPs (C3). These activities served as the foundation for the development of the pedagogical scenario used in our empirical studies with chatbots. The design of this application and its evaluation with 67 university students is presented in Chapter 4 and is featured in the following publication:

- Farah, J. C., Moro, A., Bergram, K., Purohit, A. K., Gillet, D., & Holzer, A. (2020). Bringing Computational Thinking to non-STEM Undergraduates through an Integrated Notebook Application. In T. Broos & T. Farrell (Eds.), *Proceedings of the Impact Papers at the 15th European Conference on Technology-Enhanced Learning (EC-TEL 2020)*

Fourth, we designed the *code review notebook* template to simulate social coding platforms in educational contexts (C4). To validate this template, we conducted a case study that included an online lesson on code quality standards, which was completed by 25 university students. This case study focused on the effects that the code review notebook had on usability, learning gains, self-reflection, and feedback. The design of the code review notebook, as well as findings from our case study, are presented in Chapter 5 and are featured in the following publications:

- Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Toward Code Review Notebooks. *2022 International Conference on Advanced Learning Technologies (ICALT)*, 209–211. <https://doi.org/10.1109/ICALT55010.2022.00068>
- Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2023). Integrating Code Reviews into Online Lessons to Support Software Engineering Education. In M. E. Auer, W. Pachatz, & T. Rüütman (Eds.), *Learning in the Age of Digital and Green Transition* (pp. 815–826). Springer. [https://doi.org/10.1007/978-3-031-26190-9\\_84](https://doi.org/10.1007/978-3-031-26190-9_84)

Fifth, we defined strategies to configure how educational chatbots interact with learners (C5). We validated these strategies incrementally through an online experiment, an observational study, one pilot case study, and three in-class field experiments with students at the University of Neuchâtel and the School of Engineering and Architecture of Fribourg, in Switzerland. These studies are presented in Chapters 6–10 and have been featured in the following publications:

- Farah, J. C., Sharma, V., Ingram, S., & Gillet, D. (2021). Conveying the Perception of Humor Arising from Ambiguous Grammatical Constructs in Human-Chatbot Interaction. *Proceedings of the 9th International Conference on Human-Agent Interaction (HAI '21)*, 257–262. <https://doi.org/10.1145/3472307.3484677>
- Farah, J. C., Spaenlehauer, B., Lu, X., Ingram, S., & Gillet, D. (2022). An Exploratory Study of Reactions to Bot Comments on GitHub. *2022 IEEE/ACM 4th International Workshop on Bots in Software Engineering (BotSE)*. <https://doi.org/10.1145/3528228.3528409>



- Farah, J. C., Spaenlehauer, B., Sharma, V., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices. *2022 IEEE Global Engineering Education Conference (EDUCON)*, 1634–1642. <https://doi.org/10.1109/EDUCON52537.2022.9766793>
- Farah, J. C., Spaenlehauer, B., Bergram, K., Holzer, A., & Gillet, D. (2022). Challenges and Opportunities in Integrating Interactive Chatbots into Code Review Exercises: A Pilot Case Study. *EDULEARN22 Proceedings*, 3816–3825. <https://doi.org/10.21125/edulearn.2022.0932>
- Farah, J. C., Spaenlehauer, B., Ingram, S., Purohit, A. K., Holzer, A., & Gillet, D. (2023). Harnessing Rule-Based Chatbots to Support Teaching Python Programming Best Practices [In Submission]
- Farah, J. C., Ingram, S., Spaenlehauer, B., Lasne, F. K.-L., & Gillet, D. (2023). Prompting Large Language Models to Power Educational Chatbots [In Submission]

Sixth, we proposed a model to guide the participatory design of educational chatbots (C6). To validate this model, we first conducted two pilot studies comprising a workshop with eight researchers and developers in education, as well as a case study with an undergraduate student completing a semester project on designing educational chatbots. We then conducted an illustrative study whereby 25 students—enrolled in a course on software design—took part in a participatory design session to ideate an educational chatbot. Our model is presented in Chapter 11 and is featured in the following publication:

- Farah, J. C., Spaenlehauer, B., Ingram, S., Lasne, F. K.-L., Rodríguez-Triana, M. J., Holzer, A., & Gillet, D. (2023). TRACE: A Conceptual Model to Guide the Design of Educational Chatbots [In Submission]

Finally, we proposed a technical blueprint for a system that facilitates the integration of chatbots into digital learning applications (C7). We validate this blueprint through a proof of concept implementation of our architecture to showcase how it can support a learning activity. This blueprint is presented in Chapter 12 and is featured in the following publication:

- Farah, J. C., Spaenlehauer, B., Ingram, S., & Gillet, D. (2022). A Blueprint for Integrating Task-Oriented Conversational Agents in Education. *4th Conference on Conversational User Interfaces (CUI 2022)*. <https://doi.org/10.1145/3543829.3544525>

To summarize our work, Figure 1.9 provides an overview of the research context, problem statement, research questions, objectives, and contributions that constitute the core of this thesis.

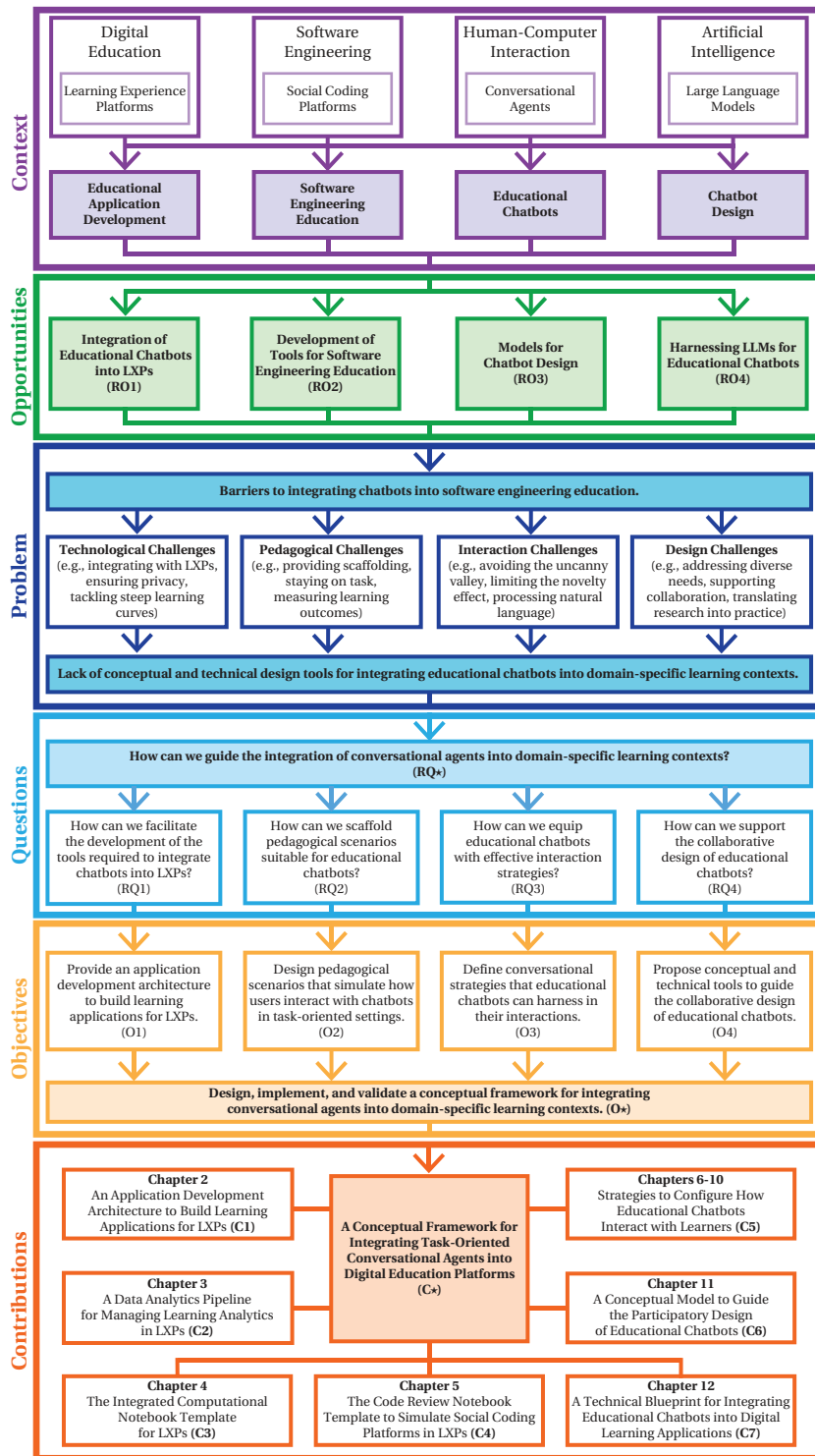


Figure 1.9: An Overview of Our Research Context, Research Opportunities, Problem Statement, Research Questions, Objectives, and Contributions

## 1.6 Structure

The remainder of this thesis is structured as follows. Each of the four subsequent parts focuses on one main cycle of the DBR process, as presented in Section 1.4.

Part I covers the first series of DBR cycles, during which we developed the *technological foundations* underpinning our work. We start in Chapter 2 by presenting the design and evaluation of the ADA used to create the learning applications employed in this thesis. Chapter 3 then outlines the LA pipeline that provided access to the data generated during our empirical studies.

In Part II, we address the *pedagogical scenarios* in which the educational chatbots will be embedded through a second series of DBR cycles. Chapter 4 presents the design and evaluation of the Code application, as well as a template to provide browser-based computational notebook solutions within digital education platforms. This evaluation informed the development of the Code Review application and the code review notebook template, which we introduce in Chapter 5.

Part III investigates *interaction strategies* for educational chatbots through a third series of DBR cycles. In Chapter 6, we report on an online experiment exploring how chatbots can harness humor to improve user perception of certain anthropomorphic qualities. Findings from this study motivated an observational study of how users interact with chatbots on social coding platforms, which we present in Chapter 7. Chapters 8–10 then present our core empirical studies, in which we explore how educational chatbots can be integrated into code review notebooks through four case studies that evaluated the effects of using Wizard of Oz (Chapter 8), rule-based (Chapter 9), and language model-based (Chapter 10) mechanisms to power these chatbots.

Our final series of DBR cycles is presented in Part IV, which builds on the findings that emerged from the first three research cycles to propose conceptual and technical tools to support the co-design of educational chatbots. Chapter 11 outlines a model that can serve to guide participatory design workshops aimed at designing educational chatbots. Chapter 12 then presents a blueprint for a technical architecture that can assist in the implementation of outcomes from design sessions guided by our model.

Finally, Chapter 13 provides a conclusion for the thesis, summarizing how the individual contributions come together to constitute a conceptual framework for integrating task-oriented conversational agents into digital education platforms.



# Technological Foundations **Part I**



---

IN the first part of this thesis, we investigate how to enhance an online learning platform with the features required to efficiently host educational chatbots. Taking a learning experience platform (LXP) as a starting point, we begin by outlining the application development architecture (ADA) that we propose as a way to standardize the implementation and deployment of web applications aimed at educational contexts. In Chapter 2, we present this ADA and an evaluation consisting of semi-structured interviews conducted with 12 developers in education. We then introduce an end-to-end learning analytics (LA) pipeline, which was implemented in an LXP to support the collection, visualization, export, anonymization, and storage of the LA data that were generated during our empirical evaluations. In Chapter 3, we present the design and implementation of this pipeline. In this brief introduction, we highlight related work.

## Related Work

Educational design approaches, frameworks, and software proposed in the literature often focus on the production of learning content (Puggioni et al., 2020) or the design and development of tools to support learning activities (Hohenwarter, 2002). Furthermore, frameworks supporting software development for education generally target specific platforms or subject matter. Although some learning management systems (LMSs) (e.g., *Moodle*) support the development and integration of custom plugins (Moore et al., 2010), there are no standard ADAs to develop web applications for LXPs. Plugins directed at specific LMSs are tightly coupled with the content hosted on the respective platform and cannot be deployed externally without custom software, such as the Learning Tool Interoperability (LTI) adapter developed by the IMS Global Learning Consortium (2019). As is the case with the LTI adapter, there have been attempts to tackle the challenge of interoperability by proposing standards that allow tools to work across learning environments. The OpenSocial standard (Häsel, 2011), for example, defined a social API specification that was adopted by some open education platforms (Gillet, de Jong, et al., 2013), but has been deprecated since 2018. Similarly, Alario-Hoyos et al. (2013)

---

proposed *GLUE!*, an architecture for the integration of external tools in Virtual Learning Environments (VLEs). However, their proposal focuses more on the architecture needed to support interoperability and less on supporting developers in building applications that adhere to those interoperability standards.

A few toolkits to develop interoperable solutions that are compatible with LXPs exist, but these are also limited by different factors. On the one hand, some online simulation providers, such as the *PhET Interactive Simulations* project (Perkins et al., 2006), provide access to their development frameworks, but these are specifically tailored for certain subjects (e.g., STEM, in the case of the PhET Interactive Simulations project). On the other hand, the *H5P* project supports the creation of reusable interactive HTML content that is both platform- and subject-agnostic. Nonetheless, while H5P is widely used in education (Amali et al., 2019), it targets publishing platforms in general and is designed primarily for content creators, not developers. Furthermore, the creation of open educational resources (OERs) with H5P can be limited by issues regarding compatibility, reusability, and “costs associated with [H5P’s] integration in a learning management system and hosting services” (Scott et al., 2021). These issues underscore the need for an accessible, platform-agnostic ADA that can be used by developers in education to create applications that support blended learning scenarios across academic disciplines. Implementations of this ADA could be designed to create applications that natively support educational chatbots, thus providing the technological base for the integration of these chatbots into LXPs. Furthermore, understanding how developers in education use ADAs, what they value and what they find challenging, could help improve the design of technological tools to support their practice.

Facilitating the development of applications aimed at LXPs also concerns the need to address how data produced by these applications—and the platforms they are hosted on—are managed. As educational institutions continue to adopt digital education solutions, the volume of educational data recorded on digital platforms continues to rise (Pardo et al., 2014). The availability of these data, along with improvements in computational capacity, has led to an increase in the resources available for LA and educational data mining. This steep rise in the generation of LA data has motivated the development of ethical frameworks that address how learner data should be collected, stored, and accessed (Slade et al., 2013).

The importance of addressing ethical and privacy issues in the context of digital education is illustrated by projects such as *LEA’s BOX* (Steiner et al., 2016) and the *Learning Analytics Community Exchange (LACE)* (Ferguson et al., 2016). On the one hand, Steiner et al. (2016) proposed a set of eight principles to devise a data protection framework for LA, which informed the design of the tools used in LEA’s BOX. Nevertheless, Steiner et al. acknowledged that “it is very difficult (if not impossible), to translate a general ethical mind-set into direct recommendations for technical designs and architectures” and concluded by highlighting broad aspects without providing specific guidelines for implementation. On the other hand, as reported by Ferguson et al. (2016), LACE spearheaded the organization of workshops to identify challenges, define goals, and share work conducted in the field of ethics and privacy



---

for LA, including how to manage the growing amount of data collected for LA.

Proper data management is one of the key dimensions of LA associated with corresponding privacy and ethics risks, as noted by Greller et al. (2012). These risks also create barriers to the adoption of digital technologies in education. To promote trust in LA solutions, Drachsler et al. (2016) proposed *DELICATE*, a list of requirements for LA implementations, highlighting educational data management as one of the aspects to be taken into account. Moreover, within a general framework, the European General Data Protection Regulation (EU GDPR 2016/679) (European Union, 2016) defined a number of requirements regarding the collection and processing of personal data.

However, standards for data models and data collection in digital education are still lacking (del Blanco et al., 2013; Hoel et al., 2014) and pose a barrier to the scaling and interoperability of LA. Indeed, the development of privacy-preserving standards and technical blueprints for LA has been somewhat eschewed. Instead, researchers have focused on other approaches to tackling these barriers. This focus is captured by Drachsler et al. (2016), who noted—in their work presenting *DELICATE*—that they would “refrain from solving a weakness in a new learning technology by proposing technical fixes or technological solutions, such as standardisation approaches”, and instead propose addressing “human factors, such as angst, scepticism, misunderstandings, and critical concerns”.

Data privacy issues are also a major challenge to promoting open data in education (Dietze et al., 2016). Although anonymization is a common strategy to protect data privacy, there are some obstacles that hinder its adoption in practice (Gursoy et al., 2017). First, there is an inherent trade-off between achieving privacy and preserving the utility of an anonymized data set, since the granularity of anonymization can hinder the envisioned data analysis. For example, a well-documented limitation of the *k-anonymization* algorithm (W. Jiang et al., 2005) is that—when processing a dataset to ensure that individual data points are no longer distinguishable—it removes useful information from the data set (Komarova et al., 2017). Anonymization algorithms based on encryption also have well-documented limitations, such as the vulnerability to attacks in which bad actors potentially reverse engineer a hash function using brute-force algorithms against hashed data attributes that have common unhashed values (Demir et al., 2018). Second, anonymization approaches depend on the data being anonymized and are not easily generalizable. For example, techniques to anonymize geolocation data will differ from those for anonymizing usernames, and two platforms may generate different formats for the same data attribute. Third, automatically identifying sensitive attributes in a data set is a complex problem, meaning that researchers need to be familiar with a dataset and its schema beforehand (Chicaiza et al., 2020).

Several techniques have been proposed to address these challenges. Privacy-preserving LA tools often include simple methods such as suppressing or hashing identifiers (Gursoy et al., 2017; Heurix et al., 2015; Steiner et al., 2016). Other robust techniques include the aforementioned *k-anonymity* (W. Jiang et al., 2005), which ensures that the quasi-identifiers

---

map to at least  $k$  users, and *l-diversity* (Machanavajjhala et al., 2006), which guarantees that the sensitive values are diversified enough for each set of users with the same quasi-identifiers. Finally, *t-closeness* (N. Li et al., 2007) adds the property that the distribution of sensitive values must be similar for each of these sets. Although previously proposed LA data management solutions include different anonymization strategies in their design (Chicaiza et al., 2020; Majumdar et al., 2019), there is no approach that generalizes to the diversity of formats and data types present on educational platforms.

Finally, the lack of incentives and infrastructure for data sharing poses a challenge to the adoption of open data practices in digital education. One of the issues limiting this adoption is the fact that existing data sharing platforms for digital education are not integrated into a full data life cycle (Nicholson et al., 2017), making dataset sharing cumbersome. Incorporating sharing functionality alongside data visualization and anonymization tools—without the need to switch between different platforms and tools—could help promote open data practices and foster collaboration in the spirit of open science.

## 2 Application Development Architecture

WHEN educational activities are conducted on digital education platforms, educators often harness interactive web applications (apps) to support these activities. These apps are often created by web developers or by researchers, educators, and even students with programming experience. As a first step toward integrating educational chatbots into LXPs, we designed an application development architecture (ADA) aimed at supporting developers in creating apps for education. Our goal was to provide a systematic and efficient way to develop apps compatible with LXPs. While these apps could serve different purposes and cover several educational domains, following a standardized architecture could ease the integration of chatbots into these apps. Finally, we wanted to shed light on how developers used an implementation of our ADA, the apps they created, and the challenges they faced. In this chapter, we present our ADA and report the results of a study comprising interviews with 12 developers who used an implementation of our architecture for a production-ready LXP. Our findings highlight that while the creation of web apps for education can be facilitated by a purely software-based application development kit (ADK) following our architecture, effectively exploiting such ADK requires domain knowledge that was not captured in the ADA but could be acquired through in-depth documentation, tutorials, and collaboration between developers and educators.

By laying out the technical groundwork for developing apps that will be able to host educational chatbots, this chapter partially addresses **RQ1**:

How can we facilitate the development of the tools required to integrate chatbots into LXPs?

The content of this chapter was partially presented in the following publication:

1. Farah, J. C., Ingram, S., & Gillet, D. (2022). Supporting Developers in Creating Web Apps for Education via an App Development Framework. *HEAd'22 Conference Proceedings*, 893–890. <https://doi.org/10.4995/HEAD22.2022.15652>

### 2.1 Introduction

In this chapter, we present the design of an open-source application development architecture (ADA) aimed at supporting the creation of apps for education. The design of our ADA proposes a series of components in conjunction with a software engineering process that can facilitate the development of apps specifically aimed at deployment within a digital learning environment. To provide a proof of concept, we implemented an application development kit (ADK) following this architecture for a production-ready LXP. Between 2019 and 2021, this implementation was used by 20 developers to create over 35 apps. To evaluate how our ADA could assist in the creation of apps for education, we conducted a qualitative study comprising semi-structured interviews with 12 developers who used this ADK between 2019 and 2021. The transcripts of these interviews were analyzed using thematic analysis, shedding light on the (i) *technical*, (ii) *educational*, and (iii) *licensing* aspects of our architecture, as well as on how to make app development for education more accessible. Our results point toward the need for collaboration between developers, researchers in education, and educators, which could be facilitated by extensions of our architecture.

This chapter addresses two needs that were presented in the introduction to this part of this thesis. First, the need for an accessible, platform- and subject-agnostic, ADA to help developers create apps for education. Our ADA exclusively targets educational contexts and is designed for developers working on creating web apps for such contexts. Second—to the best of our knowledge—no study has focused on gathering feedback from developers on their experience in creating apps for education. By tackling these needs, we propose and validate one of our contributions, namely an ADA to support the implementation of apps that can integrate with LXPs (C1).

This chapter is structured as follows. We start in Section 2.2 by presenting the design of the components and the process that make up our proposed ADA. In Section 2.3, we then outline how we implemented this design to create an ADK to support the development of apps for a production-ready LXP. Our evaluation with developers that used the ADK is presented in Section 2.4. Key results from this evaluation are highlighted in Section 2.5 and discussed in Section 2.6. Finally, we conclude in Section 2.7.

### 2.2 Design

Our ADA is designed following a headless and decoupled structure similar to that of the popular Gatsby framework (Gatsby, 2022). Gatsby is a JavaScript framework for creating and deploying websites. It allows developers to use a command-line interface (CLI) to create websites based on templates, some of which are featured in a public repository. Developers can then build on those templates and deploy their websites to their own infrastructure or to one provided by Gatsby. Our design adopts and adapts this strategy to digital education platforms, focusing on two key dimensions. First, on the key components that are required to

implement an ADK following our architecture. Second, on the software engineering process that these components support. In this section, we present these two dimensions in detail.

### 2.2.1 Components

We start by defining the six core components of our architecture: (i) *context*, (ii) *application programming interface (API)*, (iii) *templates*, (iv) *command-line interface (CLI)*, (v) *local development*, and (vi) *repository*.

#### Context

Our ADA is primarily designed to create apps that are to be incorporated into a digital learning context, which we refer to as running in *contextual mode*. Specifically, apps should be loaded by calling the URL where the app is hosted, either in an `iframe` (if embedded directly in a learning activity) or as a standalone web page (if opened as a separate link). To inform an app of its context, the URL should include a query string that contains its unique identifier (ID) within the learning platform, as well as the IDs of the activity that it is a part of (i.e., its *context*) and of the user that is interacting with it. The query string also informs the app whether it is running within the platform's educator or learner interface and provides the endpoint that the app needs to call to access the platform's application programming interface. Apps can then render role-specific views or components, such as a settings panel that is only available to educators. If this context is missing, apps can fall back to running in *standalone mode* (i.e., without support from a digital learning platform). The standalone mode is particularly useful for apps that do not have to persist user-generated data across sessions (e.g., simulations) or to perform demonstrations with sample data.

#### Application Programming Interface

Apps running in contextual mode are aimed at communicating with digital learning platforms and therefore need access to data hosted by these platforms, such as learning resources and activity traces. Access to these data is often made possible via application programming interfaces (APIs). While different platforms can define their API differently, our goal was to identify the core data required to support educational apps and design our API specifications accordingly. To do this, we propose a permissioned API (i.e., one that only allows access to users with the appropriate permissions). This API should expose endpoints for the app to perform the following actions:

- fetch information about the user that is currently logged in,
- fetch a list of learners that have interacted with the app or the lesson in which the app is embedded,

- fetch and update its settings,
- create, fetch, update, and delete its own resources, which include user-generated learning outputs, and
- create, fetch, and delete LA generated by users while interacting with the app.

These endpoints are loosely based on the OpenSocial standard (Häsel, 2011), which—as mentioned in the introduction to Part I—defined a social API specification that was adopted by some open education platforms (Gillet, de Jong, et al., 2013). Once the API is set in place, apps can exploit it to get access to the content they require to run.

### Templates

Although it should be possible to develop an app from scratch, templates allow developers to maximize code reusability and focus on app- and use-case-specific code. Templates are code repositories that serve as a starting point for the development of an app. An ADK based on our architecture should provide default templates and developers should be able to create custom templates, either from scratch or by building on previously created templates. This reduces the amount of boilerplate that developers need to write in order to create apps. That is, templates allow for the basic structure of a particular type of app to be only written once. The structures provided by templates can facilitate many development tasks, including enforcing code style, setting up development and testing frameworks, providing design components, and abstracting the API through ready-made functions that pre-establish the link between the app and the platform(s) it will be hosted on. Furthermore, templates can be generated in various frontend development languages (e.g., JavaScript, TypeScript, Elm) and frameworks (e.g., React, Angular, Vue) to attract a wider community of developers.

### Command-Line Interface

To make it straightforward to get started with a template, an ADK following our architecture should provide a command-line interface (CLI) that can guide developers through the process of bootstrapping an app. In particular, the CLI should provide a command to prompt developers for information concerning the template they want to use, the name of their app, and optional identifiers (IDs) required for access to any permissioned APIs and deployment to external infrastructure. The CLI should also be used to test an app locally, package it as a compressed file, deploy it, and publish it to open repositories.

### Local Development Environment

Local development should be supported by tools that provide a mock API resembling how apps will interact with the LXPs in which they will be deployed. These tools should also

allow developers to quickly switch between contexts in order to develop, visualize, and test how different stakeholders will interact with their apps. Finally, if development requires a full running version of the LXP, integration with an online or local sandboxed version of the platform should be made accessible.

### Open Educational App Repository

A final component of our design is the app repository. While the code for the apps could be hosted in any source code repository (e.g., GitHub, GitLab), these repositories are not specifically aimed at education. To allow educators to more easily discover and use apps built with an ADK following our architecture, the ADK should integrate with a repository of open educational resources that could host these apps and provide information about them in a way that is accessible to nontechnical users. Tagging published versions of apps following standards such as Semantic Versioning (SemVer) could allow LXPs to perform automatic app updates without disrupting existing deployments of the app in learning activities.

### 2.2.2 Process

The components that make up our architecture come together to provide developers with a process to manage the full life cycle of an app aimed at educational contexts. In this section, we outline the key steps in this process, which is primarily supported by the CLI component. The complete process, which features the components presented in Section 2.2.1, is illustrated in Figure 2.1.

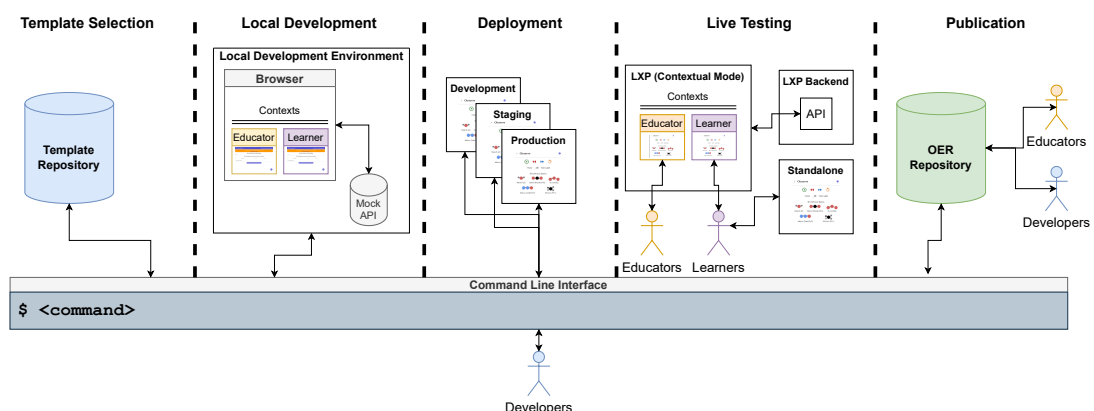


Figure 2.1: Our application development architecture defines key components and a software engineering process to facilitate the development of apps specifically aimed at deployment within a digital learning environment.

### Template Selection

The development process starts with the selection of a template that will bootstrap the development of the app. Once the appropriate template has been identified, the developer can harness the CLI to set up a local development environment as defined by the selected template.

### Local Development

Development of the app should take place in the developer's local environment. Setting up this local development environment should be facilitated by the CLI, which interfaces with the corresponding lower-level tools that are required for development (e.g., Docker, Git, Node.js). Local development also allows developers to preview the different contexts they would like the app to support, namely those contexts aimed at *educators* and those aimed at *learners*. This preview functionality facilitates development without the need to deploy to an external environment.

### Deployment

Once the app is ready to be tested internally by other stakeholders, the CLI allows the developer to deploy a version of the app to different external environments. These environments can be provided by one or more LXPs that the app is targeting, by the developers themselves, or by an external third party. Crucially, this deployment step allows developers to test their app in multiple environments (e.g., *development* for initial validation, *staging* for quality assurance) and validate the app before it is sent to a production-ready environment.

### Live Testing

When the app is deployed to production, it is ready for educators to integrate it into their digital activities for use with learners. This constitutes the *live testing* phase of this process, which validates that the app can serve educators in their practice. Incorporating error monitoring tools can serve to detect issues with the app once it is live, as they can provide ways for end users to send feedback and crash reports back to the developer.

### Publishing

An app can be published once it is ready to share with the community. Published apps can be featured in repositories of OERs so that educators can test them before incorporating them into their lessons. These repositories can also be directly linked to LXPs in order to facilitate the creation of learning activities with the apps featured in the repository. Note that this process is iterative. That is, once an app is first published, changes and updates can be incorporated through the same process before a new version is published again in the repository.





Figure 2.2: The Graasp ecosystem consists of four interfaces, each addressing a different aspect of the digital learning experience. (Adapted from Gillet, Vonèche-Cardia, et al. (2022).)

## 2.3 Implementation

One of the central aims of this thesis is to advance our ability to integrate educational chatbots into apps designed for LXPs. In order to maximize ecological validity, we chose to develop the technological ecosystem needed for this integration using a production-ready LXP as a base. Hence, to frame this development and support our work, we chose *Graasp*<sup>1</sup> (Gillet, Vonèche-Cardia, et al., 2022), a free, open-source LXP jointly developed by the École Polytechnique Fédérale de Lausanne and the Graasp Association.

### 2.3.1 Graasp

Graasp is a learning experience platform aimed at supporting educators with the creation and dissemination of learning resources. Designed following a quadriptych model, Graasp provides four main user interfaces, each focused on a particular aspect of the digital learning experience. This model is summarized in Figure 2.2.

First, there is the *Graasp Builder* interface (see Figure 2.3), which allows educators to prepare learning activities using multimedia content. Graasp Builder provides educators with the ability to integrate and configure the resources they will use to create their digital learning activities. Learning activities can be scaffolded into step-by-step exercises that can be contextualized with text, images, links, and other interactive content. Once a learning activity

<sup>1</sup>Work conducted in this thesis spanned two different versions of Graasp. While the core functionalities remained the same, certain parts of the user interface changed radically. In most cases, we have chosen to include screenshots of the latest version. Nevertheless, where applicable, screenshots of the previous version have been included.

## Chapter 2. Application Development Architecture

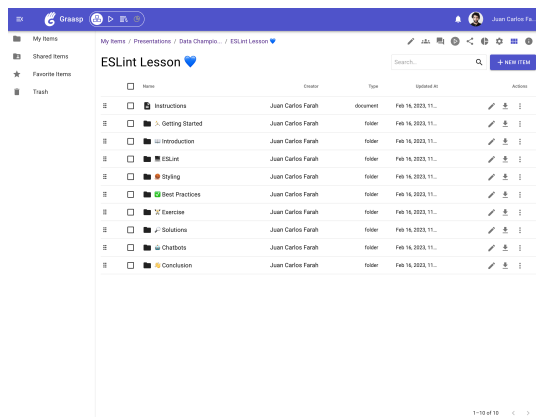


Figure 2.3: Graasp Builder is aimed at educators looking to create learning activities with resources from across the web.

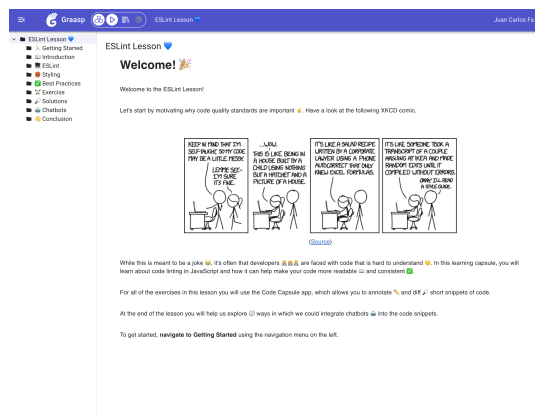


Figure 2.4: Once an activity is ready, a link to access it via Graasp Player can be shared with learners. (Comic strip by Munroe (2015).)

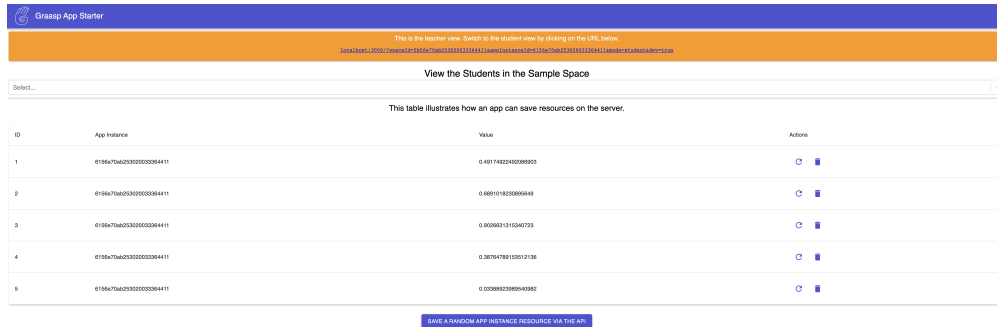
has been prepared, it can be shared with learners through the *Graasp Player* interface (see Figure 2.4). This second interface is an environment directed at learning and is accessible through a link. Using this link, learners can take part in the learning activity, navigating through pages containing the exercises prepared by the educator. A third interface—*Graasp Analytics*—focuses on visualizing and providing access to data capturing how both educators and learners interact with the learning activity. This interface allows educators and researchers using Graasp to access a high-level overview of how a learning activity is created and exploited. It contains charts of actions by day, time of day, and type, as well as an interactive map of actions coarsely clustered by geolocation. Finally, there is the *Graasp Library* interface, which allows educators to share their content as OERs with the community as a whole. Using the Graasp Library, educators can easily discover, adapt, and reuse learning resources and activities that have been created with the Graasp Builder.

Given that Graasp has been active since 2006 and—as of 2022—had over 400,000 users, it serves as an appropriate educational platform on which to scaffold our technological environment. Hence, we implemented an ADK following the proposed ADA to support the creation of apps for Graasp. Henceforth, we refer to this implementation as the *Graasp Application Development Kit (Graasp ADK)*.

### 2.3.2 Templates

The default templates provided by the Graasp ADK were written in TypeScript and JavaScript and cover multiple frontend development frameworks. These templates encapsulate the core structure of an app targeted at Graasp, including—on top of the aforementioned development tasks—privacy and role-aware data access and visualization modes, state management, an open-source license, and compatibility with an offline desktop environment. These templates

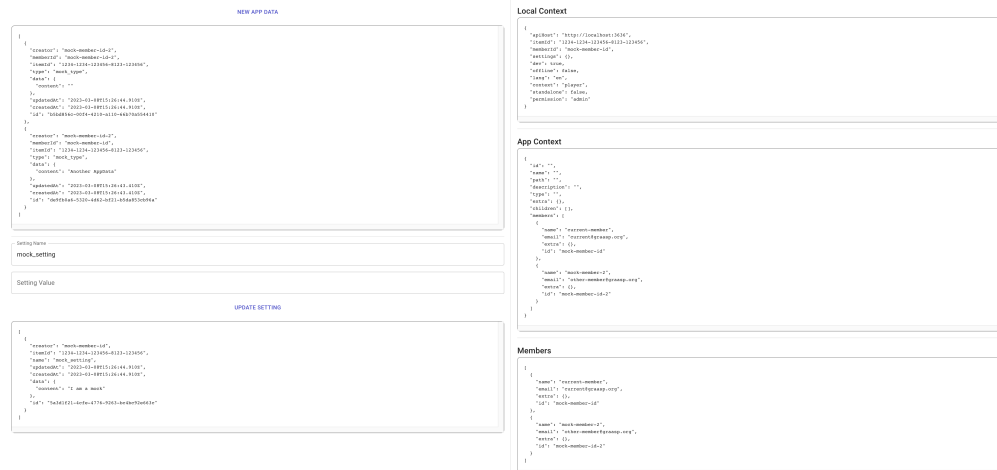
also have a minimal user interface, allowing developers to customize the visual appearance of their apps as they deem best. Nevertheless, the Graasp ADK provides basic integration with the user interface libraries used by Graasp, promoting consistency between the apps and the platform they are hosted on. Examples of these templates are shown in Figures 2.5–2.6.



The screenshot shows the 'Graasp App Starter' interface. At the top, there's a blue header with the Graasp logo and a message: 'This is the teacher view. Switch to the student view by clicking on the link below: [switch-to-student-view](#)'. Below this is a section titled 'View the Students in the Sample Space' with a dropdown menu set to 'Select...'. A note states: 'This table illustrates how an app can save resources on the server.' The table has five columns: ID, App Instance, Value, and Actions. It contains five rows of data, each with a unique ID, an app instance ID, a numerical value, and a set of actions (a copy icon and a delete icon). At the bottom, there is a button labeled 'SAVE A RANDOM APP INSTANCE RESOURCE VIA THE API'.

ID	App Instance	Value	Actions
1	0156e76a025302000394411	0.40174922402089803	
2	0156e76a025302000394411	0.689101820896549	
3	0156e76a025302000394411	0.803083131024073	
4	0156e76a025302000394411	0.3879478102012136	
5	0156e76a025302000394411	0.030882208540382	

Figure 2.5: A simple template can serve to bootstrap an app on Graasp.



The screenshot shows a more advanced template in the 'Graasp App Starter' interface. It features a 'NEW APP DATA' section with a code editor containing JSON data. Below the code editor is a 'Settings Name' field with the value 'mock\_setting' and a 'Setting Value' field. To the right, there are two panels: 'Local Context' and 'App Context', both displaying JSON data. At the bottom right, there is a 'Members' panel showing a list of members. The interface is designed to provide developers with insight into how data is generated and stored on Graasp.

Figure 2.6: By default, a more advanced template provides developers with insight into how data is generated and stored on Graasp.

### 2.3.3 Command-Line Interface

We implemented Graasp ADK's CLI as a Node.js tool that could compile to any major operating system requiring only minimal external dependencies. Following the recommendations of our design, the new command allows developers to access and customize app templates through a simple prompt. Once the required information is provided, the CLI fetches the selected template and installs all the packages needed for the app to run locally and be compatible with Graasp's deployment process, which uses *GitHub Actions* to deploy to a cloud infrastructure

hosted on *Amazon Web Services*. These commands harness lower-level utilities (e.g., *git*, *npm*, *aws*) and custom shell scripts, abstracting this logic from the developer. An example of the bootstrapping process using the CLI prompt to select the default template is shown in Figure 2.7.

```
(base) → ~ npx @graasp/cli@latest new
? Name An Awesome Educational App
? Type App
? Framework React
? Programming Language TypeScript
? Deploy using GitHub Actions (recommended) Yes
? Graasp App ID 281abdaa-fe3c-42ad-a4ff-8362bcefb587
creating new site from git: https://github.com/graasp/graasp-app-starter-ts.git
Cloning into 'graasp-app-an-awesome-educational-app'...
remote: Enumerating objects: 204, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 204 (delta 25), reused 20 (delta 20), pack-reused 152
Receiving objects: 100% (204/204), 1.92 MiB | 8.00 MiB/s, done.
Resolving deltas: 100% (60/60), done.
created starter directory layout
writing environment files...
wrote environment files
removing ignored files
branding project...
```

Figure 2.7: Creating an app is facilitated by the CLI's new command, which fetches a template and bootstraps a local development environment.

### 2.3.4 Local Development

To facilitate the setup of a local development environment, a series of tools were included with Graasp ADK's default templates. These tools help developers follow best practices from the JavaScript and TypeScript communities, including the use of libraries such as *Prettier* and *ESLint* (Zakas, 2013) for code style, as well as *Cypress* for running end-to-end browser tests. The mock API was based on *Mirage JS*, which runs directly alongside the frontend code, requiring no running backend. Nevertheless, for developers requiring a more complete local environment, we adapted and packaged Graasp's various frontend and backend services using *Docker* containers. These containers simplify deployment and provide a consistent experience across environments. Finally, mock data were included by default in the templates in order to provide validated examples of the structure developers should expect to encounter in production environments.

### 2.3.5 Repository

The implementation of the app repository was aligned with the development of the Graasp Library interface. As such, the Graasp Library was enhanced to support the inclusion of individual apps as well as collections of apps. Collections allowed external developers to

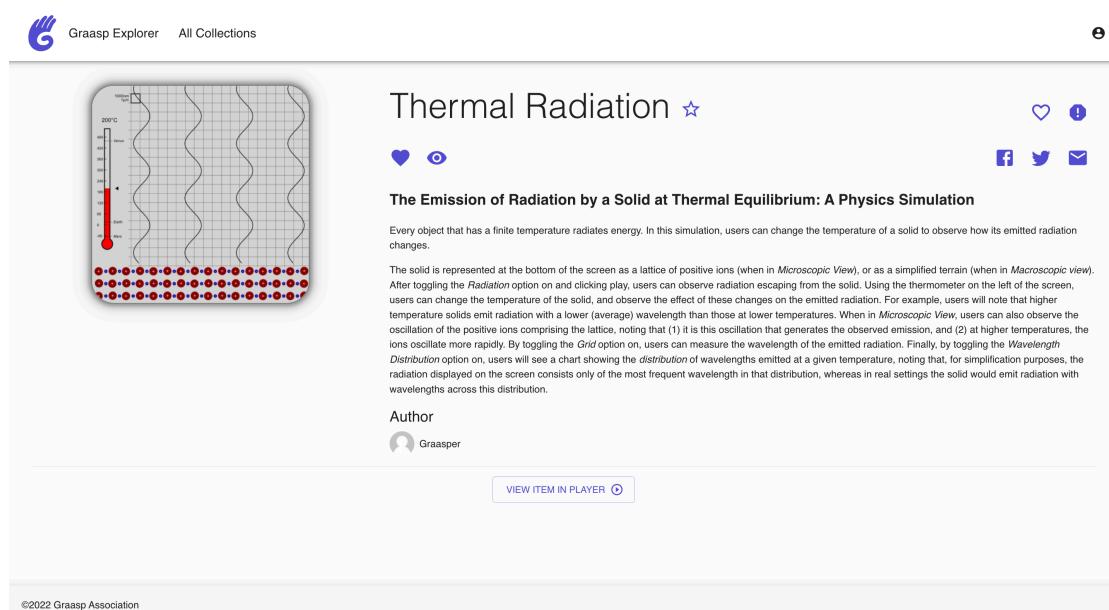


Figure 2.8: An app created with the Graasp Application Development Kit—which is an implementation of our application development architecture—can be featured on the Graasp Library, a repository of open educational resources.

showcase their apps and facilitate their discovery. This followed the approach used by other app repositories such as Go-Lab (de Jong et al., 2014), whereby educators can quickly search for, filter, and test apps that are relevant to their practice. Figure 2.8 depicts how an app can be featured in the repository.

Over 35 apps were created with the Graasp ADK between 2019 and 2021. These apps range from virtual labs targeting physics education, to learning analytics visualizations for both educators and learners, to interactive chatbot interfaces. Figure 2.9 illustrates four examples of apps that were created using the Graasp ADK:

1. *Sticky Notes* (top left) allows learners to brainstorm ideas using multicolor sticky notes.
2. *Task Management* (top right) provides task management features for collaborative learning activities (La Scala et al., 2022).
3. *Greenhouse Effect* (bottom right) helps learners visualize the effects of global warming.
4. *Light Pollution Simulator* (bottom left) aims to raise awareness of the effects of light pollution on the night sky (Gomes et al., 2019).

More importantly, the Graasp ADK served to create the apps used in the work carried out for this thesis. These apps notably include the three apps around which our pedagogical contexts were scaffolded:

## Chapter 2. Application Development Architecture

1. *Code*, which allows learners to write and execute Python and JavaScript code in the browser.
2. *Code Review*, which allows learners to annotate code and supports interactions with chatbots following rule-based scripts.
3. *Code Capsule*, which incorporates the functionalities of *Code* and *Code Review* into one app and supports interactions with chatbots powered by large language models.

Other supporting apps, such as *Text Input*, *Linear Scale*, *File Drop*, and *Submit Answer*, were also developed with the Graasp ADK and incorporated into the pedagogical contexts that were used in our empirical evaluations.

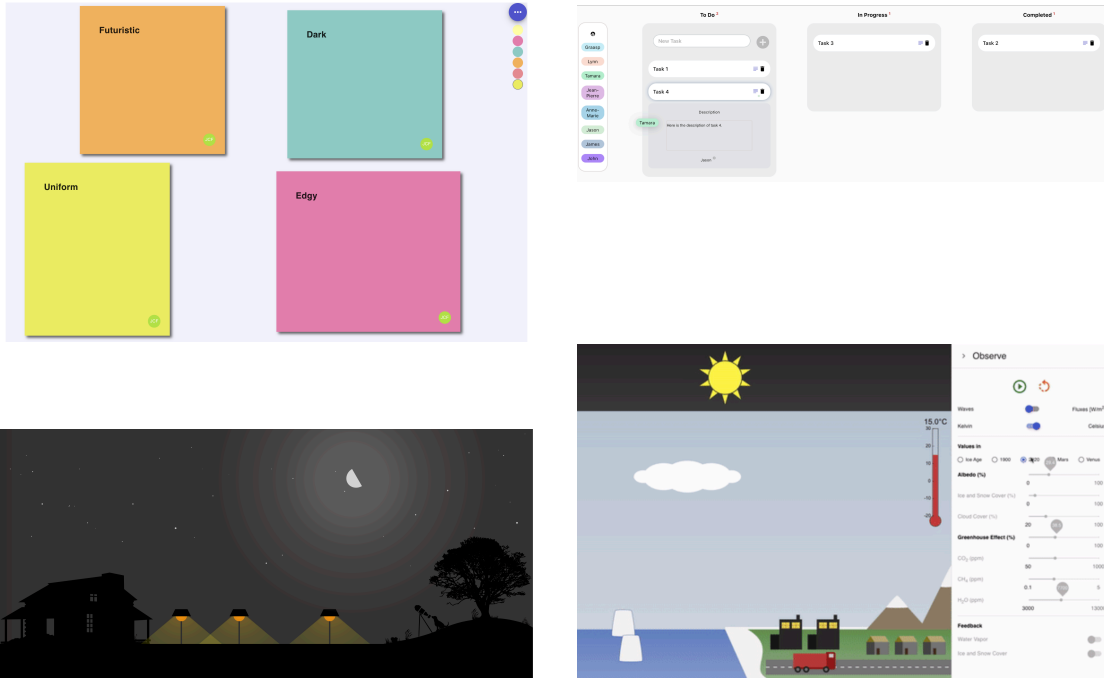


Figure 2.9: Four example apps (clockwise from top left): *Sticky Notes* for design thinking, *Task Management* for collaboration (La Scala et al., 2022), *Greenhouse Effect* for climate change awareness, and *Light Pollution Simulator* to visualize the effects of light pollution (Gomes et al., 2019).

## 2.4 Methodology

To validate our ADA, we conducted a qualitative evaluation focused on how developers had used the implementation of our architecture for Graasp. Our evaluation addressed the following research question, which is an extension of **RQ1**:

- How can our ADA facilitate the creation of apps for education? (**RQ1a**)

Specifically, we focus on three aspects of this research question: (i) *technical*, (ii) *educational*, and (iii) *licensing* aspects.

### **2.4.1 Participants**

Following the total population purposive sampling method, we contacted all developers who were identified as having worked with the Graasp ADK as of January 2022. A total of 13 developers agreed to participate and were interviewed in January and February 2022. One interview was excluded from our analysis due to technical issues with the transcript. The 12 developers (2 female, 10 male) included in our final analysis include two researchers, two software engineers, four bachelor's, two master's, and two doctoral students. Participants were based in five different countries.

### **2.4.2 Instruments**

We followed a qualitative approach to addressing our research question. This approach harnessed semi-structured interviews, given their applicability to small-scale research (Drever, 1997). The interviews lasted approximately 30 minutes, took place using an online video conference tool, and were loosely guided by 15 questions covering four topics: (i) developer background, (ii) experience with the ADK, (iii) educational context, and (iv) importance of open-source code. The interviews were recorded with the consent of the participant.

### **2.4.3 Data Analysis**

We analyzed our qualitative data using thematic analysis, performing line-by-line coding (Char-maz, 2006) on the interview transcripts to identify emergent themes with respect to the three aspects of our study.

## **2.5 Results**

In this section, we highlight the key findings of our qualitative analysis, focusing on the three aspects of our research question.

### **2.5.1 Technical Aspect**

Concerning the technical aspect, all 12 developers were overall positive about the support received by the ADK, although five developers expressed that they found it challenging to get started due to their lack of experience either with JavaScript (two developers) or React (three developers). Five developers noted that the boilerplate provided by the default template was helpful, while nine developers alluded to the consistent and adequate structure of the code. Eight developers highlighted the need for more technical support, with five developers

suggesting the creation of video tutorials. These tutorials were suggested as a way to explain what was included in the template (two developers) and to help developers create a sample app (two developers). Four developers also noted the need for more templates to support other technology stacks (one developer), keep the code up-to-date with current standards (three developers), and make parts of the code optional (one developer).

### 2.5.2 Educational Aspect

Concerning the educational aspect, all developers were overall positive about the ADK's impact on helping create apps for education, with six developers noting that the ADK's support for LA was an important feature. Nevertheless, five developers reported that understanding the educational context and nomenclature of the ADK was challenging. To address this challenge, four developers highlighted the importance of collaboration between developers and educators. Two developers specifically suggested incorporating requirements elicitation and validation processes into the ADK. The importance of collaboration is best illustrated by a quote from one of the developers.

“One of the challenges is that very often it's not really about the code and how skilled the coders are... it's more [about] the teachers and the teachers who know the [subject matter] and who know how the students will receive [the app] and what they will find challenging and what they will find interesting... I'm not a teacher, so I wouldn't have that knowledge. I don't know the [subject matter]... maybe I could learn it, but I'm not going to learn it in a way that someone who's been teaching it for 5–10 years knows it, and also [knows the] students.”

### 2.5.3 Licensing Aspect

Finally, concerning the licensing aspect, all 12 developers were positive about the fact that the ADK encouraged open-source. Reasons put forth for the importance of open-source included (i) not reinventing the wheel (two developers), (ii) making knowledge and examples accessible (seven developers), and (iii) building a community (three developers).

## 2.6 Discussion and Implications

The results of our evaluation show that on the one hand, developers found the Graasp ADK useful in terms of the technical advantages it provided through convenient boilerplate, the ease of use of the CLI, and the consistent structure of the code. On the other hand, the Graasp ADK failed to fully support developers in understanding the educational aspects involved in the development of apps aimed at learning contexts. The need to better support and encourage collaboration between the different stakeholders in education is aligned with work by Tavares et al. (2020), who proposed a participatory design process to bring together researchers in ed-



education and end users of educational mobile apps. In the context of designing an educational chatbot, Durall Gazulla et al. (2023) also emphasized the need to support collaboration when developing learning technologies. Indeed, our findings suggest that collaboration should also be established between software developers, researchers in education, and educators. This collaboration could be facilitated by the ADA, which could include educational aspects in its documentation and tutorials, incorporate an automated validation process, provide a library of reusable components that are strongly linked to pedagogical interfaces, and follow nomenclature defined through a co-design process involving both developers and educators.

These findings have concrete implications for the work conducted in the rest of this thesis and significantly inform five of the other contributions. First, we harnessed the implementation of our ADA to create the apps that were used in our empirical studies. In turn, these apps were used to scaffold the pedagogical scenarios that served to contextualize these studies, including both the *integrated computational notebook* (C3) that we introduce in Chapter 4 and the *code review notebook* to simulate social coding platforms in educational contexts (C4), which we present in Chapter 5. Second, findings suggesting the need for collaboration between developers and educators inform our approach to designing a model for facilitating participatory design sessions (C6), which will be presented in Chapter 11. Third, the need to incorporate elements from education into our technical tools guided the design of our data analytics pipeline (C2) and our technical blueprint for integrating educational chatbots into learning applications (C7). Presented in Chapter 3 and Chapter 12, respectively, these technical architectures are more closely tied to educational concepts than the ADA presented in this chapter.

## 2.7 Conclusion

In this chapter, we presented the design and evaluation of an ADA aimed at supporting developers in creating apps for educational contexts. Our results suggest that developers would benefit from support from experts in education, which could be facilitated by the ADA. These results have important practical and theoretical implications for the rest of the work carried out in this thesis, serving to create the tools presented in the following chapters as well as providing concrete guidance for the design processes that will be discussed in Part IV.

Nevertheless, our evaluation has some limitations worth noting. First, while the sample size is appropriate for a qualitative study using semi-structured interviews, it would be useful to complement these findings with a larger sample size, along with a quantitative analysis of the actual usage of the ADA. Second, our ADA allows apps to be deployed in standalone or contextual mode. However, our proof-of-concept implementation tested contextual mode only with Graasp. To increase portability, contextual mode should be compatible with multiple platforms. In future work, we aim to address these limitations and incorporate the feedback received into the next iteration of our architecture.



## 3 Learning Analytics Pipeline

IN the previous chapter, we presented an architecture for implementing ADKs that can support the creation of apps for LXPs. When educators and learners interact with these apps, their actions are often recorded for learning analytics (LA). This data-driven approach to analyzing how learners interact with educational technologies also applies to the conversations and interactions that learners can have with educational chatbots. Nonetheless, despite the importance of LA in digital education, there is limited support for researchers to generate, access, and share experimental data while complying with ethical guidelines and privacy legislation. To address this need, we propose an architecture for a pipeline to support researchers with these data-related tasks and present a blueprint for how this architecture can be integrated with existing platforms, enabling researchers to conduct experiments within learning environments, adhere to legal and ethical privacy frameworks, and share their data with a wider audience. In this chapter, we present the design of our architecture and the proof-of-concept implementation of the pipeline on Graasp.

By laying out the technical groundwork for collecting, extracting, anonymizing, and visualizing data from the applications that will host the educational chatbots, this chapter partially addresses **RQ1**:

How can we facilitate the development of the tools required to integrate chatbots into LXPs?

The content of this chapter was partially presented in the following publication:

- Farah, J. C., Soares Machado, J., Torres da Cunha, P., Ingram, S., & Gillet, D. (2021). An End-to-End Data Pipeline for Managing Learning Analytics. *2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*. <https://doi.org/10.1109/ITHET50392.2021.9759783>

### 3.1 Introduction

Digital education platforms often record learner interactions within online lessons and activities, providing educators and researchers with detailed learning analytics (LA) (Kellogg et al., 2015). LA offer significant potential to improve educational experiences and can support content personalization, the provision of real-time feedback, and the early detection of at-risk learners (Siemens et al., 2011). Although work has been done to promote data sharing in education and make education-focused open data repositories publicly available, ethical concerns and recent privacy legislation pose a challenge to the wider adoption of open data practices in education (Dietze et al., 2016). Furthermore, data interoperability and contextualization issues make it particularly difficult to support cross-platform LA (Ruipérez-Valiente et al., 2020).

In this chapter, we propose a comprehensive LA data pipeline architecture aimed at supporting open research in education and present the implementation of this architecture on Graasp. The goal is to outline the pipeline's architecture and the practical considerations encountered when incorporating the pipeline into a web-based LXP, thereby demonstrating the technical feasibility of developing and deploying this architecture and validating our second contribution, namely an end-to-end LA pipeline to support the full data management cycle for research in education (C2). Given that flexibility is often required for such implementations, we focus both on how our pipeline's components can be used as part of an integrated system and individually as standalone units. Our architecture aims to address the need for technical designs providing end-to-end solutions to manage LA, as highlighted in the introduction to Part I of this thesis. By integrating sharing functionality alongside data visualization and anonymization tools, LA management solutions following our architecture could also help promote open data practices and foster collaboration in the spirit of open science.

This chapter is structured as follows. We begin in Section 3.2 by providing an outline of the design considerations that guided our work. We then present the architecture of our data pipeline in Section 3.3. Section 3.4 presents a discussion of our implementation as well as its implications for the rest of the work conducted in this thesis. We conclude, highlighting limitations and future work, in Section 3.5.

### 3.2 Design Considerations

To guide the design of our pipeline, we build on our review of related work—as presented in the introduction to this part of this thesis—and on a previous requirements elicitation process comprising a survey conducted with 40 researchers in education (Machado et al., 2019). Outcomes from this elicitation process showed that researchers were most interested in 19 features across five different processes related to LA data collection, management, and dissemination. Table 3.1 summarizes these features.

Informed by this requirements elicitation process, our goal was to provide educators and

Table 3.1: Features Requested to Support Research on Digital Education Platforms (Adapted from Machado et al. (2019).)

Process	Features
<i>Bootstrapping Research Studies</i>	<ul style="list-style-type: none"> <li>• Allow researchers to ask teachers to participate in research studies.</li> </ul>
<i>Ensuring Consent</i>	<ul style="list-style-type: none"> <li>• Allow participants to provide consent.</li> <li>• Allow researchers and participants to view signed consent forms.</li> <li>• Allow participants to withdraw consent.</li> </ul>
<i>Gathering Data</i>	<ul style="list-style-type: none"> <li>• Configure the data-related parameters of an experiment.</li> <li>• Collect data generated inside the educational platform.</li> <li>• Provide contextual information.</li> </ul>
<i>Managing Datasets</i>	<ul style="list-style-type: none"> <li>• Dedicated access for participants to view data collected about them.</li> <li>• Automatic removal of data from participants who withdraw consent.</li> <li>• Compliance with data privacy protection requirements.</li> <li>• Store data in a custom location.</li> <li>• Verification of the authenticity of the data generated on the platform.</li> </ul>
<i>Supporting Open Research and Collaboration</i>	<ul style="list-style-type: none"> <li>• Repository to expose datasets and associated resources.</li> <li>• Different levels of exposure and granularity to share datasets.</li> <li>• Citable identifier for datasets.</li> <li>• Data export in multiple formats.</li> <li>• Data import from external repositories and new contributions.</li> <li>• Clear specification of rights and terms of use of the dataset.</li> <li>• Interaction with datasets in the repository.</li> </ul>

researchers with a holistic pipeline to work with these data from the moment of capture to the moment they are being openly shared with the pedagogical and educational research community. The pipeline design was therefore driven by three primary considerations:

1. **An integrated, end-to-end approach.** Although many digital platforms in education and beyond provide the functionality to store, visualize, and otherwise work with data, one of our contributions was to combine these functionalities into a unified and cohesive architecture for managing data over their life cycle. With the tools proposed by our architecture, users should be able to capture and visualize learning data, download complete datasets for offline analysis, and anonymize datasets to make them shareable. However—although designed as part of an integrated process—educators and researchers should also be able to use whichever subset of these tools is appropriate for their workflows and use cases.
2. **A versatile technical toolkit.** The pipeline's audience includes educators and researchers with different backgrounds and levels of technical experience. By design, tools should be accessible and useful for such a broad audience. For example, a user-friendly graphical user interface (GUI) could be complemented with functionalities enabling more technical users to directly customize certain features.
3. **Transparency and openness.** Given the sensitivity of working with private data, the architecture encourages full transparency at every step of the pipeline. For example, data should only be collected after user consent is given, and users should be able to download and view all collected data directly through a web dashboard, without intermediary requests. Simultaneously, the architecture promotes and encourages open data practices and principles, and therefore provides tools that encourage data sharing.

These three considerations guide our architecture's design and implementation, which we discuss in the following section.

### 3.3 Architecture

Our architecture was conceived and designed as an integrated end-to-end pipeline to capture, visualize, anonymize, and share the data generated by educational platforms. The blueprint outlined in Figure 3.1 summarizes our architecture and shows how data flow through the different stages of the pipeline, satisfying the design considerations presented in Section 3.2. In this section, we present the five stages of our pipeline and demonstrate how we implemented and integrated these stages into Graasp.

Our implementation addressed the following research question, which is an extension of **RQ1**:

- Is it possible to integrate our LA pipeline into an LXP? (**RQ1b**)

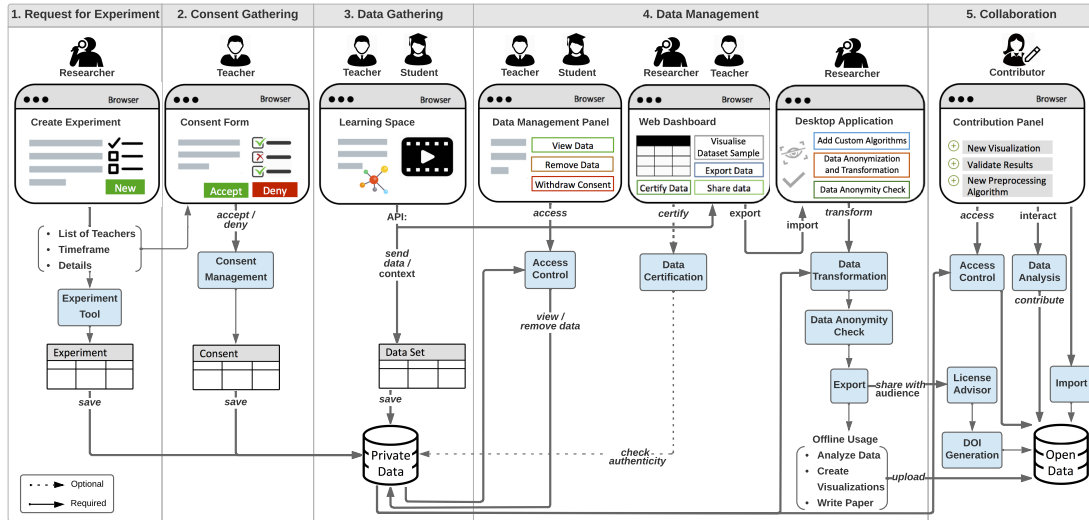


Figure 3.1: The user flow in our architecture involves five main stages. First, there is a request for experiment, followed by consent gathering, data gathering, data management, and finally collaboration. The tools that we propose support stakeholders across all stages and are highlighted in blue.

The design of our pipeline follows a loosely coupled, modular approach. That is, each stage has one or more key components or *tools* that have standalone purposes and can function as discrete units. As an example, certain users may entirely forego the tools described in Sections 3.3.1–3.3.3 and Section 3.3.5 and use only the data anonymization tool described in Section 3.3.4. However, combining the tools into an integrated process offers a more comprehensive set of solutions for working with LA data. Indeed, although many digital platforms incorporate some of these functionalities, one of the innovations in our approach is combining them into a unified and cohesive architecture for managing data over their life cycle.

### 3.3.1 Request for Experiment

The pipeline begins by allowing researchers to bootstrap experiments on the platform. This process is mediated by an *Experiment Tool*, which establishes a connection between researchers and educators. Using the tool, a researcher invites educators to participate in their experiment, specifying parameters such as the experiment’s purpose and duration, the data collected, and privacy and legal disclosures (see Figure 3.2). The tool allows the researcher to distribute this invitation to multiple educators and makes setting up research experiments quick and efficient.

### 3.3.2 Consent Gathering

Having accepted the invitation to participate in the experiment, educators use the *Consent Management Tool* to review the details of the experiment and provide their consent for participation and data collection. The tool timestamps and saves this consent, and therefore acts as a central repository for both researchers and educators to review consent forms. The tool also supports an alternative, reversed workflow in which teachers invite researchers to participate in their experiments. In that case, teachers can specify the conditions under which they accept granting access to their data.

In the proof-of-concept implementation of our pipeline on Graasp, the *Experiment Tool* and the *Consent Management Tool* were presented together in the *Membership via Consent* panel, as shown in Figure 3.2.

**Membership via consent** ✕

When membership via consent is enabled, non-members who access this space through its link will be prompted to accept the terms below. After agreeing to the terms, they become members of the space and can view its contents.

[Copy a link to this space to your clipboard](#)  
[Share a link to this space by email](#)

☒ Enabled

Dear instructor,

You are receiving this invitation to participate in our research experiment investigating how undergraduates studying physics use interactive web-based simulations to build mental models of natural phenomena. The experiment will be conducted over the Fall 2020 semester, with data collection beginning on September 8, 2020, and ending on December 19, 2020. In accordance with our Ethics Policy (link below), the data

50 characters minimum

☒ Only allow users with emails in the following list to access this space via consent:

[Redacted]

Comma separated list

Download consents

Cancel Save

Figure 3.2: The *Experiment Tool* and the *Consent Management Tool* of our architecture were implemented within the *Membership via Consent* panel on Graasp.

### 3.3.3 Data Gathering

Provided consent has been given, the *Data Gathering Tool* enables the platform's learning activities and the applications within them to capture LA. Predefined user interactions with a learning activity or app, such as navigation between lessons or typing text into an input box, generate data points known as *actions*, which are sent to the Data Gathering Tool. This tool validates that the data can be stored before persisting them in the platform's database.



To integrate this Data Gathering Tool into Graasp, we extended Graasp's API to receive, filter, and process user-generated actions. Each action is defined as a JavaScript Object Notation (JSON) object. These objects are based on the Experience API (xAPI) standard (Berg et al., 2016) and contain information about a discrete interaction, including its type, date, time, and content, as well as the username of the user performing the interaction. Actions also contain information regarding the context in which they occurred, such as the lesson, learning phase, or activity, as well as an approximate geolocation based on the user's IP address.

#### 3.3.4 Data Management

The data management stage covers three key areas: (i) data exposure, (ii) data visualization, and (iii) data anonymization.

##### Data Exposure

A *Data Exposure Tool* enables seamless access to the LA data stored on the platform's servers. On Graasp, we implemented this as a dedicated API to expose data in a structured and well-defined format. The API includes endpoints for retrieving data on a learning activity, as well as on the learners who participated in and the actions generated during this activity. Returned as a JSON object, the data can be analyzed by researchers in a well-supported and recognized format, in addition to being easily consumed by frontend client interfaces.

##### Data Visualization

Once the LA data is made accessible via an API, a *Data Visualization Tool* provides a way for educators and researchers to get a high-level idea of the type of information recorded during the data gathering stage. On Graasp, we built a web dashboard to visualize these data (Figure 3.3). The dashboard was designed to provide a high-level overview of any given learning activity and contains charts of actions by day, time of day, and type, as well as an interactive map of actions coarsely clustered by geolocation. Additionally, the dashboard has functionalities that allow users to filter the actions displayed by one or more of the users who participated in the learning activity.

The dashboard is particularly useful for nontechnical stakeholders who would not otherwise be able to manipulate and extract insights from the JSON data returned by the API. For more advanced users, the dashboard includes an export functionality to request and download an activity's complete dataset for offline analysis.

##### Local Data Anonymization

An integral component of the pipeline is a *Data Anonymization Tool* to transform, filter, and anonymize the datasets downloaded via the web dashboard. This tool allows researchers

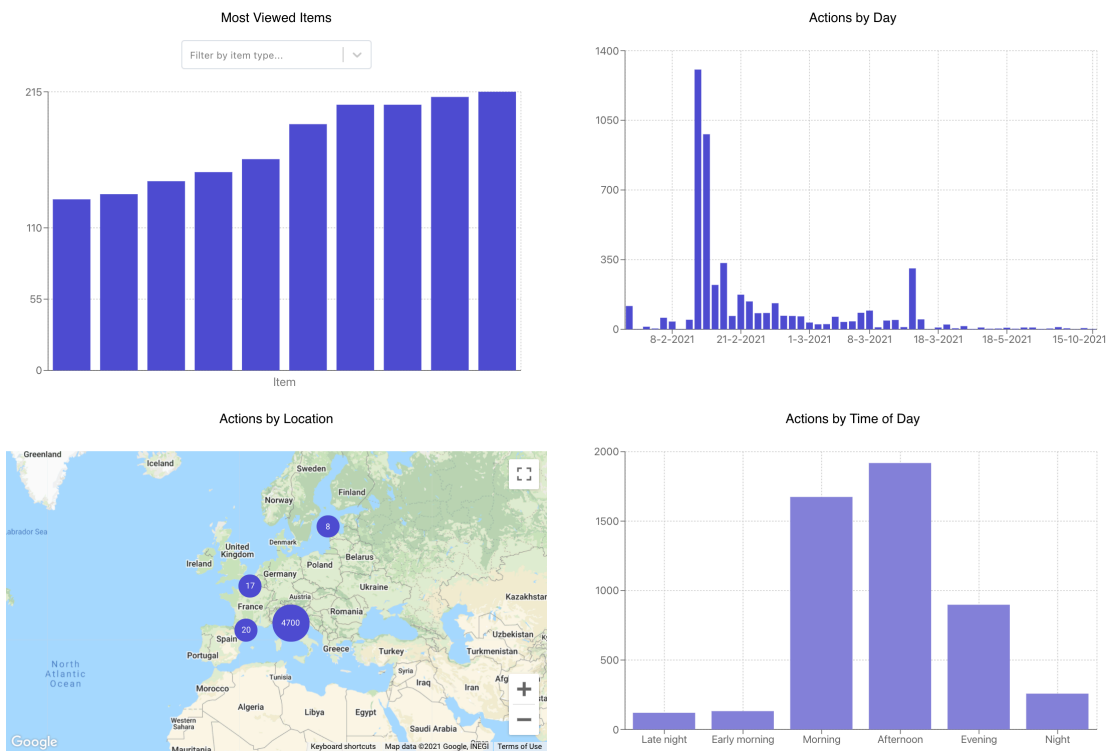


Figure 3.3: A web dashboard serves to visualize the learning analytics data collected by the educational platform, providing high-level insights into a learning activity’s usage for technical and nontechnical users alike.

to make their datasets more open and shareable, for which the ability to anonymize data is critical. In our design, this tool should be implemented as an offline desktop application. The choice of platform for this data anonymization tool was an essential architectural decision for this component of the pipeline, as this allows users to anonymize and process their datasets *entirely in their local environment*, without needing to share them with third-party services.

Our proof-of-concept implementation of this tool on Graasp is a cross-platform desktop application. The application, which we named *Graasp Insights*, works as follows:

1. **Dataset:** The JSON dataset downloaded from the web dashboard is loaded into the application, where it can be viewed alongside summary statistics and visualizations (Figure 3.4).
2. **Schema:** A schema describing the dataset’s structure is generated via the application’s *Schemas* tab. This allows datasets with a given schema to be automatically identified and tagged.
3. **Algorithms:** The application contains a number of precomposed anonymization algorithms written in Python (Figure 3.5). Some of these algorithms are optimized for the

Name	Size	Created	Last Modified	Quick Actions
<b>Thermal Radiation Exercise</b> <span>Graasp</span> Learning traces collected during the lesson on thermal radiation that took place in March 2021.	97.48KB	10/15/2021, 1:15:32 PM	10/15/2021, 1:15:29 PM	<> Download Delete
<b>Moodle</b> <span>Moodle</span> Data exported from Moodle.	219KB	10/15/2021, 2:12:31 PM	10/15/2021, 2:12:31 PM	<> Download Delete
<b>2020 Tweets</b> <span>Twitter</span> A selection of tweets collected during 2020.	219KB	10/15/2021, 2:11:33 PM	10/15/2021, 2:11:33 PM	<> Download Delete

Figure 3.4: The *Datasets* tab, where users load the datasets they want to work with. A schema has been generated for each of these datasets, which are tagged with that schema's label.

dataset schema generated by Graasp, but others work on general datasets. A user can review these algorithms and their code in a dedicated section of the application, where they can also add their own custom algorithms and scripts. A further discussion of these anonymization algorithms follows below.

4. **Executions:** Having loaded their dataset(s) into the application and reviewed and added anonymization algorithms, users proceed to an in-app *Executions* tab where they can run selected algorithms against their dataset(s) (Figure 3.6). Users can view the generated results in the application (Figure 3.7), or export the underlying JSON file to another location on their device.
5. **Pipelines:** In the *Executions* tab, users can run only one anonymization algorithm at a time against a single dataset. Via the application's *Pipelines* tab, users can chain multiple algorithms to be executed sequentially against a dataset, allowing them to efficiently perform various operations (e.g., hash or suppress fields) on their dataset in one quick step.

6. **Validation:** Before making their data available, users can select verification algorithms to run against the anonymized dataset. These algorithms will check for the presence of sensitive attributes. Examples of such algorithms include *k-anonymity* verification, *l-diversity* verification, and *username scan* based on regular expressions, among others (Figure 3.8). If a test fails, additional anonymization algorithms can be applied, as previously described.
7. **Sharing:** Finally, users can share their anonymized datasets and algorithms in a repository of open educational datasets.

As highlighted above, the application contains precomposed anonymization algorithms, some of which are optimized for the platform's schema, and others that can be used on general JSON datasets. The diversity of learning platform data and the complexity of JSON datasets, with deeply nested and potentially inconsistent structures, make composing generalized algorithms a challenging problem and necessitates the composition of platform-specific algorithms. The algorithms built into the application are the following:

- **Data Suppression:** With a dataset's schema identified by the application, users can select the fields to be suppressed (i.e., completely removed) from the dataset. For example, users may choose to suppress the *name* field from the action data, given its highly sensitive nature.
- **Data Hashing:** With the dataset schema as a reference, users can select the fields to which they would like to apply a SHA-256 hash. For example, users may hash the *actionId* field from the action data.
- **Geolocation k-anonymization:** Optimized for the educational platform's schema, this algorithm gradually suppresses the *country*, *region*, and *city* keys within an action's *geolocation* key, until there are at least *k* users containing each combination, satisfying *k-anonymity* (Sweeney, 2002) on the premise that, for the purposes of this algorithm, geolocation is the only identifying attribute.
- **User Cleansing:** Optimized for the educational platform's schema, this multistep algorithm (i) replaces *userIds* in a dataset's *actions* key with a SHA-256 hash of each *userId*, (ii) removes identifying information, which may include names and emails, from a dataset's *users* key.
- **Advanced User Cleansing:** This algorithm scans the entire dataset to detect remaining instances of identifying user information, such as names and IDs, replacing them with hashed versions thereof. This handles edge cases in which the data contains, for example, identifying information captured via user-generated inputs. The scan is performed using regular expressions, hence matching close representations of user information (e.g., *John Doe* matches variations like *john.doe@institution.domain*).

Dataset (Required)

Thermal Radiation Exercise 

Grasp

Algorithm (Required)

k-Anonymize geolocation

Save As...

Thermal Radiation (k-Anonymized)

EDIT PARAMETERS

EXECUTE

Dataset	Algorithm	Result	↓ Executed	Status	Quick Actions
Software Design 2020	Shuffle fields	Software Design 2021 (Shuffled)	10/15/2021, 2:17:11 PM	✓	<div><div>⌵</div><div>🗑</div></div>
Software Design 2021	Hash users	Software Design 2021 (Hashed)	10/15/2021, 2:16:49 PM	✓	<div><div>⌵</div><div>🗑</div></div>

Rows per page: 10 ▾ 1-2 of 2 < >

Figure 3.6: The *Executions* tab, where users can select the datasets they have imported and run pre-composed or custom algorithms against them.

Validations				
Name	Executed Validation	Status	Verified	Quick Actions
Software Design 2021 (Hashed)	Detect users	<span>✓</span> <span>🔗</span>	10/15/2021, 2:32:49 PM	<span>🗑️</span>
Software Design 2020	Verify potentially dangerous attributes	<span>⚠️</span> <span>🔗</span>	10/15/2021, 2:31:44 PM	<span>🗑️</span>
Thermal Radiation Exercise (Shuffled + k-Anonymized)	Verify k-anonymity	<span>⚠️</span> <span>🔗</span>	10/15/2021, 2:30:50 PM	<span>🗑️</span>
	Detect users	<span>✓</span> <span>🔗</span>		

Figure 3.8: The *Validations* tab, where users can run anonymity verification algorithms against the resulting datasets and see the outcomes of the tests (success, warning, or failure).

Where applicable, algorithms were structured to support in-app parameterization, allowing users the freedom to select to which data attributes to apply these algorithms. Within the application, the algorithms are presented alongside plain language descriptions, allowing nontechnical users to understand them. Additionally—in order to promote transparency—the code of each algorithm can be accessed within the application. More technical users can review and edit this code. Finally, as shown in Figure 3.9, the application allows users to add their own custom algorithms, selected from their file system or written using the in-app editor.

**Add Algorithm**

☐ Load algorithm from a file

☐ Add Graasp algorithm

☒ Write algorithm

```
1 from graasp_utils import load_dataset, save_dataset, parse_arguments
2
3 def main():
4     # Prepares the parameters for the algorithm (dataset_path and output_path
5     # come by default). You can then use them with args.parameter_name.
6     # Avoid editing the parameters here, use the dedicated utility instead and
7     # the code will change accordingly.
8     args = parse_arguments([
9         {'name': 'location', 'type': str},
10        {'name': 'age', 'type': int}
11    ])
12
13    # load the json dataset, available as a python dictionary
14    dataset = load_dataset(args.dataset_path)
15
16    # write your dataset changes here
17    save_dataset(dataset, args.output_path)
18
19 if __name__ == '__main__':
20     main()
21
22
23
```

Algorithm name\*  
My Custom Algorithm

(Required)

Description  
This algorithm is custom-made for my data...

(Optional)

Type  
Anonymization

Parameters

Name	Type	Default value
location	String	Lausanne
age	Integer	18

ADD PARAMETER

Figure 3.9: Users can add their own custom algorithms using the built-in code editor. Algorithms can also be loaded from a file.

### 3.3.5 Collaboration

Our architecture proposes a number of tools for collaboration. An *Open Data Repository* is aimed at sharing datasets and associated contextual resources, such as metadata from learning activities that were part of an experiment. Resources and datasets in the repository should support being tagged using a *DOI Generation Tool*, facilitating their citation in subsequent publications. To support researchers in providing the appropriate terms of use for the datasets they are looking to share, a *License Advisor Tool* should provide recommendations for how to license the data. Once a dataset is shared, the *Access Control Tool* should allow researchers to grant or deny access to their datasets according to how open they would like them to be. Optionally, the *Data Certification Tool* could ensure that even if the data is stored outside the platform, its authenticity can be verified through a certification mechanism.

In our implementation, we enhanced the Graasp Library to support publishing data alongside a copy of the learning activity in which the data was generated. We also incorporated a Creative Commons license advisor to support and encourage users to make their data and learning

activities open. Finally, for researchers looking for more granular data sharing, the Graasp Builder supports sharing data with individual users from within the learning activity in which the data was generated.

## 3.4 Discussion and Implications

The three design considerations outlined in Section 3.2 fundamentally shaped our pipeline's architecture. First, the architecture encompassed a complete toolkit for managing experimental data, from seeking appropriate consent to being able to openly share anonymized datasets. Second, every individual component—as well as the pipeline as a whole—was designed to be accessible to nontechnical users, without limiting the tools' ability to also provide more technical users with advanced features and customization. Third, transparency was rigorously enforced throughout the pipeline, from ensuring that data was gathered only after appropriate approvals, to making the data available to users without intermediary requests, to openly displaying the anonymization scripts built into the desktop application. The aim is to encourage feedback and audit reports from the users of these tools, promoting an open assessment mechanism by which our pipeline becomes more robust and at the same time more pertinent to both educators and researchers.

In essence, our architecture provides an extensible blueprint on how a digital education platform can be enhanced to include more comprehensive LA data management tools for its stakeholders. To highlight a few examples, in the outlined deployment we (i) created a direct communication channel between researchers and educators, helping increase transparency in how research is framed and conducted, (ii) provided users with on-demand access to their complete datasets, and (iii) provided researchers with a user-friendly but versatile application to anonymize their datasets. While the architecture targets educator-mediated contexts, its design is readily transferable to other domains, such as digital healthcare or e-government, and could also help support open data in these contexts. We highlight that—by design—the data collection, processing, and sharing process is fully controlled and under the responsibility of the educators and researchers involved. They can personalize each step of the pipeline and validate the integrity of the final datasets using the tools and dashboards provided, or with custom-made visualization and validation algorithms.

We validated our architecture through a proof-of-concept implementation on Graasp. The implementation of our architecture on Graasp had significant practical implications for the rest of the work conducted in this thesis and beyond. First, the end-to-end nature of the pipeline ensured that we (i) had all the necessary tools to manage their data over their life cycle, (ii) did not need to find, configure, and adapt tools designed by different entities, and (iii) benefited from a consistent user experience and interface design, which simplified collaboration with educators and other researchers. Second, consistent generation and access to the data captured by LA simplified our data analysis process, allowing us to make our apps LA-compatible and use the same analysis procedures across experiments. Third, given

that data anonymization is a complex problem (Wagner et al., 2018) and that existing LA anonymization tools are proposed only as proofs of concept (Gursoy et al., 2017), our desktop application is a significant step forward in providing a starting point for anonymization of our datasets. We harnessed these tools for all the empirical studies conducted on Graasp, which span all the contributions put forth in this thesis.

### 3.5 Conclusion

In this chapter, we detailed a comprehensive architecture to manage the acquisition, visualization, anonymization, and sharing of LA data. Although the potential benefits of LA are well documented (Siemens et al., 2011), achieving these benefits is challenging due to the diversity of the stakeholders involved (Sclater, 2016) and the tools required (Machado et al., 2019). Our contribution is to design, develop, and implement an integrated end-to-end pipeline that addresses these diverse stakeholders and their needs and, in doing so, helps promote open data practices. The implementation of this pipeline had important practical implications for the rest of the work conducted in this thesis, serving to manage the data generated in our empirical evaluations and further emphasizing the need for collaboration between learners, educators, developers, and researchers when designing tools for education.

One of the key limitations of the deployed architecture, as it stands, is its relatively narrow scope. Specifically, the desktop application is optimized for the anonymization of datasets generated by Graasp. Although some of its algorithms are generalized and can be used on any dataset, more complete anonymization continues to require familiarity with the underlying dataset schema. Further generalization of the desktop application's anonymization algorithms is an area of work with significant potential. User interface and workflow design can allow researchers to guide the application in (i) identifying and understanding a dataset's schema and (ii) tagging its quasi-identifying and sensitive attributes, after which generalized algorithms can be more accurately deployed, as implemented in ARX (Prasser et al., 2020). More extensive tests and validation regarding the effectiveness with which a dataset has been anonymized are other promising and important areas for future work. Even after removing direct identifiers from a dataset, it is critical to perform deidentification checks to ensure that no record can be used to identify an individual. To that end, the application could introduce comprehensive data anonymity checks that quantify the privacy levels attained after applying anonymization algorithms, displaying such outcomes to users. In particular, deep learning solutions in the domain of natural language processing have been recently proposed for generically detecting values in text that could potentially be sensitive, achieving promising results (Hassan et al., 2019; Xu et al., 2019). Those solutions could be an important asset in digital education but have not yet been tested in this domain. Finally, as data literacy grows, there is significant potential in making the desktop application available directly to individual users, who can use the application to view, understand, and potentially anonymize the personal datasets they acquire via *subject access requests* on the internet platforms they use. As issues surrounding data privacy become more widespread, educating users about



the nature of the data that are collected by online platforms—and helping both researchers and end users protect these data—could assist in the prevention of accidental disclosures of sensitive information. Following these and other use cases, we believe that our pipeline’s tools can inform the design of responsible data collection and dissemination policies not only for digital education but also for other domains where guaranteeing user privacy is essential.



# **Pedagogical Scenarios** **Part II**



---

IN the second part of this thesis, we investigate how to harness our technological foundations to design the pedagogical scenarios that will be used for our empirical evaluations with educational chatbots. As outlined in Chapter 1, our aim was to integrate chatbots into software engineering education. We start by exploring a preliminary pedagogical scenario resembling computational notebooks using *Code*, an app focused on writing and executing Python code in the browser. In Chapter 4, we present this app, the introductory programming pedagogical scenario it supported, and a case study conducted with 67 students at the University of Neuchâtel. We then introduce the *code review notebook*. The code review notebook is supported by a series of apps built with our application development framework and, particularly, by *Code Review*, one of the apps that will host the educational chatbots studied in this thesis. In Chapter 5, we present the design of *Code Review* and the code review notebook template, as well as an empirical evaluation with 25 university students addressing how code review notebooks could serve to teach programming best practices. In this brief introduction, we highlight related work.

## Related Work

Introductory programming courses are prime candidates for blended learning (Boyle et al., 2003) and the simplicity and readability of Python have made it an attractive introductory programming language (Radenski, 2006). For instance—in the context of learning programming—Chu et al. (2018) used a blended learning approach and discovered that the majority of their students accessed their learning platform outside of class to study and improve their Python programming knowledge. Although there are a large number of online Python tools available (Kim et al., 2017), computational notebooks (e.g., Jupyter Notebooks) have become common in introductory Python courses (Cardoso et al., 2018; Chapman et al., 2015; Zastre, 2019). Computational notebooks are online tools that combine resources (such as text or images), executable code, and both textual and graphical outputs. The combination of an online coding environment that does not require external software and the possibility to run code

---

embedded in text and multimedia content is particularly well suited to teach introductory computer science topics, such as computational thinking (O'Hara et al., 2015).

Computational thinking can be defined as the different thought processes used in computer science to solve problems (Yadav, Stephenson, et al., 2017). Among the main concepts used are modeling problems using abstractions, division of problems into subproblems, design of solutions through sequential steps (algorithms), and identification of patterns. Over the past decade, computational thinking has become a tool to solve problems in virtually every field of study (Barr et al., 2011). Learning computational thinking thus becomes crucial not only for engineers and computer scientists but also for students in domains outside science, technology, engineering, and mathematics (STEM). In line with this trend, computational thinking courses have been introduced not only for a wide array of university degrees but also in secondary schools (Grandell et al., 2006; Yadav, Zhou, et al., 2011) and all the way down to early childhood education (Bers et al., 2014).

Previous work has explored the usage of online notebooks to teach computational thinking in different learning activities. For instance, O'Hara et al. (2015) evaluated its usage for (i) lectures, (ii) reading, (iii) homework, and (iv) exams. Typically, among the opportunities offered by applications such as Jupyter, students can iterate on their coding assignments on the same platform without the need to switch between the assignment and the coding software (Project Jupyter et al., 2019). Jupyter also includes several tools specifically designed for teaching purposes, such as grading modules.

It should be noted that such notebooks can also have a negative impact on learning, as some argue that they promote poor coding practices because they make it difficult to break code into smaller reusable modules and develop and run tests on the code (Perkel, 2018). Furthermore, there is a tension between exploration and explanation, as it requires a lot of effort for a user to convert a messy exploratory notebook into a clean shareable notebook (Rule et al., 2018). Moreover, such environments still lack support for a wider range of interaction, collaboration, activity awareness, and access control mechanisms (A. Y. Wang et al., 2019). Finally, although computational notebooks are valuable for beginners, they may be inadequate for experienced users (Borowski et al., 2020; Chattopadhyay et al., 2020).

To address these challenges, notebooks can be personalized according to learning style, programming level, or learning context (O'Hara et al., 2015). Other types of personalization for computational notebooks have been undertaken in the context of artificial intelligence classrooms. For example, O'Hara et al. (2015) developed their own computational notebook systems—called ICalico and Calysto—and explored their use with their students. Aside from Jupyter, other approaches focus on integrating smart content hosted on different servers to enhance the learning experience (Brusilovsky et al., 2018), while several web-based tools for teaching Python have also been proposed (S. H. Edwards et al., 2014; Guo, 2013; Pritchard et al., 2013).

It is important to note, however, that self-hosted computational notebook solutions such

---

as Jupyter often require a backend server infrastructure to execute code and manage users, and directing students to cloud-based solutions such as Google's Colaboratory (Bisong, 2019) risks violating privacy and legal regulations—such as the European General Data Protection Regulation (GDPR) (European Union, 2016)—which many institutions are required to adhere to. Therefore, there is an opportunity to support less technical educators and learners with more accessible computational notebook solutions. These solutions could be integrated directly into digital learning platforms to provide technopedagogical scenarios for introductory software engineering education.

A pedagogical tool that is particularly relevant to software engineering education is the code review process. The importance of code reviews in software engineering education has long been recognized (Towhidnejad et al., 1996). A study conducted by X. Li et al. (2005) on effectively teaching coding standards in programming revealed that most students believe that coding standards are important. However, students still fail to comply with them, thus highlighting possible flaws in current teaching strategies. To that end, the authors suggested code review as a possibly effective pedagogical tool, given that it provides an element of learning by example and practice, as well as an opportunity to obtain feedback from teachers and other learners. Another study conducted by Bacchelli et al. (2013) on Microsoft employees highlighted additional benefits of code review, such as knowledge transfer, increased team awareness, and the creation of alternative solutions to problems.

The most common approach to incorporating code reviews into educational contexts is to include them as a *peer* review exercise. Peer review has been championed as a way to teach students to provide and receive constructive criticism (Anewalt, 2005), as well as to build collaboration skills (Trytten, 2005). In a recent systematic review of the literature, Indriasari et al. (2020) reported that the first “use of peer code review in a programming course was published in 2003”, with 187 studies published as of April 2019. This review highlights the use of social coding platforms or bespoke tools to incorporate the code review process into learning activities. Examples of these bespoke tools include CaptainTeach (Politz et al., 2014), EduPCR (Zong et al., 2021), and Caesar (Tang, 2011). These tools are specifically designed for peer code review. Indeed, most tools used to incorporate code review into education focus on supporting the peer code review use case, providing features such as automatic review assignment and anonymous reviews (Indriasari et al., 2020). For example, Y. Wang et al. (2012) proposed an online assessment system based on code review to assess how students learn programming languages. Empirical results from their study showed that the assessment based on code review significantly improved student learning outcomes in several areas such as programming skills, collaborative learning, and compliance with coding standards. Positive results were also obtained when code review activities were tested in high school and university settings (Kubincová & Csicsolová, 2018; Kubincová & Homola, 2017).

While peer code review is particularly useful for students to gain practical experience performing and receiving these reviews, there are some barriers to the adoption of peer code review activities in education. In fact, it has been reported that in some peer code review studies, stu-

---

dents did not have the skills necessary to satisfactorily complete code reviews (Indriasari et al., 2020). If students are expected to review code submitted by their peers, they should be able to do so satisfactorily, as otherwise, the exercise is neither useful for the student conducting the review nor for the student receiving the feedback.

A few tools focus on providing individual students with a demonstration of the code review process. Alves (2013), for instance, developed *Source COde REview Learning (SCOREL)*, a serious game consisting of a desktop application “to streamline and encourage the practice of code review”. SCOREL was later migrated to web technologies and adapted to support collaborative reviews by Ribeiro Guimarães (2016). Another serious game aimed at learning to perform code reviews was proposed by Ardiç et al. (2020). In this game, players review predefined code snippets, identifying lines containing defects in order to advance to the next level. X. Song et al. (2020) developed a tool for peer code review, which they also used to introduce the code review process by having students review code that had already been graded. Although these tools address the need to support students in learning to conduct code reviews, they are standalone applications that are not designed to be embedded directly within an online lesson or lecture, limiting their compatibility with LXPs and other online learning technologies.

Furthermore, the rising popularity of social coding platforms as pedagogical tools has resulted in students being exposed to professional code review tools and processes as part of their education. In fact, the use of social coding platforms in education has been widely studied (Feliciano et al., 2016; Fiksel et al., 2019; Glazunova et al., 2021; Haaranen et al., 2015; Zagalsky et al., 2015), with a recent survey of 7530 students and 300 educators concluding that the use of GitHub “predicted better learning outcomes and classroom experiences” (Hsing et al., 2019). However, several researchers have highlighted a number of challenges associated with the use of social coding platforms in education, including steep learning curves and privacy considerations due to the often public visibility of the interactions on such platforms (Feliciano et al., 2016; Fiksel et al., 2019; Zagalsky et al., 2015).

Given the lack of tools to integrate the code review process directly into LXPs and the ubiquitous use of social coding platforms for code review, there is therefore an opportunity for the development of technological tools that are able to replicate the core functionalities provided by social coding platforms and can be integrated into educational contexts. These tools could, in turn, support pedagogical scenarios aimed at teaching the code review process, addressing the need to better prepare students for peer code review exercises.



## 4 Integrated Computational Notebooks

WE start our exploration of pedagogical scenarios suitable for integrating educational chatbots into learning experience platforms (LXPs) by considering the computational notebook. Computational notebooks, such as Jupyter, are popular solutions to develop the programming skills typically taught in introductory software engineering and computer science courses. However, these solutions often require technical infrastructure and lack support for rich educational experiences that integrate discussion, active feedback, and learning analytics (LA). In this chapter, we introduce an app designed to address these challenges. We present blended learning scenarios supported by this app and evaluate them in an eight-week computational thinking course comprising 67 students pursuing a Bachelor of Science in Economics and Business at the University of Neuchâtel, Switzerland. We include in our results the impact of the disruption caused by the COVID-19 pandemic, which forced a move from blended to online distance learning for the second half of our evaluation.

By proposing the first iteration of the pedagogical scenario used to integrate educational chatbots into learning experience platforms (LXPs), this chapter partially addresses **RQ2**:

How can we scaffold pedagogical scenarios suitable for educational chatbots?

The content of this chapter was partially presented in the following publication:

- Farah, J. C., Moro, A., Bergram, K., Purohit, A. K., Gillet, D., & Holzer, A. (2020). Bringing Computational Thinking to non-STEM Undergraduates through an Integrated Notebook Application. In T. Broos & T. Farrell (Eds.), *Proceedings of the Impact Papers at the 15th European Conference on Technology-Enhanced Learning (EC-TEL 2020)*

### 4.1 Introduction

An essential part of computational thinking is a basic understanding of programming and therefore courses that introduce computational thinking skills to students often include a programming component (Lye et al., 2014). Due to the complexity of providing a consistent experience across different devices and operating systems, introductory programming courses have traditionally required a technical setup to ensure that all students are running the same development environment (Chapman et al., 2015). This can be a high barrier to entry for less technical students, teachers, and even institutions lacking the proper information technology support. To lower this barrier, computational notebooks have been proposed as a way to minimize the amount of technical setup needed to provide a homogeneous programming environment. However—as mentioned in the introduction to Part II—there are technical and regulatory challenges associated with current computational notebook solutions. Furthermore, these solutions do not easily integrate with LXPs and customizing such notebooks to allow them to support educational chatbots is not trivial.

With these concerns in mind, we exploited the Graasp ADK presented in Chapter 2 to design a novel app that allows students to run Python code directly in the browser. This app is free and open source and can be integrated into online learning platforms—along with collaboration and LA tools—to offer features present in computational notebooks and foster rich learning experiences. We refer to these computational notebooks, which we integrated into the Graasp LXP, as *integrated computational notebooks*.

This chapter addresses a particular need that was presented in the introduction to Part II. That is, we directed the design of our app to enable computational notebooks that can be integrated into learning environments in an attempt to lower the barrier to entry to these technologies for less technical learners and educators. To validate this design, we conducted an analysis of learner interactions with the app and with the learning environment in which it was deployed. This analysis also contains insights into how the switch from a blended learning scenario to a distance learning scenario impacted usage. In addressing this need, we put forth one of our contributions, namely a template to provide browser-based computational notebook solutions within digital education platforms (**C3**). Furthermore, our design paves the way for apps that can support pedagogical scenarios for software engineering education that are capable of integrating educational chatbots.

This chapter is structured as follows. We present the design of our app and the integrated computational notebooks that it supports in Section 4.2. In Section 4.3, we outline the methodology followed for our empirical evaluation. We present the results of this evaluation in Section 4.4 and discuss these results, as well as their implications on the rest of the work carried out in this thesis, in Section 4.5. Finally, we conclude in Section 4.6.

## 4.2 Design

In this section, we present the design of our app. We then describe how it enables educators to scaffold pedagogical scenarios following the integrated computational notebook template to provide a variety of learning activities for learners.

As noted in Section 4.1, we developed an open source web app (henceforth *Code*) to provide a ready-made Python environment for educators and learners. Code leverages the *Pyodide* library to execute Python directly on the browser without any additional dependencies. It supports reading and writing files, receiving input from users, and displaying graphical output from libraries such as *Matplotlib* (Hunter, 2007). The app also features a CLI simulator that serves both to display output and to allow students to navigate a virtual file system. In its simplest form, Code can be used independently of any other software simply by accessing a link. Nevertheless, it can leverage APIs exposed by LXPs to enable advanced features, as well as LA. To enable these features and to provide a context resembling computational notebooks, we designed Code to be compatible with Graasp.

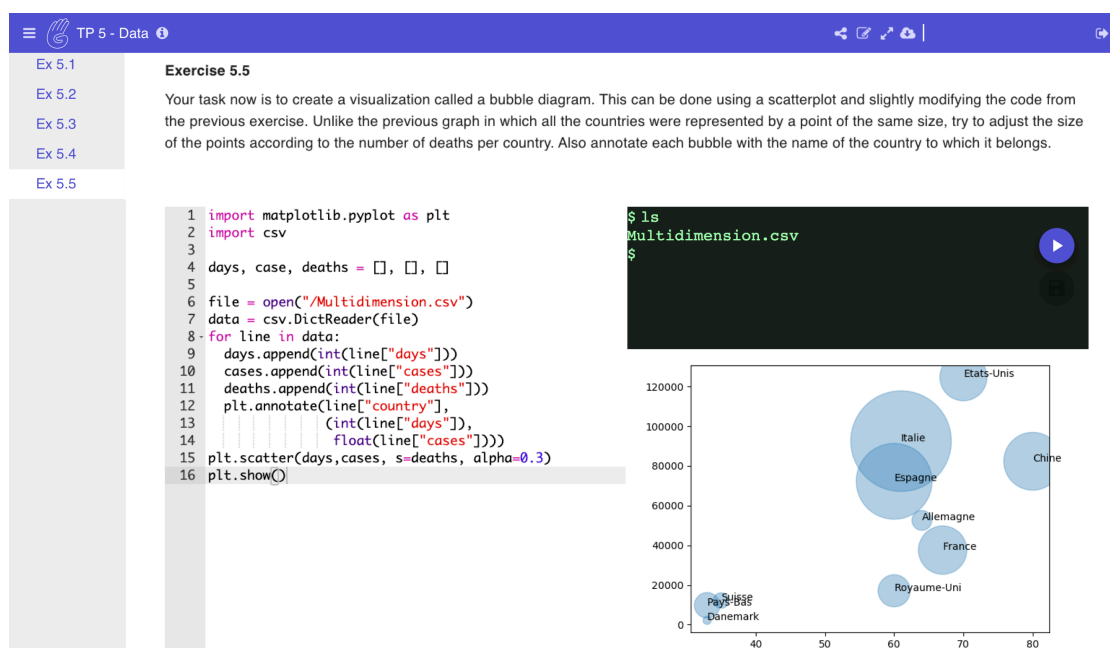


Figure 4.1: A computational notebook learning activity on Graasp. Students can write and execute code, navigate a virtual file system, and view graphics.

As noted in Chapter 2, most of the interactions on Graasp occur in two of its interfaces. The first interface is the Graasp Builder, where educators integrate and configure the resources they will use to create their online lessons, which we refer to as *learning activities*. Learning activities can be scaffolded into step-by-step code exercises, which can be contextualized with text, images, links, chatrooms, and other interactive content. Within this educator-centric view, Code can be preconfigured with sample code, data files, and instructions for learners. It also features a feedback functionality that allows educators to review the code submitted by

## Chapter 4. Integrated Computational Notebooks

each learner and provide comments in a way that is similar to code reviews on social coding platforms such as GitHub.

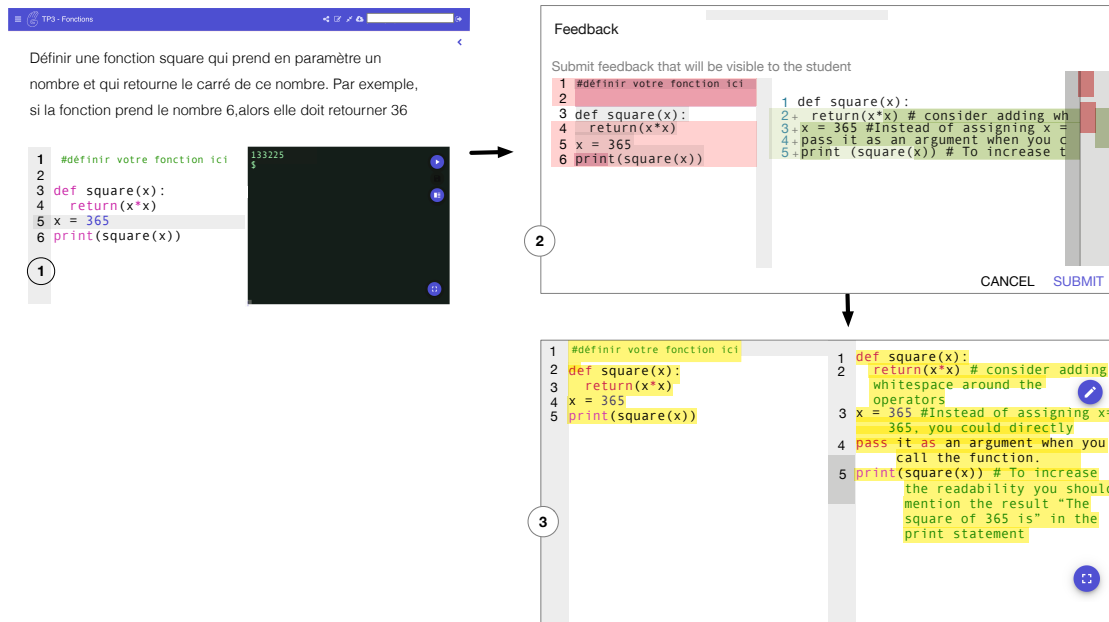


Figure 4.2: Feedback feature in the learning platform. (1) Learners write code in the app. (2) Educator gives feedback to the learners. (3) Learners can view the feedback received.

The second interface is the Graasp Player, which is an environment directed at learners. By accessing this environment through a unique link, learners can take part in the online lesson, navigating through pages containing the exercises prepared by the educator. Within this learner-centric view, Code enables learners to write, execute, and save code, review feedback provided by the educator, and visualize graphics. The result, as shown in Figure 4.1, is a learning activity that resembles a computational notebook and is integrated into the Graasp LXP.

These integrated computational notebooks support active learning as defined in Chapter 1. During lectures or while watching videos or reviewing slides, learners can execute code and test results using Code. Graasp Player also supports a presentation mode, which the educator can use to guide the learners through the learning activity. Several tools can be included within the learning activity to support formative assessment. A simple input app allows learners to submit text, while a real-time communication app enables learners to spontaneously ask questions and respond to multiple-choice questions posed by the educator. Educators can also use the feedback feature inside Code to provide comments to learners.

Finally, through the analytics features of the learning activity, educators can access an overview of the progress and difficulties learners are encountering, and thus adjust their teaching accordingly. As an example, Figure 4.3 shows a learning dashboard to track user activity. More specifically, it shows the order in which each learner has visited the pages available in the

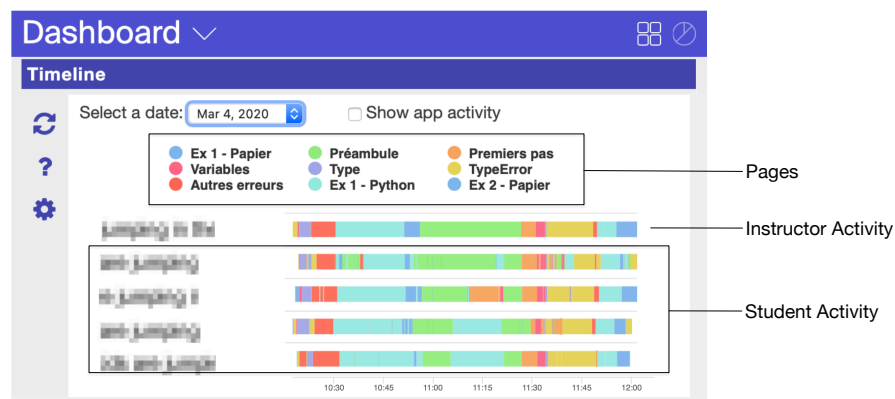


Figure 4.3: A learning analytics dashboard to track how learners interact with an activity.

Graasp Player, as well as the time spent on each of them. If educators use the Graasp Player at the same time, then their data can be compared against the learners' data. Each color represents a page inside the Graasp Player. Were learners to be perfectly synchronized with the educator, their color patterns would all be the same.

### 4.3 Methodology

To better understand how Code could support the acquisition of computational thinking skills, we put forth the following research question, which is an extension of **RQ2**:

- How do non-STEM students in introductory programming courses use and perceive integrated computational notebooks as a tool for learning programming? (**RQ2a**)

To address this question, we incorporated integrated computational notebooks supported by Code into a computational thinking course for students pursuing a bachelor's degree in economics and business at the University of Neuchâtel, Switzerland (henceforth the *university*). Our initial goal was to analyze the usage and perception of the app within a blended learning scenario. This goal focused on two aspects:

1. Engagement: *How did students interact with the integrated computational notebook?*
2. Usability: *How usable were the tools comprising the integrated computational notebook?*

However, at the end of the fourth week of our study, the university had to shut down due to the COVID-19 pandemic, forcing us to adapt the course to a purely online distance learning scenario. This unexpected turn of events prompted a third aspect of our analysis:

3. Modality: *How is the usage of our computational notebooks different between a blended and a distance learning scenario?*

These three aspects of our research question guided our evaluation.

### 4.3.1 Scenario

The evaluation took place in a first-year course on information technologies for students enrolled in the Bachelor of Science in Economics and Business at the University of Neuchâtel. A total of 69 students were enrolled, 31 of them female (45%). Two of the 69 students did not opt in for the study; therefore, their data was removed. The course lasted one semester (14 weeks) and consisted of two periods (1.5 hours) of weekly lectures and two periods of weekly lab sessions with exercises. Student presence in class was not mandatory. The first half of the semester, which is the focus of this study, covered computational thinking, with two weeks for general theory about concepts (e.g., abstractions, problem division, algorithms) and six weeks of introduction to programming with Python to put the theory into practice (see Table 4.1). Note that during these eight weeks, teaching was dramatically impacted by the COVID-19 pandemic. Indeed, all in-class lectures were suspended at the end of the fourth week of the semester and all teaching was moved online. Teaching in the course progressed through three main phases, as outlined below.

Week	Date	Lecture (Wednesdays)	Lab (Thursdays)	Teaching Style	Data Collected
1	17.02	Concepts 1/2	Game	In Class	Video, Pre-Survey
2	24.02	Concepts 2/2, Python Basics 1/2	Game, Group Activities	In Class Blended	Video, Activity Traces
3	02.03	Python Basics 2/2	Start Lab 1	In Class Blended	Video, Activity Traces
4	09.03	Python Lists	Solution Lab 1, Start Lab 2	In Class Blended	Video, Activity Traces
<b>16.03.2020, COVID-19 Confinement starts. No more in-class lectures or lab sessions after that date.</b>					
5	16.03	Python Functions	Solution Lab 2, Start Lab 3	Distance	Video, Activity Traces
6	23.03	Python Dictionaries	Solution Lab 3, Start Lab 4	Distance	Video, Activity Traces
7	30.04	Python Graphs	Solution Lab 4, Start Lab 5	Distance	Video, Activity Traces
8	06.04	—	Solution Lab 5	Distance	Post-Survey

Table 4.1: Structure of the Information Technologies Course

#### Phase 1: Concepts (*in class*)

The first phase covered the first week and a half and consisted mainly of in-class lectures with in-class interactive activities. Exercise (or lab) sessions also took place in class and focused on getting familiar with algorithmic concepts using a game—Human Resource Machine (Tomorrow Corporation, 2015)—as well as through practical group exercises (e.g., designing an analog algorithm to find the most frequent word in a text that was handed out on a piece of paper).

#### Phase 2: Python (*blended*)

The second phase covered the end of the second week and the two weeks that followed and consisted of blended learning, both for lectures and lab sessions. During the lectures, the

instructor used the presentation mode of our learning activities. Concretely, the instructor logged in to the Graasp Player and moved from one page to the next, typing and executing code while providing explanations. Meanwhile, students logged in to the same learning activity using their own credentials and thus accessed their own version of the exercises, where they could write and execute code while the instructor was giving the presentation. During the lectures, a real-time communication app was integrated into the learning activity. Students asked questions and the instructor conducted several polls to see if the level of the course was adequate. During the lab sessions, students were given a learning activity with five questions, each one containing Code as well as an input box to provide answers to the questions. The solutions to each exercise were presented the following week. During these first two phases, lectures were recorded and posted on the university's learning management system (LMS).

### **Phase 3: Python (*distance*)**

The third phase was not planned and was triggered by the national response to the COVID-19 pandemic. As the government imposed partial confinement, the university had to cancel its on-site lectures. For this last phase, lectures were prerecorded and published on the LMS. Lab instructions were published on the LMS and a video explaining the solutions was recorded and posted on the LMS a week later. Student interaction with teaching assistants (TAs) and the instructor took place principally (i) through email, (ii) through the communication app, and (iii) through the feedback feature of Code. We offered to conduct live sessions using a video conferencing tool, but there was no interest from students.

### **4.3.2 Participants**

At the beginning of the course, students were informed of our study and were asked to opt in to participate. The learning experience was identical for both those who opted in and those who did not. Of the 78 students who participated in the course, 69 were officially registered, of which 67 provided their consent for participation in this study. Data from students who did not opt in were excluded from this study.

### **4.3.3 Instruments**

We employed three types of methods to collect data: online activity traces, tests, and questionnaires.

We used three forms of activity traces. The first was a measure indicating the number of edits (e.g., keystrokes, deletions, copy/paste actions) that a student performed within the instances of Code embedded in the learning activities. We refer to this measure as the *code interaction* metric. The second form of activity traces was linked to the in-class video recordings and the screencasts for distance learning, which were uploaded to the university's video repository service. The service tracked each time a student logged in and recorded how long a video

was played. We refer to this metric as the *time spent watching videos* metric. The third form was generated by tracking how students moved between the different pages of our learning activities, as visualized in Figure 4.3.

At the beginning of the course, students were asked to fill in an optional pre-survey about their programming experience and their attitude toward learning technologies. This pre-survey also included an ungraded test assessing their Python knowledge. Students were not informed of their performance. At the end of Phase 3, an optional post-survey was conducted. The post-survey included an ungraded post-test. This test followed the same format and included some of the questions posed in the pre-test. This time, students were informed of their performance.

To assess how students rated the learning technologies used, the post-survey included the System Usability Scale (SUS) (Bangor, Kortum, et al., 2008), which concerned the Code app, and a custom three-item questionnaire concerning the other components of the integrated computational notebook. The SUS is a ten-item questionnaire using a five-point Likert scale to measure usability. Results are presented as scores ranging from 0 to 100, with higher scores representing better usability. The questions in the custom three-item questionnaire followed a five-point Likert scale from *Strongly Disagree* to *Strongly Agree*: (i) *I think the use of the interactive slides on Graasp during the course was useful*, (ii) *I think the use of the chat feature on Graasp was useful for the course*, and (iii) *I think the feedback feature on Graasp was useful*. Finally, students were presented with an open question: *In your opinion, what are the pros and cons of the digital technologies used in this course?*

### 4.3.4 Data Analysis

Our data analysis comprised both quantitative and qualitative methods. We applied both descriptive and inferential statistics to our quantitative data, reporting the sample sizes ( $n$ ), sample means ( $\bar{x}$ ), medians ( $\tilde{x}$ ), and standard deviations ( $s_x$ ), as well as Pearson correlation coefficients ( $r$ ), results from Kruskal Wallis H tests ( $H$ ), and one-sample Wilcoxon signed-rank tests ( $Z$ ). Qualitative data were analyzed by articulating emergent themes using line-by-line data coding (Charmaz, 2006).

## 4.4 Results

In this section, we present results with respect to the three aspects considered in our evaluation.

### 4.4.1 Engagement

Use of Code—as measured by the code interaction metric—varied widely ( $\bar{x} = 22167$ ,  $\tilde{x} = 23990$ ,  $s_x = 13223$ ). However, as illustrated in Figures 4.4–4.5, usage was not significantly correlated with students’ self-reported programming experience ( $r = -0.129$ ,  $p = 0.309$ ), and a



Kruskal-Wallis H test found no significant differences by gender ( $H = 0.0228$ ,  $p = 0.880$ ). The number of days that students interacted with Code throughout the duration of the course also varied widely across students. On average, students actively interacted with Code on nine different days ( $\bar{x} = 8.925$ ,  $\tilde{x} = 9$ ,  $s_x = 4.831$ ).

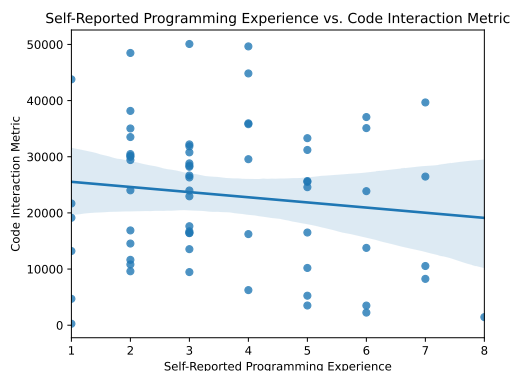


Figure 4.4: Self-reported programming experience at the beginning of the study was not a predictor of the code interaction metric.

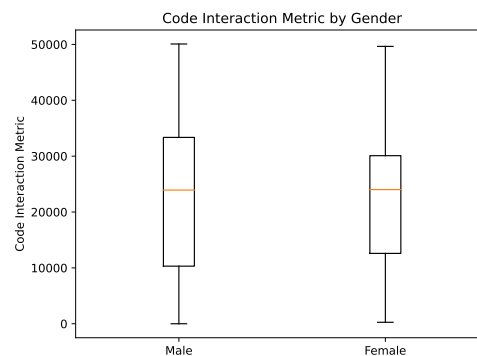


Figure 4.5: There were no significant differences in the distribution of the code interaction metric by gender.

#### 4.4.2 Usability

The SUS score ranges from 0 (worst) to 100 (best). Code achieved a mean score of 71.3 ( $n = 57$ ), which indicates good usability (Bangor, Kortum, et al., 2009). A total of 56 students responded to the three individual questions about the tools comprising the integrated computational notebook. A one-sample Wilcoxon signed-rank test indicated the median for the first item was significantly different from three (the neutral position),  $Z = 6.57$ ,  $p < 0.001$ , with a very strong effect size ( $r = 0.87$ ). Similar significant results were achieved for the second and third items, with  $Z = 4.58$ ,  $p < 0.001$ ,  $r = 0.61$  (strong effect size) and  $Z = 4.47$ ,  $p < 0.001$ ,  $r = 0.59$  (moderate effect size), respectively.

A total of 41 students provided open-ended comments on the tools integrated into the computational notebook learning activities. Here, we discuss the major themes that emerged.

*1. Easy to use.* The first theme that emerged was that Code was easy to use, easy to understand, and easy to get used to. One student explained: “You get used to the [code app] service quite quickly.” Another commented that “the positive points are the ease of use of [Code], the clarity and the stability of the service”.

*2. Ready-made.* Students also appreciated the ability to execute Python in Code without any installation requirements. One student commented: “The biggest positive point for me is that we didn’t need to install an application. We can easily access the [code app] service. Compared to R, it is easier to use and more modern.” Another student noted: “Easy to use / nothing to

install / nothing is saved on our computers”.

3. *Mirroring*. A third theme worth noting focused on the possibility for students to mirror what the teacher was doing using the learning activities. One student reported: “We can put the examples we have seen into practice and we can check by ourselves the explanations given to us during the course work. I find that it puts our computational thinking into practice and motivates us to move forward in this course”. Another student commented that “the use of [Code] during the course allows for a better understanding of the course. You don’t just listen, you already assimilate the material”.

4. *Multiple access*. Several students asked for greater flexibility to access multiple learning activities in parallel. Codes such as *multitask* and *parallel control* were recurrent. One student reported: “Can’t use multiple learning activity pages without one page closing.” Another student commented that “[Code] does not open for the lecture and for the lab, you still have to [sign in] and it’s painful, it’s either one or the other”.

### 4.4.3 Modality

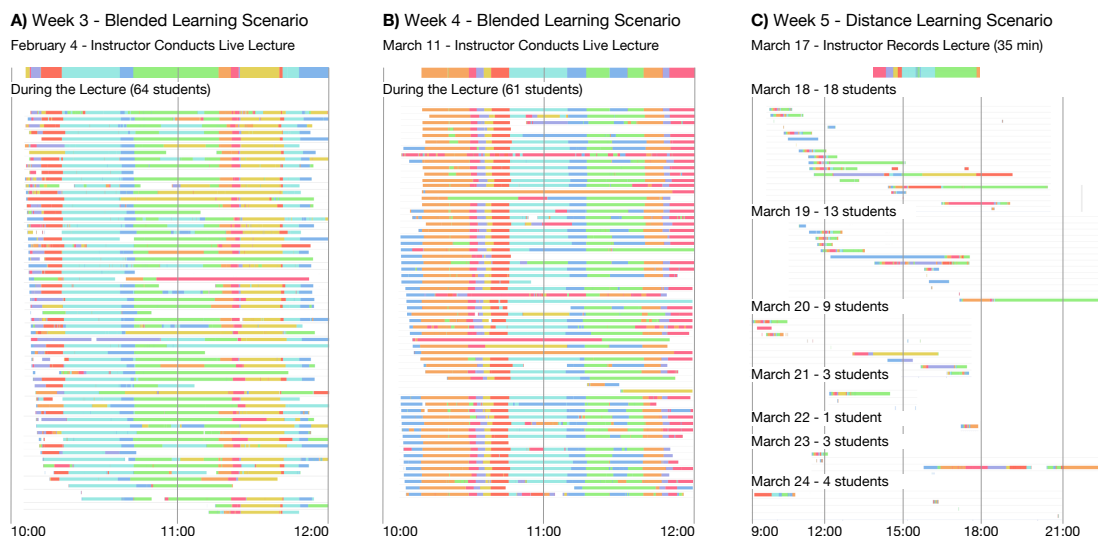


Figure 4.6: Comparing blended (A, B) and distance (C) learning scenarios. The educator’s activity is shown on a thick line above and each thin line represents a student.

Using data from the learning dashboard presented in Figure 4.3, we examined usage patterns from Week 3 and Week 4 (blended learning), and Week 5 (distance learning) (Figure 4.6). In the *blended learning scenario*, where the learning platform is used by the educator and the students at the same time, the dashboard gives the educator a visual impression of how synchronized students are during lecture. Figure 4.6 (A) includes all student activity during Week 3, showing that 64 students, including a teaching assistant, were active at some point in the lecture. A visual analysis considering only active students—not those who left the class early or arrived late—indicates that there seem to be only 5–6, approximately 10% of

all students, who are not following the general page change pattern. Figure 4.6 (B) shows the dashboard for Week 4. The results are very similar to Week 3, with 61 students active on the platform, most of them—except around 5 students—following the educator’s pattern closely. Note that for both weeks we counted the number of students physically present at the beginning of the second part of the lecture to be 53 (24 female) on Week 3 and 50 (25 female) on Week 4. These student counts are in line with the number of students observed online and convey the fact that the integrated computational notebooks were used virtually by all students present in class.

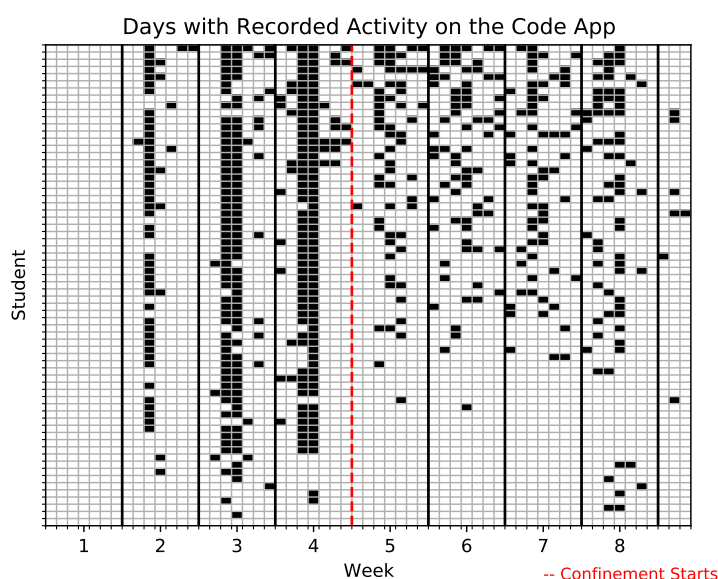


Figure 4.7: Days in which there was recorded activity on Code. Each horizontal line represents a student and each block represents a day in which the student recorded an interaction. A change in use is evident after the switch to distance learning.

In the *distance learning scenario*, the dashboard allows us to see when students logged in to the online platform to work on the course. Figure 4.6 (C) shows an overview of the lecture during Week 5 (March 17). The educator’s video recording of the lecture was 35 minutes long (the content was not changed compared to a 90-minute live lecture, however, the online recording did not include interaction time). Once the video was posted, students could access it at their discretion. Figure 4.6 (C) also shows all activity on the platform related to that particular integrated notebook for the whole week after the video was posted. In total, 51 students accessed the notebook and 29 spent at least 30 minutes on the platform. As expected, the usage pattern is not synchronized across students. Nevertheless, there is a diminishing trend of active students per day, with 18 students accessing the learning activity on the scheduled lecture date (Wednesday, March 18), 13 students the day after, nine students on the Friday, and a minority of students over the weekend, through to the following Monday.

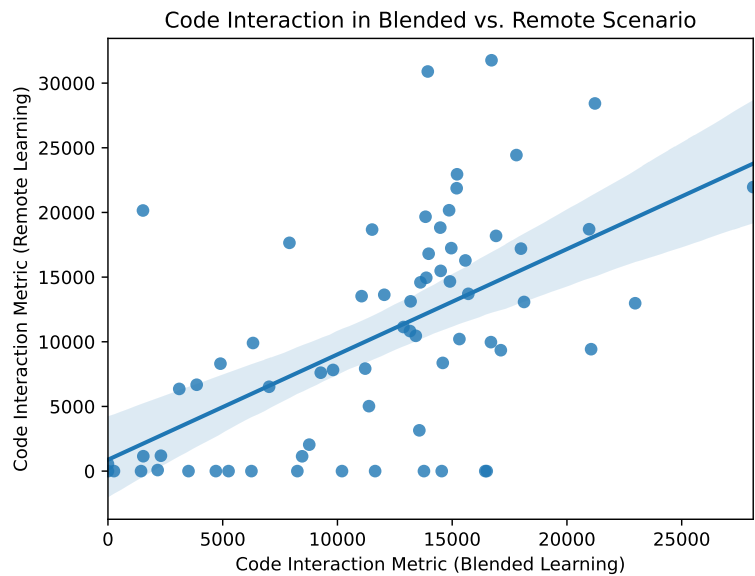


Figure 4.8: How much a student interacted with Code in the blended learning scenario predicted how the student would interact in the remote learning scenario.

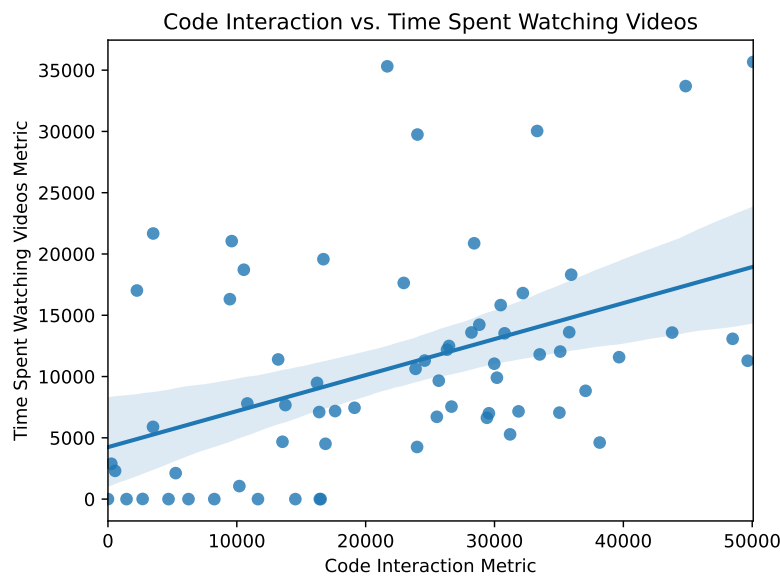


Figure 4.9: The code interaction metric and time spent watching videos metric presented a strong positive correlation.

Regarding the Code app specifically, as shown in Figure 4.7, before the switch from blended to distance learning, usage was concentrated on Wednesdays and Thursdays, coinciding with lectures and labs. Although around a third of students stopped using Code—or only used it sporadically—after the switch, around two-thirds of students continued to use it regularly. This pattern is relatively consistent throughout Weeks 5–8 in the same way that the long synchronized blocks are present throughout Weeks 2–4. It is worth noting that interaction in the blended scenario was a predictor of interaction in the distance learning scenario ( $r = 0.577$ ,  $p < 0.001$ ), as shown in Figure 4.8.

Given the difference in usage before and after the switch to distance learning, we also wanted to explore the relationship between Code and other digital tools used for distance education. Specifically, we considered the link between Code and the video recordings that replaced the live lectures at the start of Week 5. There was a strong positive relationship between the code interaction metric and the time spent watching videos metric, as shown in Figure 4.9 ( $r = 0.444$ ,  $n = 67$ ,  $p < 0.001$ ). In other words, the more students used Code, the more time they spent watching videos and vice versa. This might be due to the fact that these resources were meant to be used in parallel to simulate the in-class experience.

## 4.5 Discussion and Implications

The results of our evaluation give rise to a number of discussion points concerning the different aspects of our research question. First, in terms of inclusion, our results show that there are no significant differences in usage metrics related to either digital literacy or gender. From a learning design perspective, this shows that Code did not involuntarily discriminate against gender or programming skills. This is particularly important since there can be strong stereotype threats that can hinder learning in computer science, where female students are still widely underrepresented (Starr, 2018). At the University of Neuchâtel, there are typically around 50% of female students registered in the business curriculum and thus present in all mandatory courses, such as our information technology course. Nonetheless, this percentage drops below 20% in the elective programming course. It is therefore imperative that the tools employed in introductory courses do not discourage students from continuing studies that reinforce computational thinking skills.

Second, while the usage of Code and the integrated computational notebook was synchronized across students and heavily concentrated on lecture and lab days during the blended learning scenario, the distance learning scenario introduced a sharp departure from that pattern of use. Access to the learning activities became spread over the first two to three days after the lecture was posted, with some students changing their study habits and viewing the material during the evening, as was also highlighted by Chu et al. (2018). Similarly, the usage of Code became scattered throughout the week. Nevertheless, it is important to highlight that around two-thirds of the students continued to use Code regularly throughout the distance learning period. This could signal that Code and the integrated computational notebooks could be

successfully deployed in both types of learning scenarios. Furthermore, the usage of Code during the blended scenario predicted its usage in the distance scenario. This could help educators identify students who might find the switch more challenging and provide them with adequate support, as suggested by Van Goidsenhoven et al. (2020).

Third, the results from our surveys indicate that students were generally positive regarding the usability and pertinence of both Code and the computational notebook learning activities it supports. Moreover, students particularly appreciated the browser-based experience, with no required installation or setup. This serves as a crucial insight into the possible improvements that could be done to the current computational notebook ecosystems in order to lower the barrier to entry for non-STEM students.

These findings have practical and theoretical implications for the work conducted in the rest of this thesis and serve to inform the design of the code review notebook. First, given that the shift to distance learning implied that the usage of Code became scattered throughout the week, the use of educational chatbots could alleviate the need for educators to be readily available to support learners when completing exercises outside normal office hours. Second, students' positive rating of the integrated computational notebook informed our decision to explore iterations of this format in subsequent studies, namely the design of the code review notebook to simulate social coding platforms in educational contexts. We will introduce code review notebooks in the following chapter. Finally—as will be presented in Chapter 9—we used integrated computational notebooks to embed and evaluate educational chatbots following rule-based scripts.

### 4.6 Conclusion

To conclude, our study provides a snapshot of how digital tools designed for blended learning can allow both educators and learners to adapt to an unforeseen change in pedagogical scenarios. Although our evaluation is by no means conclusive, the fact that usage of Code was not correlated with prior programming skills and that students found it easy to use is a good sign that it will be positively received in introductory programming courses. In future work, we aim to study how students transition from our code app to more advanced technologies, such as Jupyter Notebooks or integrated development environments. Furthermore, we aim to refine our digital tools following the feedback received and continue to investigate how they can enable computational thinking courses.

## 5 Code Review Notebooks

INSPIRED by the popularity of software development bots on social coding platforms and the reported pedagogical value of peer code review exercises, we selected the code review process as an appropriate task to support with educational chatbots. Before integrating chatbots, to ensure that the activities used in our empirical evaluations were relevant to learners, we used the Graasp ADK to implement and test a baseline learning scenario, addressing both the technological and pedagogical dimensions required to support the code review process within an LXP. As a matter of fact, integrating code reviews into educational contexts is particularly challenging due to the complexity of both the process and of popular code review tools. We proposed to address this challenge by designing *Code Review*, an app aimed at teaching the code review process directly within existing digital learning platforms. Using Code Review, educators can scaffold pedagogical scenarios that introduce the code review process to learners through code snippets, following a template resembling integrated computational notebooks. We refer to this template as the *code review notebook*. Through a case study comprising an online lesson on code quality standards, which was completed by 25 university students, we evaluated the effects of using a code review notebook on four aspects of the learning experience: (i) usability, (ii) learning gains, (iii) self-reflection, and (iv) feedback. Our mixed-method analysis suggests that the code review notebook provides a positive user experience, encourages learners to reflect on their learning process, and can result in short-term learning gains.

By harnessing the Graasp ADK to create the technopedagogical scaffolding to support code review notebooks, this chapter partially addresses **RQ2**:

How can we scaffold pedagogical scenarios suitable for educational chatbots?

The content of this chapter was partially presented in the following publications:

1. Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Toward Code Review Notebooks. *2022 International Conference on Advanced Learning Technologies (ICALT)*, 209–211. <https://doi.org/10.1109/ICALT55010.2022.00068>
2. Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2023). Integrating Code Reviews into Online Lessons to Support Software Engineering Education. In M. E. Auer, W. Pachatz, & T. Rüütman (Eds.), *Learning in the Age of Digital and Green Transition* (pp. 815–826). Springer. [https://doi.org/10.1007/978-3-031-26190-9\\_84](https://doi.org/10.1007/978-3-031-26190-9_84)



## 5.1 Introduction

Code reviews, the process by which code written by one programmer is cross-checked by another programmer for potential bugs or issues, are standard practice in software engineering. Most studies addressing the use of code reviews in education focus on *peer* code reviews, in which learners complete a programming assignment and review each other's submissions (Indriasari et al., 2020). Nevertheless, as discussed in the introduction to Part II, one of the challenges of using peer code review in education is the fact that learners often lack the ability to perform code reviews (Indriasari et al., 2020). Moreover, teaching the code review process in an educational setting can be challenging, given the complexity associated with tools supporting code reviews. Social coding platforms, for example, have steep learning curves that can be overwhelming for entry-level students (Zagalsky et al., 2015).

To address these challenges, we propose *Code Review*, an app focused on introducing learners to the code review process. Code Review integrates with LXPs to support lessons in formats similar to the integrated computational notebook presented in the previous chapter. In this chapter, we present the design and evaluation of both the Code Review app and its use inside an online lesson following a format resembling an integrated computational notebook. This format constitutes the fourth contribution proposed in this thesis, namely the *code review notebook* to simulate social coding platforms in educational contexts (C4).

This chapter is structured as follows. We start in Section 5.2 by presenting the design of the Code Review app, as well as the code review notebook template. In Section 5.3 we present our evaluation of our design through a case study with 25 university students. We present our results in Section 5.4 and discuss their implications in Section 5.5. Finally, we conclude in Section 5.6.

## 5.2 Design

Our aim was to design an application that incorporates code review features directly within a digital education platform. To guide our design, we take inspiration from the integrated computational notebook presented in the previous chapter. Our approach was to first design Code Review as a lightweight, embeddable app focused on showcasing the code review process using short snippets of code.

Following this approach, Code Review was designed to simulate the code review process used in software development (Wiegers, 2002), especially as it takes place on social coding platforms. The core of this process consists of a user submitting code for review and another user reviewing this submission by adding comments to specific lines of code. As a starting point, the two main features that we proposed to implement in the app were (i) supporting code snippets and (ii) allowing users to comment on specific lines within the code snippet. This differs from traditional computational notebooks in one key dimension. That is, the code presented is static, since its purpose is not to be modified or executed, but rather to be

commented on by users to start a discussion about possible improvements. In this section, we outline the use cases that we intended to support with Code Review and describe the interfaces implemented in the app to enable these use cases. We then present the design of the code review notebook template, including its structure, the pedagogical patterns it supports, and a first implementation on Graasp.

### 5.2.1 Use Cases

We designed Code Review to support two different use cases, which were in turn based on four contexts in which computational notebooks have been proposed as useful tools for education: (i) lectures, (ii) flipped classroom settings, (iii) homework, and (iv) exams (O'Hara et al., 2015).

The first use case is for *explanatory* purposes, which corresponds to the use of computational notebooks for lectures and flipped classroom settings. Educators should be able to use Code Review to demonstrate the code review process. This could be done by configuring Code Review so that it displays a code snippet with aspects that the educator wants to illustrate. These aspects can then be explained by the educator, who can (i) annotate the code snippet with comments before sharing Code Review with learners—providing learners with a complete example of a code review—or (ii) create those comments during a lecture, showing learners how to complete the code review live.

The second use case aims to provide learners with *practical* experience reviewing code and corresponds to the use of computational notebooks for homework and exams. Educators should be able to configure the application with code snippets containing elements that learners are then required to identify. If presented as an unmarked exercise, this configuration is a way to provide learners with practical experience. The same configuration can also be presented as a quiz or assessment to evaluate how well learners understand the review process and the topic at hand.

### 5.2.2 Interfaces

We built Code Review using the Graasp ADK presented in Chapter 2. In this section, we describe the application's interfaces, including the features implemented to support our use cases.

#### Educator Interface

We provide educators with an interface consisting of four main modes. The *Settings Mode* allows educators to configure Code Review. As shown in Figure 5.1, educators can input a code snippet in one of the four supported programming languages (JavaScript, Python, Java, and MATLAB). The *Annotation Mode* provides educators with a preview of how learners will view the code snippet. Educators can then seed the snippet with comments that are visible

to all learners. This is the core feature needed to support the explanatory use case, as it can help educators illustrate the code review process by example. To monitor overall learner performance, educators can access the *Summary Mode* (see Figure 5.2), which features a table that provides an overview of how learners have engaged with the code snippet. Finally, the *Feedback Mode* allows educators to view how individual learners have annotated the code snippet. Educators can assess learner responses and reply to them with feedback.

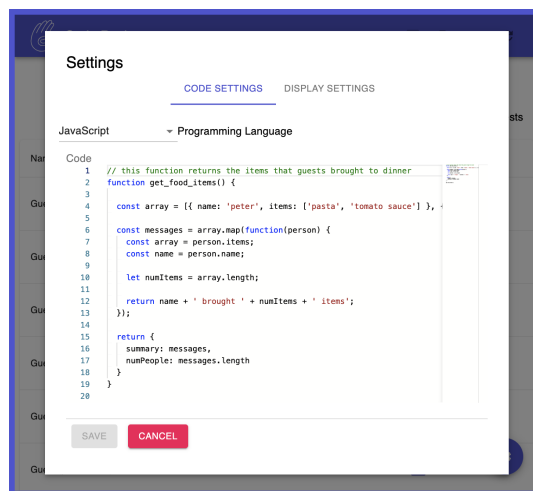


Figure 5.1: Educators can configure Code Review to display syntax-highlighted code in four different programming languages.

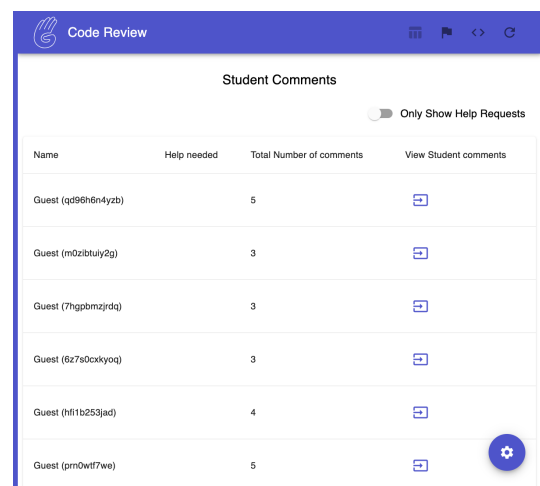


Figure 5.2: The educator interface of the app provides an overview of how learners have interacted with the code snippet.

## Learner Interface

Once the educator has configured Code Review, learners can access it via a dedicated interface consisting of the *Review Mode*. This mode presents learners with the static code snippet as configured by the educator, including any comments that the educator might have seeded within the snippet. Line numbers precede each line of code and syntax is highlighted based on the programming language used. Learners can annotate the code using comments, which are associated with a specific line of code. As illustrated in Figure 5.3, adding comments is done by clicking the “+” icon, which opens an editor that supports the Markdown format for styling text (Gruber et al., 2004). The editor has two modes: (i) an *Edit Mode* where users write and format the content of their comments and (ii) a *Preview Mode* that shows how the comment will be displayed once submitted. Once a comment is submitted, it is displayed under the line it refers to. Providing learners with a means to annotate the code allows us to support the practical use case, as it allows learners to get hands-on experience performing code reviews. Figure 5.4 provides an overview of how comments are displayed.

Our application also supports interactions between learners and educators. Users can reply to comments by clicking the arrow-shaped icon in the top right corner of a comment. Learner

## Chapter 5. Code Review Notebooks

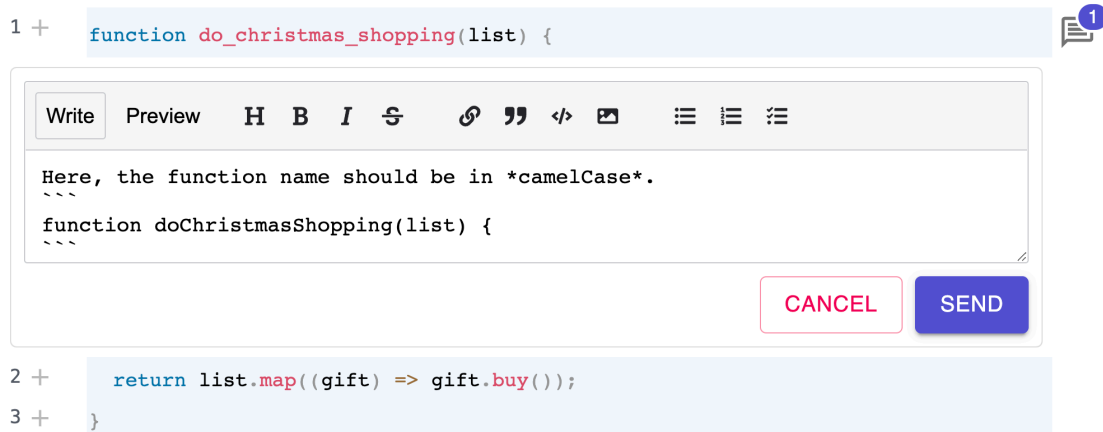


Figure 5.3: Code Review includes a Markdown editor that allows both educators and learners to annotate the code snippet with rich text and images.

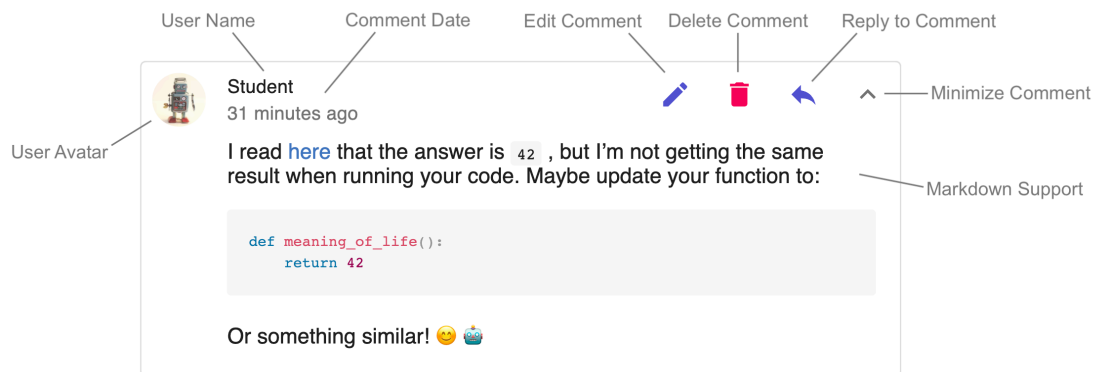


Figure 5.4: Comments can be edited, deleted, replied to, or hidden.

comments are private by default, meaning that they can only be seen by the learner and the educator. However, to support collaboration, educators can make all learner comments visible to all other learners. Learners can then reply to each other and engage in the collaborative review of a code snippet.

### 5.2.3 Code Review Notebooks

The design of the code review notebook aims to guide educators in creating technopedagogical scenarios for teaching various subjects related to programming education through the use of the code review process. Inspired by the integrated computational notebook, the code review notebook is designed to be integrated with a digital learning platform, such as an LXP. The notebook then harnesses features provided by an app that supports the code review process (e.g., Code Review) to explain the subject in question and gauge learners' grasp of the subject,

all the while providing learners with practical code review experience.

### Structure

The three key elements defining a code review notebook are the following:

1. Using code snippets to present a topic in programming education.
2. Scaffolding these code snippets with multimedia content to provide motivation, explanations, and other supporting material.
3. Emphasizing active learning by inviting learners to inspect the code and annotate it with comments with respect to the particular topic in question.

As long as these key elements are followed, the actual structure of a code review notebook can be very flexible. However, to guide educators in designing these notebooks, we define a basic structure as follows. Note that we refer to the parts or steps of the code review notebook as its *phases*, which are meant to be completed sequentially.

1. An initial phase comprising a code review exercise aimed at gauging learners' grasp of the topic in question before the lesson.
2. One or more introductory phases whereby the topic is motivated using multimedia content.
3. One or more phases presenting the topic through examples featuring the code review process.
4. A phase—similar to the initial phase—comprising an exercise to gauge learners' grasp of the topic after the lesson.
5. A concluding phase where learners can also reflect on their performance in the exercise.

### Pedagogical Patterns

Code review notebooks can be designed to follow several pedagogical patterns that are often used in computer science courses. A few relevant examples from those proposed by Bergin (2000) include (i) *Lay of the Land*, where learners are asked to examine a large amount of code in order to understand the complexity of a topic they are studying, (ii) *Fixer Upper*, where learners are presented with code that contains certain flaws that they have to identify and fix, and (iii) *Larger than Life*, where learners examine and possibly suggest modifications to code that they are not yet able to write on their own. The focus these pedagogical patterns place on examining code, identifying some aspect of that code (e.g., complexity, flaw, improvement),

and reporting these aspects align closely with the code review process. Code review notebooks can serve to provide the technopedagogical scaffolding required to facilitate the creation of learning activities following these patterns.

### Implementation

To prototype our proposed code review notebook template, we integrated Code Review with Graasp. As explained in Chapter 3, Graasp exposes an API supporting LA, which we connected Code Review to in order to (i) show the respective interface to educators and learners, (ii) configure the code snippet and settings to be used by the application, (iii) save the educator and user comments, and (iv) record LA when learners interacted with the application. Graasp also supports the structure proposed for code review notebooks. Using Graasp, educators can use Code Review to scaffold a pedagogical scenario comprising multiple phases that a learner can explore. Each phase can include text, images, and other media that serve to support the code review activity. Some phases can provide context (e.g., a phase introducing the lesson), while others can feature Code Review for its explanatory use case (e.g., a phase showing examples alongside explanations) or for its practical use case (e.g., an exercise). Once educators have set up the code review notebook, they can share it with learners via a link. Through this link, learners access a dedicated interface that presents them with the notebook as configured by the educator. An example code review notebook on Graasp is depicted in Figure 5.5.

## 5.3 Methodology

To evaluate Code Review and the code review notebook, we conducted an online study comprising a within-subjects experiment that took place on Graasp. The purpose of our evaluation was to address one main research question, which is an extension of **RQ2**:

- What are the effects of using a code review notebook on students' experiences learning to identify code quality issues in JavaScript? (**RQ2b**)

To frame our evaluation, we focused on four aspects of the learning experience: (i) usability, (ii) short-term learning gains achieved, (iii) self-reflection on the learning experience, and (iv) feedback. We evaluated our code review notebook through a case study focusing on these four aspects. In this section, we present the methodology used for our evaluation.

### 5.3.1 Scenario

To perform our evaluation within an educational setting, the context for our case study was an asynchronous online lesson on code linting. The process of linting code dates back to the 1970s and involves the use of static analysis tools designed to detect issues in software (Johnson, 1978). Linting tools are often used in industry to enforce code quality standards and help

**Getting Started**

Let's start with a quick exercise. Imagine that this bit of code has been sent to you for review. What would you tell the programmer that sent it to you? Add your comments in the respective lines **by clicking the corresponding + icons**.

```

1 + // this function returns the items that guests brought to dinner
2 + function get_food_items() {
3 +
4 +     const array = [{ name: 'peter', items: ['pasta', 'tomato sauce'] }, { name: 'paul',
5 +       items: ['salad', 'cheese'] }, { name: 'mary', items: ['wine', 'dessert', 'coffee'] }];
6 +
7 +     const messages = array.map(function(person) {
8 +       const array = person.items;
9 +       const name = person.name;
10 +
11 +       let numItems = array.length;
12 +
13 +       return name + ' brought ' + numItems + ' items';
14 +     });
15 +
16 +     return {
17 +       summary: messages,
18 +       numPeople: messages.length
19 +     }
20 +   }
21 +   get_food_items();

```

Figure 5.5: In this example code review notebook, the lesson consists of seven phases that can be explored using a vertical navigation bar. This figure highlights the *Getting Started* phase, which uses Code Review to present students with an initial exercise.

automate code reviews. Therefore, it is pertinent to teach the concept of code linting using a code review notebook, especially since the code review process has also been proposed as a pedagogical tool for introducing students to code quality standards (X. Li et al., 2005). In our case, we specifically introduced students to ESLint (Zakas, 2013), a prominent tool specifically designed to find linting errors in JavaScript (Tómasdóttir et al., 2020), as well as the Airbnb JavaScript Style Guide (Airbnb, 2012), a popular configuration for ESLint.

The lesson was structured into seven phases that the students were supposed to navigate in order. As depicted in Figure 5.5, the first phase (*Getting Started*) featured the practical use case of the Code Review app. Following the *Fixer Upper* pedagogical pattern (Bergin, 2000), students were presented with an exercise comprising a code snippet and asked to annotate it with any potential issues they could identify. This initial exercise served as a pre-test to gauge the student's initial knowledge of code reviews, linting, and JavaScript best practices. In the second phase (*Introduction*), we formally introduced the lesson with a short text regarding linting. The third phase (*ESLint*) presented ESLint and the Airbnb JavaScript Style Guide. The fourth (*Styling*) and fifth (*Best Practices*) phases made up the core of the lesson and featured the explanatory use case of the Code Review app. In these phases, we used the code quality issues seeded into the pre-test as examples and walked the student through them using (i) an explanation based on the ESLint documentation and Airbnb JavaScript Style Guide, (ii) a

code snippet highlighting the issue, and (iii) a code snippet illustrating a possible solution. Figure 5.6 depicts a sample explanation block.

### String Templates

In ES6, we can use template literals instead of string concatenation. Template strings give you a readable, concise syntax with proper newlines and string interpolation features.

In the `return` statement, we could make use of a string template.

```
1 + return name + ' brought ' + numItems + ' items';
```

Solution:

```
1 + return `${name} brought ${numItems} items`;
```

Figure 5.6: Code quality issues were explained using text accompanied by a code snippet highlighting the issue and another illustrating a possible solution.

In a sixth phase (*Exercise*), we presented an exercise that served as a post-test to gauge students' learning gains. The exercise consisted of a snippet containing code quality issues, once again harnessing Code Review for its practical use. A seventh phase (*Conclusion*) served as the conclusion of the lesson, exploiting the explanatory use case of Code Review to present the answers to the post-test.

### 5.3.2 Participants

We recruited 27 students pursuing degrees in technical subjects via our internal networks at the École Polytechnique Fédérale de Lausanne and the School of Engineering and Architecture of Fribourg in Switzerland. Students were informed that this was an optional, ungraded exercise, and were asked to share the lesson with their peers. Two students did not complete the post-test and were therefore excluded from our analysis. A total of 22 students—9 female and 13 male—completed an optional demographics questionnaire. Of these students, 17 were currently completing a master's degree, while 5 were enrolled in a bachelor's program. Participants were also asked to report on a scale of 1 to 5—with 1 being *Beginner* and 5 being *Expert*—their programming experience both overall and specifically using JavaScript. Responses to this question show that while students' programming experience was above average ( $\bar{x} = 3.18$ , Mode = 4), they were mostly uninitiated in JavaScript ( $\bar{x} = 1.95$ , Mode = 1).

### 5.3.3 Procedure

Our study took place in January 2022 and was carried out remotely. Students were sent a link to the code review notebook described in Section 5.3.1 and were instructed to take a total of 30 minutes to complete the evaluation. Students accessed the online lesson anonymously and completed an initial exercise that served as a pre-test before going through the code review notebook and then taking a post-test. Upon completion of the online lesson, students were



directed to an online questionnaire that included an optional demographics section.

### 5.3.4 Instruments

To address our research question, we used the following instruments. First, usability was measured using two standard instruments. For Code Review, we used the short version of the User Experience Questionnaire (UEQ-S) (Schrepp et al., 2017). The UEQ-S is an eight-item questionnaire measuring two meta-dimensions of the user experience: (i) *pragmatic quality* and (ii) *hedonic quality*. Pragmatic quality captures a product's *efficiency*, *perspicuity*, and *dependability*, while hedonic quality captures its *stimulation* and *originality*. For the usability of the code review notebook template, we used the SUS (Brooke, 1996). Second, learning gains were calculated as the difference between a student's post- and pre-test scores, with a possible range from  $-100\%$  to  $100\%$ , inclusive. That is, for a given student  $l$ , their learning gain  $g_l$  was calculated as the difference between their score on the post-test  $w_{l,\text{post}}$  and their score on the pre-test  $w_{l,\text{pre}}$ . We refer to this metric as the *learning gain* ( $g$ ).

$$g_l = w_{l,\text{post}} - w_{l,\text{pre}}$$

The third aspect—self-reflection—was addressed via a question asking students to provide feedback on their performance in short answer form: *Did you manage to find all of these [issues]? If not, which ones did you miss? Did you find any of them particularly tricky/helpful?* Finally, feedback comprised two instruments. On the one hand, an open-ended question asking students to provide their thoughts on the lessons, as well as any comments or suggestions: *What did you think about this lesson? Any comments, suggestions, or feedback?* On the other hand, we asked students to rate 10 features that could be added to Code Review on a scale of one (not interested) to five (very interested). Features were selected based on functionalities available on computational notebooks (e.g., *Executing Code*), social coding platforms (e.g., *Bots*), and intelligent tutoring systems (e.g., *Hints*). The 10 features are listed below.

1. *Automated Feedback*: Provide information about code quality.
2. *Peer Code Review*: Comment and edit other students' code.
3. *Bots*: Interact with bots within the application to get feedback.
4. *Code Editing*: Edit, save, and compare code versions.
5. *Hints*: Provide hints without giving away the answer.
6. *Heat Map*: Show which lines received comments from others.
7. *Executing Code*: Execute code within the browser.
8. *Fill In The Blanks*: Finish an incomplete code snippet.

9. *Dashboard*: Visualize one's activity against that of one's peers.
10. *Overview*: Summarize activity across all snippets in a lesson.

### 5.3.5 Data Analysis

The dataset used for our analysis was extracted from Graasp using the LA pipeline presented in Chapter 3. To analyze our data, we followed a mixed-method approach. Quantitative data were analyzed using descriptive and inferential statistics. Specifically, we report the sample mean ( $\bar{x}$ ), median ( $\tilde{x}$ ), and standard deviation ( $s_x$ ), occasionally including the minimum ( $x_{\min}$ ) and maximum ( $x_{\max}$ ). We also performed a dependent  $t$ -test for paired samples to compare the distributions of pre-test and post-test scores. Qualitative data were analyzed using line-by-line data coding (Charmaz, 2006). We also performed sentiment analysis on the self-reflection and feedback comments. For this analysis, we used VADER (Hutto et al., 2014)—a sentiment analysis model trained on social media data—to assign a sentiment score ranging from  $-1$  to  $+1$  to each comment. Responses from the UEQ-S were further processed using its standard data analysis tool, which compares results to benchmark data (Schrepp et al., 2017).

## 5.4 Results

In this section, we outline our results following the four aspects of our study.

### 5.4.1 Usability

A total of 23 students completed the questionnaire rating the usability of both Code Review and the code review notebook template. Code Review—evaluated using the UEQ-S—received a mean overall score of 1.29 ( $\bar{x} = 1.25$ ,  $s_x = 0.81$ ), which is considered *positive* and *above average* (25% of results better, 50% of results worse) when compared to the benchmark. When considering pragmatic and hedonic qualities separately, Code Review achieved mean scores of  $\bar{x} = 1.76$  ( $\tilde{x} = 2.00$ ,  $s_x = 0.91$ ) and  $\bar{x} = 0.82$  ( $\tilde{x} = 0.75$ ,  $s_x = 1.01$ ), respectively. While both of these results are considered *positive*, the pragmatic score corresponds to an *excellent* result (top 10%) when compared to the benchmark, while the hedonic score is *below average* (50% of results better, 25% of results worse). The code review notebook template was evaluated using the SUS and received a mean score of  $\bar{x} = 84.35$  ( $\tilde{x} = 85$ ,  $s_x = 12.46$ ), a usability score that can be described as *excellent* (Bangor, Kortum, et al., 2009).

### 5.4.2 Learning Gains

In the pre-test, students detected 14.5% of issues on average ( $\bar{x} = 14.5\%$ ,  $\tilde{x} = 9.09\%$ ,  $s_x = 13.1\%$ ), while in the post-test, the mean increased to  $\bar{x} = 48.6\%$  ( $\tilde{x} = 50.0\%$ ,  $s_x = 23.1\%$ ). The differences

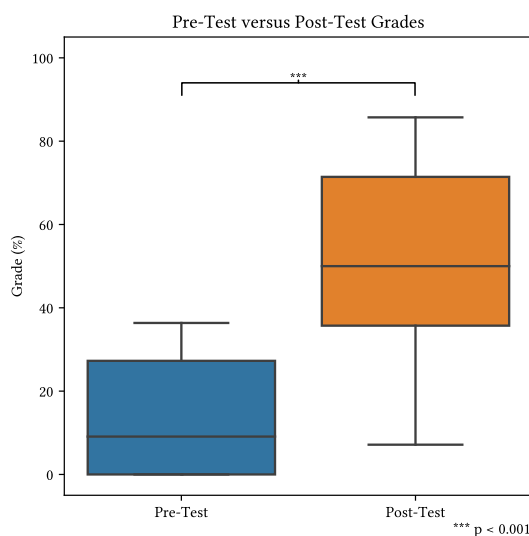


Figure 5.7: Boxplots showing the differences between the pre- and post-test scores.

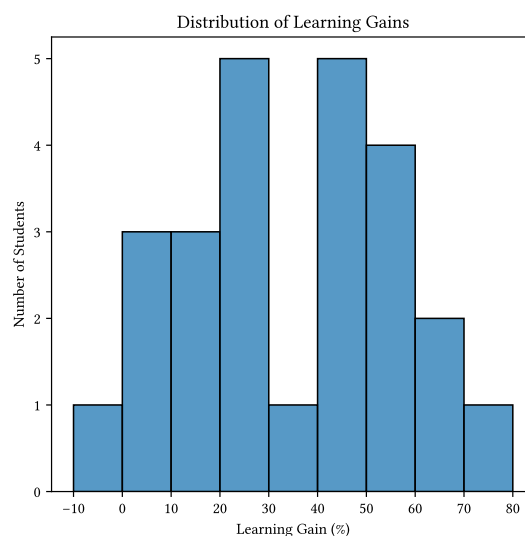


Figure 5.8: Histogram depicting the distribution of learning gains achieved. Students are grouped by percentage achieved in steps of 10%.

between the pre- and post-test resulted in a mean learning gain of  $\bar{x} = 34.0\%$  ( $\tilde{x} = 35.7\%$ ,  $s_x = 20.0\%$ ,  $x_{\min} = -1.95\%$ ,  $x_{\max} = 71.4\%$ ), with all students except one achieving positive learning gains. To assess the distribution of learning gains (and therefore the distribution of the differences between the pre- and post-test), we performed a Shapiro-Wilk test, which did not show evidence of nonnormality ( $W = 0.967$ ,  $p = 0.578$ ). Indeed, Figure 5.8 depicts a histogram of the learning gains achieved, grouping students in incremental steps of 10%. This histogram visually suggests a normal distribution. A dependent  $t$ -test for paired samples showed that there is a significant difference in the distribution of the pre- and post-test scores ( $t(24) = 8.51$ ,  $p < 0.001$ ). Figure 5.7 illustrates the differences between both distributions.

### 5.4.3 Self-Reflection

A total of 23 students provided a short open-ended answer to a question asking them to reflect on their performance in the post-test. Comments ranged from 34 to 553 characters in length ( $\bar{x} = 189.1$ ,  $\tilde{x} = 151.0$ ,  $s_x = 131.7$ ). Of the 23 students who provided an answer to the question on self-reflection, 2 students did so without identifying the specific issues they found problematic, while 17 students specifically reported issues they had missed. For students who mentioned specific issues, they reported a mean of  $\bar{x} = 59.6\%$  ( $\tilde{x} = 55.8\%$ ,  $s_x = 26.5\%$ ,  $x_{\min} = 11.1\%$ ,  $x_{\max} = 100.0\%$ ) of issues they had missed. Our sentiment analysis of these comments revealed that they had a negative sentiment on average ( $\bar{x} = -0.278$ ,  $\tilde{x} = -0.340$ ,  $s_x = 0.546$ ). Figure 5.9 shows the distribution of the sentiment scores and Figure 5.10 a plot of these sentiment scores against their corresponding student's learning gain. A simple linear regression was

performed to test whether learning gains were an adequate predictor of sentiment scores ( $R^2 = 0.115$ ,  $F(1, 21) = 2.733$ ,  $p = 0.113$ ), but the results were not significant.

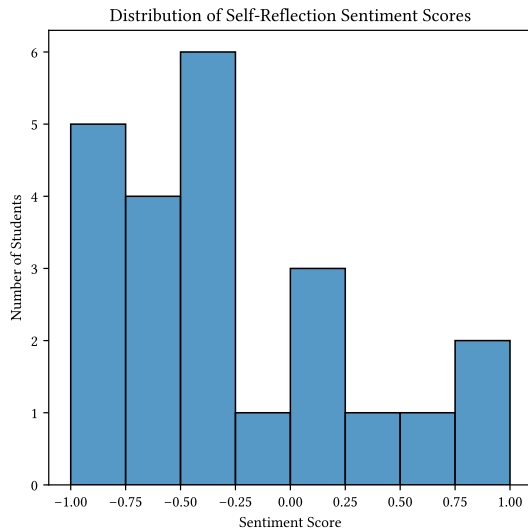


Figure 5.9: While the distribution of students' self-reflection sentiment scores was wide, it shows a tendency towards negative scores.

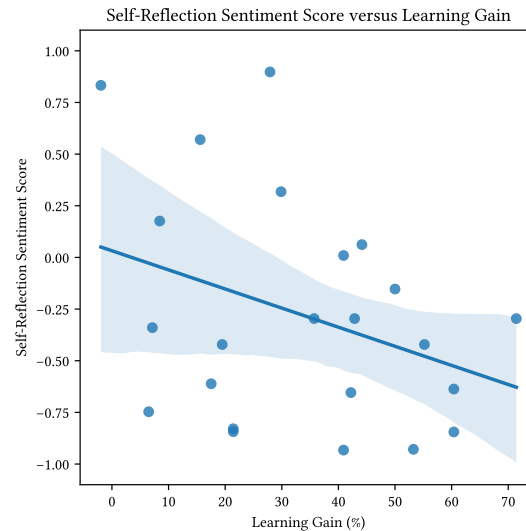


Figure 5.10: There is a negative relationship between learning gains and self-reflection sentiment scores, but learning gains did not significantly predict sentiment scores.

### 5.4.4 Feedback

In addition to self-reflection, 22 students provided feedback in the form of responses answering an open-ended question about their overall satisfaction with the lesson. Comments ranged in length from 4 to 924 characters ( $\bar{x} = 177.3$ ,  $\tilde{x} = 132.5$ ,  $s_x = 188.4$ ) and the sentiment analysis performed on these comments yielded a positive mean sentiment of  $\bar{x} = 0.556$  ( $\tilde{x} = 0.668$ ,  $s_x = 0.374$ ). Figure 5.11 shows the distribution of these sentiment scores and Figure 5.12 the relationship between sentiment scores and learning gains. Once again, we performed a simple linear regression to test whether learning gains were an adequate predictor of sentiment scores, resulting in results that were not significant ( $R^2 = 0.165$ ,  $F(1, 19) = 3.767$ ,  $p = 0.0673$ ). Finally, using line-by-line coding, we identified prominent themes that were present in the feedback. On the one hand, 15 students highlighted the positive aspects of the lesson, describing it as “clear”, “good”, and “useful”. This is best summarized by one particular student, who described the lesson as follows: “[The lesson is] very pedagogical, [it] gives a good first overview without being too dense or discouraging, [and] makes you want to learn more by yourself!”<sup>1</sup>. On the other hand, three students pointed out their lack of experience in the chosen programming language as a limitation, while three different students found the lesson difficult.

<sup>1</sup>“Très pédagogique, donne un bon premier aperçu sans pour autant être trop dense et décourageant, donne envie d’aller apprendre plus par soi-même !” (Translated from French by the authors.)

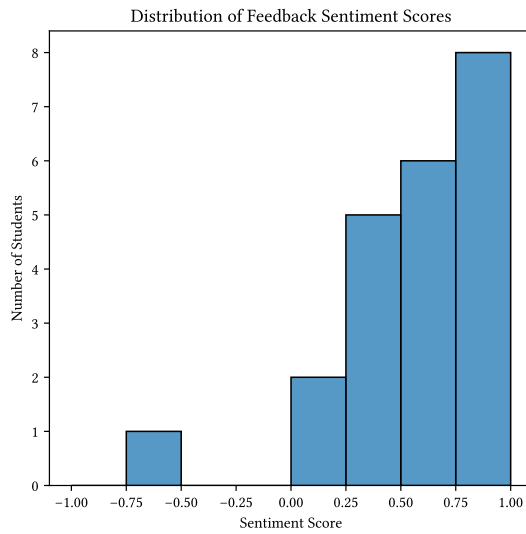


Figure 5.11: The distribution of sentiment scores for the feedback provided was predominantly positive.

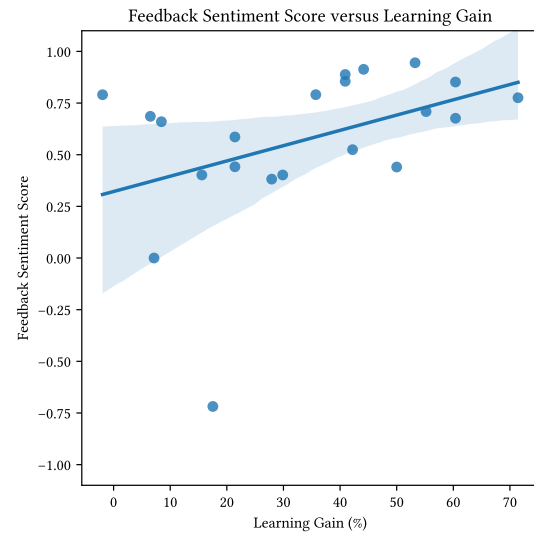


Figure 5.12: While there is a slightly positive relationship between learning gains and feedback sentiment scores, learning gains did not significantly predict sentiment scores.

The three most requested features were *Code Editing* ( $\bar{x} = 4.22, \tilde{x} = 4, s_x = 1.17$ ), *Automated Feedback* ( $\bar{x} = 4.04, \tilde{x} = 4, s_x = 1.19$ ), and *Hints* ( $\bar{x} = 3.87, \tilde{x} = 5, s_x = 1.79$ ). The three least requested features were *Dashboard* ( $\bar{x} = 2.52, \tilde{x} = 2, s_x = 1.73$ ), *Bots* ( $\bar{x} = 2.65, \tilde{x} = 3, s_x = 1.64$ ), and *Heat Map* ( $\bar{x} = 2.91, \tilde{x} = 3, s_x = 1.56$ ). Results for the full set of features requested are shown in Figure 5.13.

## 5.5 Discussion and Implications

The design of our code review application—as well as its implementation within a code review notebook and its use to teach software engineering best practices to undergraduate students—is a first validation that code review functionalities offered by social coding platforms can be simulated within digital learning environments. Students who participated in the experiment were able to annotate the different code snippets as if they were engaging in an actual code review process. Furthermore, the educator interface served to evaluate students' responses and gather part of the data used for our study.

This validation is a strong first step toward lowering the barrier to entry for novice students to exploit functionalities offered by social coding platforms. Students are not required to pre-install software or create accounts, and comment visibility is limited to the concerned parties. The code review notebook template also serves educators and researchers looking to better understand how learners behave and learn in scenarios involving social coding interactions. By integrating code review notebooks within an LXP, instructors and researchers can gain

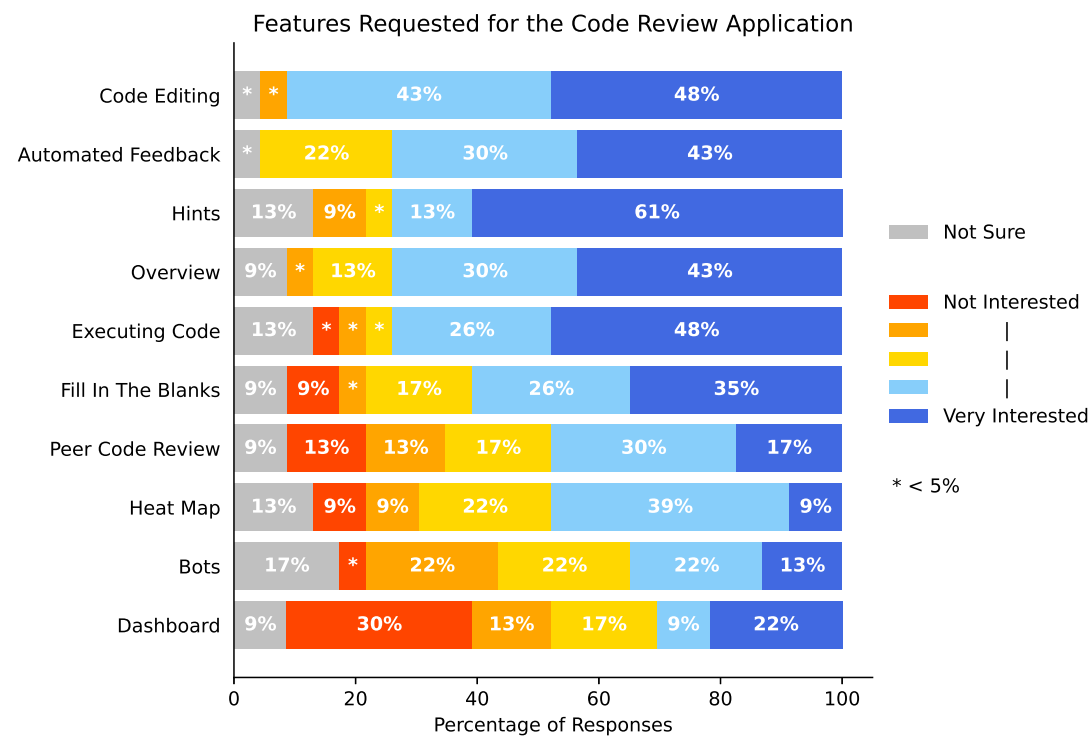


Figure 5.13: Features requested for Code Review ordered by mean score (top to bottom).

insight into learners’ experiences using LA.

Indeed, the results of our study show that the overall usability scores for both Code Review and the code review notebook template were excellent and above average, respectively. These scores suggest that Code Review and the code review notebook template lead to positive user experiences (UXs) within our experimental context. Additionally, Code Review received an excellent score—top 10% when compared to the benchmark—with respect to its pragmatic qualities. That is, users perceive its design as well-suited for a task-oriented UX. Results for the feedback aspect of our study provide key insights into how we can improve Code Review to support functionalities that learners will find useful. The three most requested features (*Code Editing*, *Automated Feedback*, and *Hints*) illustrate how learners perceive Code Review’s potential. *Code Editing*, on the one hand, would enhance Code Review to more closely resemble the development process supported by social coding platforms. *Automated Feedback* and *Hints*, on the other hand, focus on the educational aspect of Code Review, expanding support for more pedagogical scenarios, such as those typically supported by intelligent tutoring systems intended for programming education (Crow et al., 2018). Integrating these features could also make Code Review a more novel and engaging tool, thus improving its usability score with respect to its hedonic qualities, which were rated as below average in the current implementation. Finally, it is important to note that while the *Bots* feature was one of the least requested features for Code Review, it had the highest percentage of *Not Sure*

responses, suggesting that participants might not have been able to envisage how chatbots could be integrated into the lesson.

The results of our study also suggest that using a code review notebook to support an online lesson on software engineering education leads to tangible short-term learning gains. The fact that—on average—students detected 34% more code quality issues in the post-test than in the pre-test indicates that the lesson had a considerable impact on students' understanding of what constitutes good quality JavaScript code and on their ability to detect and annotate issues present in the code snippet. These results suggest that our online lesson format could serve to address one of the issues identified by Indriasari et al. (2020), namely that learners often lack the skills necessary to perform code reviews (Trytten, 2005). Specifically, our lesson format could be used to address issues similar to those experienced by Stalhane et al. (2004), who reported that during a code review exercise, almost all students found only 2–3 out of 20–30 defects (10%), even though they were expected to find around 60%. According to the authors, one of the possible explanations for this gap could be that the “process was unfamiliar and hence daunting” (Stalhane et al., 2004) for learners. Thus, a code review notebook similar to the one used in this study could be exploited to guide learners through the code review process before exposing them to more complicated code review exercises. Our work also sheds light on how to help address some of the concerns teachers have regarding the peer code review process. In a study of two computer science university-level courses, Kubincová and Homola (2017) noted that teachers perceived the feedback provided by students (to other students) as focusing mostly on minor coding issues. By harnessing a code review notebook similar to the one used in our experiment, educators could provide specific examples of both the type of issues that they want learners to look out for, as well as the way they want learners to highlight these issues when they encounter them while reviewing their peers' code.

Our analysis of the self-reflection comments provides evidence that students took the time to reflect on the learning experience. Exercises promoting self-reflection have been shown to be positively correlated with student success in software engineering courses (Pedrosa et al., 2021) and students have reported finding self-reflection helpful (Minocha et al., 2007). Educators could harness answers from these exercises in conjunction with the performance on code reviews to identify learning gaps, and thus personalize teaching strategies targeting students struggling with the concepts at hand. Furthermore, while the sentiment analysis of these self-reflection comments yielded a negative score on average, this result could be due to the way the question was posed to students. More concretely, the fact that we specifically asked students to reflect on the issues they had not been able to identify could have negatively biased the sentiment scores of their responses.

Conversely, the feedback provided was overall positive, indicating that the lesson content and the code review notebook template were predominantly well-received by students. Specific themes and comments that emerged from our qualitative analysis highlight the potential that code review notebooks have to serve as an introduction to complicated or broad topics that students can then go on to explore on their own. As there are a vast number of issues that can

be encountered when performing an open-ended code review exercise, using a code review notebook to motivate the most common ones could be useful for educators looking to prepare students for open-ended peer code review exercises.

Finally, the regression analysis performed shows that learning gains were not a good predictor of either the sentiment score of students' self-reflection comments or students' feedback on the lesson. While this study's sample size is limited, the results of our regression analysis could suggest that actual performance in the code review exercises was independent of motivation to reflect on the learning experience and of the perception of the lesson itself. This makes these types of lessons more adequate as a first exposure to the code review process for novice programmers, who should not feel discouraged if they do not perform as well as expected.

Our results have concrete implications for the work conducted in the rest of this thesis. First, having validated the code review notebook as a useful technopedagogical scenario for teaching programming best practices, we will harness it in our empirical evaluations with educational chatbots. Second, the design and development of Code Review provides us with the context in which the educational chatbots will be interacting with learners. We will use Code Review and various code review notebooks to test the different interaction strategies that constitute one of the core contributions of this thesis (C5). Finally, the features requested for Code Review informed the development of a third app—Code Capsule—which combines Code and Code Review into one app that allows users to edit, execute, and review code.

### 5.6 Conclusion

In this chapter, we presented results from a case study aimed at assessing the impact of code review notebooks on students' learning experiences in the context of an online lesson on detecting code quality issues in JavaScript. Results from our mixed-method analysis provide insight into the potential for code review notebooks to help students learn how to perform code reviews. Our findings are particularly pertinent to practitioners who use or are looking to incorporate code reviews into their courses. By introducing the code review process using an online lesson following the code review notebook template, educators could prepare students to conduct reviews that are useful both for themselves and for the student receiving the feedback. As more educators adopt such approaches, research into how best to scaffold these lessons for different audiences (e.g., age groups, fields of study) could shed more light on the impact our approach could have in domains outside of software engineering education.

Our study has some limitations worth highlighting. First, while the sample size achieved in this study is suitable for an initial within-subjects short-term learning gains analysis, data from more subjects could allow us to consolidate our findings and draw clearer conclusions with respect to how learning gains are related to self-reflection and feedback. Second, our learning activity took place outside of the formal classroom, as an optional exercise. This might have resulted in sample bias, as those students who responded to our request for participation might have been students already motivated enough to take part in an ungraded



exercise. To address this, the experiment should be reproduced in the context of a formal software engineering course, possibly as a mandatory—albeit ungraded—exercise, which would provide us with a possibly less biased cohort of students. Third, this study focuses on one particular use of code review notebooks, which concerns introducing code linting concepts for JavaScript programming. In order to generalize our conclusions, it is necessary to replicate our methodology with code review notebooks tackling other subjects—such as introducing basic programming concepts (e.g., loops, conditional statements, functions)—that could also be applicable to software engineering education. Fourth, while the feedback on the lesson was mostly positive, some negative comments regarding the usability of the interface emerged during our analysis. These comments should be addressed. Furthermore, the current evaluation was limited to students pursuing degrees in technical subjects. To test whether code review notebooks can indeed lower the barrier to entry with respect to the code review functionalities available on social coding platforms, future studies should include nontechnical students following an introductory programming course. Finally, even though our use of Graasp is suitable for a proof of concept, it is important to show how Code Review can be incorporated into other learning platforms to highlight its portability. In future work, we aim to address these limitations and explore the potential to expand the code review notebook to more use cases.



# **Interaction Strategies** **Part III**



---

IN the third part of this thesis, we investigate different strategies that educational chatbots could use in their interactions with learners. Having laid out the technological foundations for integrating chatbots into digital education platforms and the pedagogical scenarios in which we will embed these chatbots, we harnessed the Graasp ADK to build apps aimed at testing these interaction strategies in different settings.

We start in Chapters 6 and 7 by presenting the initial exploratory studies we conducted to better understand how anthropomorphic qualities, such as humor, arise in interactions between learners and chatbots, both in educational and software engineering contexts. More specifically, in Chapter 6 we make a short detour away from software engineering education and into the domain of language learning—an area where the use of chatbots is particularly promising and historically relevant. We present results from a controlled online experiment with 400 participants in which we tested the effect of equipping a chatbot with the ability to convey the perception of humor as a part of a short language learning exercise. In Chapter 7 we continue our detour—this time outside education, but within software engineering—to report on a large-scale observational study of how developers interact with chatbots through GitHub’s emoji reaction interface. Building on our findings from Chapter 6, we focus on the use of the *laugh* reaction and its possible use to convey humor.

We then make our way back to software engineering education and present findings from a series of empirical field studies employing code review notebooks. In Chapter 8, we present results from a study featuring an enhanced version of the ESLint code review notebook presented in Chapter 5. This enhanced version included static comments from chatbots, which we embedded in Code Review. In Chapter 9, we equip Code Review with chatbots following dynamic rule-based scripts and report the results of a week-long pilot and a semester-long study using these chatbots to teach Python programming best practices. Finally, in Chapter 10, we outline a strategy for prompting large language models (LLMs) to power educational chatbots aimed at educational contexts. These LLM-powered chatbots were integrated into a new code review notebook based on the one used in Chapters 5 and 8, which we incorporated into a final field study. In this brief introduction, we highlight related work.

---

## Related Work

The use of chatbots in education dates back to shortly after the appearance of ELIZA—generally considered to be the first chatbot (Weizenbaum, 1966)—with conversational interfaces being incorporated into learning environments as early as the 1970s (Kaplow et al., 1973). Historically, particular emphasis was placed on the potential that interactive systems have for second language learning (L2). This potential was highlighted by Nelson et al. (1976), who stressed the inherent need for language learning programs to take advantage of the interactive capabilities of the computer and proposed strategies to achieve this interactivity. Since then, interactivity has been championed for computer-assisted language learning (CALL) systems under different names, including *intelligent language tutors*, *dialog systems*, and—most recently—*chatbots* (Bibauw et al., 2019).

Early implementations of chatbots for L2, such as *CSIEC* (Jia, 2004), paved the way for research incorporating them into education with a focus on learner autonomy (Abu Shawar et al., 2007b) and open learner models supporting personalization and adaptivity (Kerly et al., 2007). Hayashi (2015) conducted an empirical study to investigate the use of pedagogical conversational agents (PCAs) to understand the effects of social facilitation on a learner's cognitive processes during a web-based learning activity. Tegos et al. (2016) also investigated the use of PCAs in an online explanation activity, with a focus on analyzing the differences arising from varied intervention modalities. In recent years, the rise of instant messaging platforms has motivated both educators and learners to adopt chatbots across all disciplines, with recent surveys identifying 89 educational conversational agents on the Facebook Messenger platform alone (Smutny et al., 2020).

Recent reviews of the literature have also revealed that chatbots are often used to teach computer-related topics (Kuhail et al., 2023; Wollny et al., 2021). Coronado et al. (2018) proposed a personal agent to support students in learning the Java programming language. Their empirical evaluation, which relied on both objective and subjective metrics, showed that integrating social dialog in question answering agents increased user satisfaction and engagement with the system in which the agents were deployed. Mad Daud et al. (2020) also proposed a chatbot for learning Java. By generating different code control structures, the proposed *e-Java* chatbot specifically helped students in learning different ways of coding solutions for the same problem. Custom surveys were used as evaluation metrics to assess the perceived usefulness of the proposed chatbot. Binkis et al. (2021) implemented a rule-based chatbot, which they used in a software engineering course for undergraduate students. A usability assessment of their chatbot using a standard questionnaire resulted in an acceptable usability score. Laiq et al. (2020) also built and evaluated a chatbot aimed at teaching software engineering students. Their chatbot focused on helping students learn how to perform requirements elicitation interviews and achieved promising results. Positive results integrating chatbots into software engineering education were also reported by Winkler, Hobert, et al. (2020), who designed *Sara*—a chatbot to scaffold online video lectures—which they evaluated using introductory Python programming tutorials. Similarly, Ismail et al. (2019) proposed a

---

chatbot to support novice programmers and conducted an evaluation with students, who reported the tool as useful.

The potential for chatbots to support programming education is best summarized by Hobert (2020), who developed *Coding Tutor*—a chatbot to support university students in introductory programming courses—which was also positively rated by students. Referring to Coding Tutor, Hobert indicated that “such a conversational intelligent programming tutor is suited to take over tasks of teaching assistants in times when no human teaching assistant or lecturer is available for help”. Given our observation in Chapter 4 of the shift in students’ patterns of interaction with the learning material as activities moved from blended to remote learning scenarios, this is a particularly relevant feature that educational chatbots can provide.

As we explore the effect that educational chatbots can have on the learning experience, one important dimension to consider is the strategies that these chatbots will follow in their interactions with learners. Chatbots can be powered by rule-based, statistical data-driven, or end-to-end neural dialog systems (McTear, 2021), or using Wizard of Oz techniques whereby they are controlled behind the scenes by a human, unbeknownst to the end user (Dahlbäck et al., 1993). Furthermore, as alluded to in Chapter 1, how chatbots harness different social cues can have a considerable impact on how users perceive these chatbots. Feine, Gnewuch, et al. (2019) proposed a taxonomy of 48 social cues, including verbal (e.g., sentence complexity), visual (e.g., facial expression), auditory (e.g., volume), and invisible (e.g., response time) cues. While Feine, Gnewuch, et al. do not propose *humor* on its own as a social cue, they do propose *joke* and *laughing*, two cues that are strongly related to humor. Inspired by the role humor can play in improving the learning experience (Hellman, 2007; Wanzer et al., 1999), a part of the work presented in the following chapters explores how chatbots can harness humor in their interaction strategies.

Humor specifically serves important social functions in conversation (Graham et al., 1992). If chatbots are expected to act as conversational partners, recognizing and harnessing humor in their interactions with humans should be considered an essential feature of such agents. Indeed, strategies for equipping chatbots with humor have been the subject of a long line of research (Loehr, 1996; Morkes et al., 1999; Nijholt et al., 2017). Often, these strategies are linked to endowing a chatbot with a *personality*, as highlighted by Chaves et al. (2020).

Early attempts to develop chatbots that explicitly incorporated humor were started by Loehr (1996), who developed a system called *Elmo* that used natural language and could embed puns in conversations. Their research also suggested that if a machine did not use humor appropriately, it might irritate users rather than amuse them. Morkes et al. (1999) introduced humor through a conversational agent during a task-oriented interaction and found that the presence of humor enhanced the agent’s likability in the eyes of human participants. Nijholt (2007) discussed several factors to take into account when creating humorous chatbots, presenting a basic architecture that could be adopted for designing humorous anaphora systems. Based on these discussions, Dybala et al. (2009b) provided the first consistent

---

definition of humor-equipped talking agents, referring to them as *humoroids*.

The strategies presented in the following chapters cover first the content (humor, emoji) and second the mechanism (Wizard of Oz, rule-based scripts, large language models) of the chatbot interactions. The goal—as outlined in Chapter 1—is to shed light on how these strategies apply to education (Chapter 6), software engineering (Chapter 7), and more specifically, software engineering education (Chapters 8–10). Indeed, the core set of empirical case studies conducted in this thesis (Chapters 8–10) focus on an application of chatbots to programming education that has not been addressed in the literature, that is, their ability to support code review exercises. In particular, given the widespread use of chatbots on social coding platforms (Wessel, de Souza, et al., 2018), there is an opportunity to study how chatbots could support the code review process within educational contexts, and what effect these chatbots could have on students’ learning experiences. A better understanding of these interactions could help inform the design of not only educational chatbots aimed at software engineering education, but also educational chatbots aimed at other disciplines and chatbot technologies in general.



## 6 Perception of Humor

CHATBOTS have long been advocated for computer-assisted language learning systems to support learners with conversational practice. A particular challenge in such systems is explaining mistakes stemming from ambiguous grammatical constructs. Misplaced modifiers, for instance, do not make sentences ungrammatical, but introduce ambiguity through the misplacement of an adverb or prepositional phrase. In certain cases, the ambiguity gives rise to humor, which can serve to illustrate the mistake itself. We conducted an online experiment with 400 native English speakers to explore the use of a chatbot to harness such humor. In an interaction resembling an advanced grammar exercise, the chatbot presented participants with a phrase containing a misplaced modifier, explained the ambiguity in the phrase, acknowledged (or ignored) the humor that the ambiguity gave rise to, and suggested a correction. Participants then completed a questionnaire, rating the chatbot with respect to ten traits. We performed a quantitative analysis of participants' responses, focusing on the effect that acknowledging the humor had on their perceptions of the chatbot. Our findings showed a statistically significant increase in how participants rated the chatbot's personality, humor, and friendliness when it acknowledged the humor arising from the misplaced modifier. This effect was observed whether the acknowledgment was conveyed using verbal, nonverbal (emoji), or mixed cues. Implications of these findings for the personalization of educational chatbots and the design of educational applications are discussed.

By investigating how learners perceive a chatbot's acknowledgment of humor during an interaction resembling a language learning exercise, this chapter partially addresses **RQ3**:

How can we equip educational chatbots with effective interaction strategies?

The content of this chapter was partially presented in the following publication:

1. Farah, J. C., Sharma, V., Ingram, S., & Gillet, D. (2021). Conveying the Perception of Humor Arising from Ambiguous Grammatical Constructs in Human-Chatbot Interaction. *Proceedings of the 9th International Conference on Human-Agent Interaction (HAI '21)*, 257–262. <https://doi.org/10.1145/3472307.3484677>

### 6.1 Introduction

In this chapter, we take a short detour away from software engineering education to present findings from a study conducted in the form of a lesson resembling a language learning exercise. This study aimed to explore how to integrate humor into short interactions between learners and educational chatbots. The focus on language learning allowed us to target a large online participant base and also served to test the generalizability of our tools. Given that chatbots harness natural language in their interactions, the interaction strategy investigated in this chapter is also pertinent to software engineering education and to other domains. Nevertheless, in this chapter, the pedagogical scenario used in our evaluation was informed by challenges faced by language learning systems. A challenge for these systems, in general, is to provide meaningful feedback for mistakes caused by ambiguous grammatical constructs (Amaral et al., 2011; Morgado da Costa et al., 2016). This problem is particularly relevant for chatbot interfaces due to the relatively unconstrained nature of the interactions they enable (Bibauw et al., 2019). Furthermore, the ambiguity does not always arise from a grammatical mistake *per se*. Sentences with misplaced modifiers, for instance, are not necessarily ungrammatical, but can portray the wrong message due to the awkward placement of an adverb or prepositional phrase (Lester et al., 2018). In some cases, misplaced modifiers can cause unintended humor, as exemplified in:

*My client has discussed your proposal to fill the drainage ditch with his associates.* (Fordyce-Ruff, 2011)

The humor elicited by this phrase is aligned with *incongruity theories* of humor, which postulate that “humor arises from the perception of an incongruity between a set of expectations and what is actually perceived” (Attardo, 2008). In verbal humor, the General Theory of Verbal Humor (GTVH) builds on incongruity theories to propose that a “text is funny if and only if both... (i) the text is compatible, fully or in part, with two distinct scripts; and (ii) the two distinct scripts are opposite” (Ruch, 2008). In the example above, two possible interpretations of this sentence are (i) that the client has discussed the proposal with his associates or (ii) that the proposal entails using the client’s associates to fill the drainage ditch. Humor arises from the incompatibility between (i) and (ii).

If chatbots are expected to act as conversational partners or language tutors, recognizing and handling these ambiguities appropriately is not only important for pedagogical purposes, but also from the perspective of HCI. As discussed in Chapter 1, the Computers Are Social Actors (CASA) framework has been successful in showing how computers are subject to the same social conventions guiding interactions between humans (Nass, Steuer, et al., 1994). Moreover, Moon (2000) showed that this effect becomes even more pronounced when human-like traits such as humor—which serves important social functions in conversation (Graham et al., 1992)—are conveyed by the computer. Following this line of research, chatbots could be designed to recognize the humor in ambiguous grammatical constructs, convey that they have

perceived it, and harness it to explain the mistake in a way that resembles the relationship a learner would have with an educator.

The primary objective of this study was to assess the effect of acknowledging humor on anthropomorphic perceptions. More specifically, we examined whether a chatbot that reacted to humorous misplaced modifiers within an interaction resembling an advanced grammar exercise was rated more positively than one that completed the task without explicit acknowledgment of the humor. To assess this effect, we first designed a simple, configurable interface to host a short interaction between a user and a chatbot. We then used this interface in an online experiment with 400 participants. The experiment evaluated the effect of using verbal and nonverbal (emoji) cues to convey the perception of humor arising from ambiguous grammatical constructs. Given that misplaced modifiers affect both English as a second language (ESL) and native speakers (Edlund et al., 2012; Joubert et al., 2015)—and to minimize the possibility of introducing confounding factors due to misapprehension of the phrases used—participants were all native English speakers, born and currently living in the United States.

This quantitative study provides contributions that are relevant to both research and practice. We build on advances in the field of computational humor (Taylor et al., 2005; West et al., 2019) and the design of chatbots equipped with humor capabilities (Dybala et al., 2009a; Shum et al., 2018), as well as their application to language learning (Bibauw et al., 2019; Coniam, 2014). Our empirical results provide support for the CASA paradigm, are consistent with both early and recent assessments of humor used by task-oriented chatbots (Medhi Thies et al., 2017; Morkes et al., 1999), and extend this line of research to chatbots intended for CALL. We also provide evidence that emoji-only, verbal-only, and mixed messages are all appropriate means of conveying the perception of humor in this type of interaction. Moreover, our results were achieved by harnessing unintended humor arising from grammatical ambiguity, a form currently unexplored in the study of chatbots and humor.

These findings provide a first strategy that could be used to configure how educational chatbots interact with learners, making up a part of the fifth contribution in this thesis (C5). In particular, the approach laid out in this chapter can be used to personalize educational chatbots based on user feedback and to augment current pedagogical scenarios with the aim of incorporating humor into the learning process.

This chapter is structured as follows. We start in Section 6.2 by presenting the design of the chatbot interface and the interaction harnessing humor arising from misplaced modifiers. In Section 6.3 we present the methodology used for our controlled online experiment. We present our results in Section 6.4 and discuss their implications in Section 6.5. Finally, we conclude in Section 6.6.

### 6.2 Design

In this section, we introduce the chatbot used for our experiment. We start by motivating the design of a simple yet interactive interface able to host a configurable exchange between a user and a chatbot. We then outline the preliminary survey conducted to select the misplaced modifiers employed by the chatbot in its interactions.

#### 6.2.1 Chatbot Interface

Our objective for the interaction between the user and the chatbot was to maximize ecological validity without threatening the internal validity of the study. To achieve this, we designed an interface that could simulate an exchange a user might expect to have with a chatbot, but ensuring that only the manipulated variable changed across the different conditions addressed by our experiment. Although this control can be enforced by exposing participants to screenshots of precomposed conversations—as used by Beattie et al. (2020)—an interactive exchange helped us ensure that the duration of the exposure was as consistent as possible and that attention was given to each of the messages presented by the chatbot. We achieved this by presenting the chatbot's messages individually and with short delays in between—as users would expect in computer-mediated communication (CMC) with a human counterpart—and by prompting the user for responses at predefined points in the dialogue. In turn, this allowed us to home in on the effect of small changes in the content of the dialogue while keeping the interactive nature of the conversation.

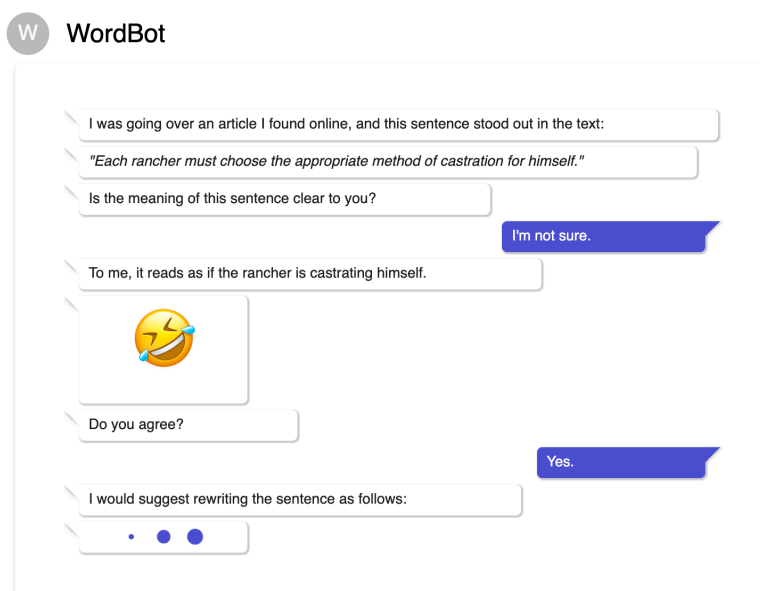


Figure 6.1: Our chatbot interface was configured to resemble an interactive grammar exercise.

We incorporated best practices for the design and implementation of chatbots highlighted by Chaves et al. (2020), Ferman Guerra (2018), and Følstad and Brandtzaeg (2020) that were relevant to our scenario. These included (i) providing the chatbot with an identity, (ii) explicitly stating the chatbot's purpose, (iii) following a conversational script, (iv) keeping messages short, (v) simulating typing proportional to the length of the incoming message, and (vi) including content that the user could find pleasant, evocative, or playful. Since our aim was not to conceal that our chatbot was not human—and to avoid unexpected gender (McDonnell et al., 2019) or ethnic biases—our chatbot was introduced to users under a generic name (*WordBot*) alongside an avatar consisting of a text-based icon featuring the letter *W* on a gray background.

Furthermore, to ensure that the interaction was as consistent as possible even among participants exposed to the same condition, the script followed by the chatbot was not contingent on user input. This required that the wording of the questions posed by the chatbot and the messages that followed be compatible with any response that could be specified by the user. For simplicity, user input was limited to *Yes*, *No*, or *I'm not sure*. To provide these answers, participants were prompted to reply using quick response buttons. This strategy has been used to ensure conversational structure (Chaves et al., 2020) and although users have described such buttons as restrictive (Duijst, 2017), they have also been reported to be appreciated (Jain et al., 2018) and effective in nudging participants in task-oriented interactions (Tallyn et al., 2018).

Finally, to approximate the user experience a learner would have in a CALL system, we embedded the chatbot interface within a web application that resembled an exercise on an LXP. This application contained three screens. The first screen provided the user with instructions for the exercise and a button to start the interaction. The second screen encapsulated the conversation with the chatbot. After the interaction was complete, users were directed to a third screen containing a concluding message. The result of our design process—depicted in Figure 6.1—was a customizable chatbot application that satisfied the ecological validity requirements and could be configured to maximize consistency across the conditions of our experiment.

### 6.2.2 Misplaced Modifiers

To select the sentences that our chatbot would use in the main part of our study, we conducted a preliminary survey. This survey aimed to identify phrases in which humor arises from the ambiguity introduced by a misplaced modifier. Ensuring that the sentences used in the experiment were funny was essential to justify the chatbot's reaction when presenting them to the user. Using Prolific—which has been shown to be suitable for online experiments in the social sciences (Palan et al., 2018)—we recruited a sample of 50 participants (23 male, 27 female) whose first language was English and who were born and currently live in the United States. The age of the participants ranged from 19 to 59 years, with a mean age of 34.2

( $s_x = 10.6$ ) and a median age of 33.0.

We first compiled 150 examples of misplaced modifiers from English grammar textbooks, writing guides, and academic papers on the subject. We then filtered out phrases that were too long ( $> 20$  words), were known jokes, depended on real-world references that could be unknown to the participant, were repetitive, abstruse, or had other grammatical errors. This process yielded 45 phrases from six sources (Blumenthal, 1981; Fordyce-Ruff, 2011; Hansen, 1983; Lederer, 1989; McLean, 2013; Nazario et al., 2010). We presented these 45 phrases alongside five control phrases. These control phrases were selected from the corrected versions of misplaced modifier examples that were not included in the final selection. Following previous studies on the rating of humorous content (Ruch, 2008), respondents were asked to rate each phrase using a seven-point Likert scale (1 = *not funny*, 7 = *very funny*). For each respondent, the order of the phrases was randomized.

Ratings for each phrase were averaged across all respondents, with a higher mean rating indicating a funnier phrase. We then ranked the phrases and selected the one with the lowest mean rating (P1) and the one with the highest (P2) to use in our experiment. This allowed us to test whether the effect of acknowledging humor was contingent on the intrinsic funniness of the phrase used in the interaction. Our selection is highlighted in Table 6.1.

Table 6.1: Phrases Selected for the Experiment

Phrase	Ranking	Text	Rating $\bar{x}$ ( $s_x$ )
P1	Least Funny	The candidate was falsely accused of covering up a crime by the media.	1.48 (1.13)
P2	Funniest	Each rancher must choose the appropriate method of castration for himself.	3.36 (2.15)

### 6.3 Methodology

The purpose of our evaluation was to address one main research question, which is an extension of **RQ3**:

- What is the effect of a chatbot’s acknowledgment of humor on the anthropomorphic perceptions it elicits? (**RQ3a**)

To approach this research question we conducted an online controlled experiment following a between-subjects,  $2 \times 4$  factorial design, with two phrases (see Table 6.1) and four acknowledgments comprising one control (no acknowledgment) and three treatments (emoji-only acknowledgment, verbal-only acknowledgment, verbal and emoji acknowledgment). For conditions containing an emoji, we used the *rolling on the floor laughing (ROFL)* emoji, as depicted in Figure 6.3. The ROFL emoji was chosen for its ability to nonverbally express the fact that the chatbot found the phrase funny. As summarized by Beattie et al. (2020), emoji

have been shown to be effective in conveying nonverbal cues in CMC. Table 6.2 summarizes the experimental setup.

Table 6.2: Experimental Setup

Condition	Acknowledgment	Phrase
CP1	None	P1
CP2	None	P2
T1P1	Emoji-Only	P1
T1P2	Emoji-Only	P2
T2P1	Verbal-Only	P1
T2P2	Verbal-Only	P2
T3P1	Verbal and Emoji	P1
T3P2	Verbal and Emoji	P2

### 6.3.1 Participants

As in the preliminary survey, we used Prolific to recruit participants whose first language was English and who were born and currently live in the United States, excluding participants who had completed the preliminary survey. As explained in Section 6.1, the recruitment of native English speakers aimed to minimize the risk of misunderstanding the phrases used in the experiment. Additionally, we required participants to access the study on a desktop device, to ensure a more homogeneous user experience.

For each of the eight conditions, we recruited between 50 and 56 respondents. Respondents were paid 0.50 GBP for participating in the five-minute study, in compliance with Prolific’s best practices at the time. The post-questionnaire included an attention check. Submissions that failed the attention check were rejected, triggering the recruitment of a new participant. For each condition, the first 50 complete submissions that passed the attention check were used for analysis, summing to 400 unique respondents in total.

Before interacting with the chatbot, participants completed a short pre-questionnaire providing information regarding their familiarity and frequency of interaction with chatbot technologies. A Kruskal-Wallis H test on the results of the pre-questionnaire indicated that there were no statistically significant differences in familiarity with chatbots ( $H = 2.89$ ,  $p = 0.895$ ) or frequency of interactions with chatbots ( $H = 1.56$ ,  $p = 0.980$ ). However, chi-square tests showed a slight discrepancy in gender balance for conditions T1P2 ( $\chi^2 = 5.80$ ,  $p = 0.016$ ) and T3P1 ( $\chi^2 = 3.92$ ,  $p = 0.048$ ), while analysis of variance (ANOVA) indicated a slight deviation from the global mean age ( $\bar{x} = 32.00$ ,  $s_x = 11.75$ ) in conditions CP1 ( $\bar{x} = 36.72$ ,  $s_x = 13.14$ ) and T1P1 ( $\bar{x} = 27.1$ ,  $s_x = 10.38$ ).

### 6.3.2 Procedure

The experimental part of our study was conducted in August 2020 during daytime hours in the United States. For all conditions, we configured the chatbot to run three short exchanges with participants. The first exchange introduced the grammatical context of the interaction. The chatbot explained that it had found a sentence that stood out in the text of an article it was reviewing. It then presented one of the sentences in Table 6.1 and asked the participant if the meaning of the sentence was clear. The participant could reply with *Yes*, *No*, or *I'm not sure*.

The second exchange varied across the experimental conditions. In the control (no acknowledgment) condition, the exchange consisted solely of an explanation of the alternative interpretation of the sentence introduced by the misplaced modifier. In the treatment conditions, the chatbot reacted by (i) displaying a ROFL emoji after explaining the ambiguity caused by the misplaced modifier (emoji-only acknowledgment), (ii) including the text "I find it hilarious!" in its explanation (verbal-only acknowledgment), or (iii) both including the text "I find it hilarious!" in its explanation and displaying the ROFL emoji (verbal and emoji acknowledgment). Under all conditions, the explanation was followed by a question asking participants if they agreed with the interpretation. Once more, participants could reply with *Yes*, *No*, or *I'm not sure*.

Finally, the chatbot proposed a reworded version of the original sentence to address the misplaced modifier. It then asked the participant if they thought the suggestion was good. As with the previous two exchanges, the participant could reply with "Yes", "No", or "I'm not sure". To ensure that both phrases were corrected using a consistent strategy, the suggestion was composed solely by rearranging the words in the original phrase. This means that although the suggested correction fixed the mistake introduced by the misplaced modifier, it may not be the optimal way of conveying the sentence's meaning. Sample interactions for experimental conditions CP2 and T3P2 can be seen in Figure 6.2 and Figure 6.3, respectively.

### 6.3.3 Instruments

Given the brevity and controlled nature of the interaction participants had with our chatbot, we did not identify an established instrument to appropriately measure the effect of our treatments on anthropomorphic perceptions. We, therefore, opted to operationalize the concept of *anthropomorphic perceptions* with participants' ratings with respect to different dimensions or *traits*. We selected these traits based on the literature. First, we adapted questions from Bartneck et al. (2009) and R. Zhao et al. (2018) addressing the perception of social traits (e.g., likability, friendliness, attentiveness) in artificial agents. Second, we incorporated items from Higashinaka et al. (2015) to address conversation-specific traits such as naturalness and continuity. Third, although personality is typically broken down into subcomponents following models such as the Big Five (Goldberg, 1993), certain subcomponents proposed by these models (e.g., openness to experience, neuroticism) were not applicable in our case. Hence, we included a question on personality as a whole. We followed a similar strategy



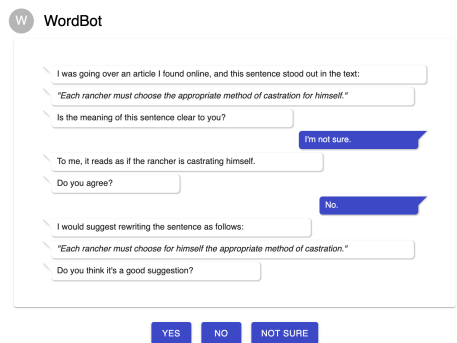


Figure 6.2: A sample interaction with the chatbot for the control condition using the funniest phrase (CP2). Users provided their responses via the buttons at the bottom of the chat window.

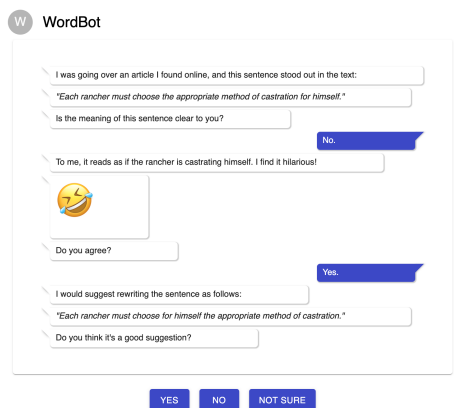


Figure 6.3: A sample interaction with the chatbot for the verbal and emoji treatment using the funniest phrase (T3P2). Note that the chatbot's responses are not affected by the user's input.

for humor, which has been identified to be multifaceted (Ruch, 2008), but was included as one trait in our questionnaire. Finally, we included a question on grammar, which has been identified as a barrier to the adoption of chatbots as conversational partners (Coniam, 2014).

The result was a post-questionnaire with two main sections (see Table 6.3). The first asked the participant to rate the chatbot using a five-point Likert scale (1 = *least*, 5 = *most*) with respect to seven traits. The second asked the participant to respond to three statements concerning three different traits using a five-point Likert scale (1 = *strongly disagree*, 5 = *strongly agree*). Within each section, questions were randomly shuffled for each participant.

### 6.3.4 Data Analysis

Data collected through the aforementioned instruments were analyzed using descriptive and inferential statistics. Specifically, we conducted Mann-Whitney U tests to evaluate the Likert scores of each treatment, trait, and phrase combination against those of its respective control condition. The Mann-Whitney U test was selected due to the nonparametric distribution of our results and the test's applicability to ordinal Likert-scale data. A statistically significant Mann-Whitney test for a control/treatment pair indicated that the respective treatment resulted in a significantly different distribution of ratings for the trait/phrase combination in question. As our data consists of five-point Likert scale responses, medians do not capture the differences between conditions effectively. We, therefore, report our test results alongside means for each condition, and use means to indicate the direction of any difference between a treatment and its respective control.

Table 6.3: Ten Traits Rated by Participants in the Post-Questionnaire

Trait	Formulation
<i>Rating on a Scale of 1 (Least) to 5 (Most)</i>	
Naturalness	Naturalness of conversation
Continuity	Continuity of conversation
Personality	Ability to convey personality
Empathy	Ability to understand humans
Humor	Having a sense of humor
Grammar	Knowledge of English grammar
Common Sense	Having common sense
<i>Agreement on a Scale of 1 (Strongly Disagree) to 5 (Strongly Agree)</i>	
Attentiveness	The chatbot was paying attention
Friendliness	The chatbot was friendly
Likability	I liked the chatbot

## 6.4 Results

For all ten traits considered, the results of statistically significant Mann-Whitney U tests—as well as the differences in mean Likert scores between treatment and control conditions—are summarized in Table 6.4. In this section, we report on the results of interest. When included, an increase (+) or a decrease (-) in the mean Likert score is with respect to the corresponding control condition. For brevity, we refer to conditions and phrases by the codes outlined in Table 6.2 and all tests refer to Mann-Whitney U tests.

Both personality and humor showed consistent, highly statistically significant increases across all treatment and phrase combinations. For personality, the largest increase in mean Likert score for P1 (+1.02) was observed in T3 ( $U = 1856.0$ ,  $p < 0.001$ ), while for P2 it was (+1.10) in T1 ( $U = 1940.5$ ,  $p < 0.001$ ). The smallest increase in mean Likert score for P1 (+0.78) was in T2 ( $U = 1693.5$ ,  $p = 0.002$ ), while for P2 (+0.94), it was in T3 ( $U = 1861.5$ ,  $p < 0.001$ ). Pairwise tests between treatment conditions did not show significant differences.

For humor, the highest increase was seen in T1, both for P1 (+1.94,  $U = 2199.5$ ,  $p < 0.001$ ) and P2 (+1.96,  $U = 2202.0$ ,  $p < 0.001$ ). The smallest increase was observed in T3, both for P1 (+1.84,  $U = 2152.0$ ,  $p < 0.001$ ) and P2 (+1.72,  $U = 2127.5$ ,  $p < 0.001$ ). Again, pairwise tests between treatment conditions did not show significant differences.

An increase in friendliness was also observed across all treatment and phrase combinations, though it was not statistically significant for the T2P2 (+0.32,  $U = 1476.5$ ,  $p = 0.085$ ) and T3P2 conditions (+0.08,  $U = 1289.5$ ,  $p = 0.771$ ). Other treatment conditions had consistent increases ranging from +0.34 ( $U = 1515.0$ ,  $p = 0.046$ ) for T3P1 to +0.40 ( $U = 1554.5$ ,  $p = 0.019$ ) for T1P2. Pairwise tests showed a significant difference only between T1P2 and T3P2 ( $U = 971.0$ ,  $p = 0.031$ ).

Table 6.4: Means and Standard Deviations for Five-Point Likert Scores by Trait and Phrase

Trait	Phrase	Control	Treatment		
		(C)	Emoji-Only (T1)	Verbal-Only (T2)	Verbal and Emoji (T3)
Naturalness	P1	4.04 (0.88)	3.72 (1.21)	4.02 (1.08)	4.04 (0.99)
	P2	4.14 (0.70)	4.12 (0.82)	4.04 (1.01)	3.82 (1.14)
Continuity	P1	4.28 (1.03)	4.34 (1.04)	4.22 (0.84)	4.16 (0.89)
	P2	4.20 (0.76)	4.26 (0.75)	4.08 (1.01)	4.00 (0.95)
Personality	P1	2.94 (1.17)	<b>3.78 (1.15)***</b>	<b>3.72 (1.11)**</b>	<b>3.96 (1.09)***</b>
	P2	3.12 (1.06)	<b>4.22 (0.89)***</b>	<b>4.16 (0.91)***</b>	<b>4.06 (0.91)***</b>
Empathy	P1	4.18 (0.85)	4.02 (1.08)	<b>3.82 (0.96)*</b>	3.84 (1.02)
	P2	3.88 (0.85)	4.06 (0.84)	4.02 (0.98)	3.74 (0.99)
Humor	P1	2.00 (1.09)	<b>3.86 (1.34)***</b>	<b>3.94 (1.08)***</b>	<b>3.84 (1.67)***</b>
	P2	2.50 (1.28)	<b>4.32 (0.87)***</b>	<b>4.46 (0.86)***</b>	<b>4.22 (0.84)***</b>
Grammar	P1	4.30 (0.79)	4.56 (0.61)	4.44 (0.67)	4.42 (0.86)
	P2	4.40 (0.76)	4.40 (0.70)	4.46 (0.68)	4.12 (0.85)
Common Sense	P1	4.06 (1.02)	4.10 (1.05)	3.92 (1.01)	4.12 (0.94)
	P2	4.18 (0.87)	4.34 (0.77)	4.00 (1.12)	<b>3.70 (0.95)**</b>
Attentiveness	P1	4.44 (0.79)	4.24 (1.02)	4.18 (0.96)	4.14 (0.97)
	P2	4.18 (0.94)	4.12 (0.82)	4.16 (1.02)	<b>3.66 (1.15)*</b>
Friendliness	P1	4.12 (0.92)	<b>4.48 (0.81)*</b>	<b>4.48 (0.74)*</b>	<b>4.46 (0.76)*</b>
	P2	4.20 (0.90)	<b>4.60 (0.61)*</b>	4.52 (0.61)	4.28 (0.78)
Likability	P1	4.20 (0.93)	4.22 (0.95)	4.30 (0.93)	4.18 (0.87)
	P2	4.18 (0.85)	4.24 (0.82)	4.26 (0.88)	3.98 (0.98)

Results in **blue** indicate an increase with respect to the control condition while those in **red** indicate a decrease. Results in **bold** are statistically significant (\* $p < 0.05$ , \*\* $p < 0.01$ , \*\*\* $p < 0.001$ ).

Continuity, empathy, grammar, common sense, and likability all showed mixed effects, with significant decreases observed in T2P1 for empathy ( $-0.36$ ,  $U = 981.0$ ,  $p = 0.050$ ) and T3P2 for common sense ( $-0.48$ ,  $U = 894.5$ ,  $p = 0.010$ ). Pairwise tests between the treatment conditions within each trait showed a significant difference between T3P2 and T2P2 for grammar ( $U = 975.5$ ,  $p = 0.040$ ), and T3P2 and T1P2 for common sense ( $U = 769.0$ ,  $p < 0.001$ ). All other pairwise comparisons were not significant.

Finally, the results for naturalness and attentiveness showed a consistent decrease in mean Likert scores across all conditions except T3P1 for naturalness, which remained equal. Pairwise tests between treatment conditions for naturalness did not show significant differences. Of particular note is condition T3 for attentiveness, which experienced the largest decrease both for P1 ( $-0.30$ ,  $U = 1032.0$ ,  $p = 0.100$ ) and P2 ( $-0.52$ ,  $U = 925.5$ ,  $p = 0.019$ ). Pairwise tests between treatment conditions for attentiveness showed a significant difference only between T2P2 and T3P2 ( $U = 931.0$ ,  $p = 0.021$ ).

### 6.5 Discussion and Implications

Although we observed statistically significant increases across all treatments for personality and humor, this was not the case for the other eight traits. However, there are a number of promising results and parallels with previous findings.

First, the increases for humor across all treatment conditions indicate that the direct effect of our manipulation—to acknowledge humor—was achieved. Moreover, the positive correlation between the effects on humor and personality is aligned with work that has highlighted a close relationship between both traits (Medhi Thies et al., 2017; Thorson et al., 1993). The observed effects also have implications in practice. As personality has been found to help chatbots gain user trust and provide a more engaging and enjoyable experience (Jain et al., 2018), chatbots should incorporate behaviors that aim to improve the way users perceive their personalities. If this improvement can be achieved with a simple interaction similar to those used in this study, it would provide an alternative to the use of jokes, which have been widely used to introduce humor in HCI (Dybala et al., 2009a; Ptaszynski et al., 2010; Schanke et al., 2021; Wu, 2008), but might become repetitive if overused (Følstad & Brandtzaeg, 2020). In language learning, one could envision a chatbot that sparsely incorporates the type of interaction used in this study, either by reacting to user inputs that contain misplaced modifiers or through short, predefined exercises on the topic. In other educational domains, such as software engineering education, these interactions could be introduced in the context of ambiguous variable names, comments, or in annotations that developers themselves provide within the code review process.

Second, we did not observe any significant differences in naturalness, continuity, or grammar under any treatment condition. Since these are traits related to the conversational capability of our chatbot, our results suggest that the introduction of an acknowledgment did not affect the user's perception of the chatbot's ability to communicate. Ensuring naturalness and continuity

of conversation when introducing diversions is especially important for task-oriented interactions, for which traditional views regarded humor as distracting and time-wasting (Morkes et al., 1999). Taking into account only these conversational traits, our results suggest that the interactions used in the treatment conditions pose little risk of eliciting significant unintended negative effects. These findings could be used to argue for implementing such interactions in task-oriented chatbots for education.

Chatbots that can provide relatable explanations through humor without sacrificing user experience could be a positive enhancement for digital education platforms. A recent study on the use of the writing software *Grammarly* suggested that it be used in conjunction with an academic learning advisor (ALA) to provide support due to the occasionally confusing or inaccurate feedback provided by the software (O'Neill et al., 2019). Incorporating a chatbot that could serve this purpose when the ALA is not available, or in blended and remote learning scenarios facilitated by LXPs, could help mitigate this issue. Nevertheless, significant decreases in the results of three other traits (empathy, common sense, and attentiveness)—though for each observed in only one condition/phrase combination—serve as a reminder that playful interactions, such as the acknowledgment of humor, can introduce an overall higher expectation of the chatbot, which the chatbot might not be able to extend to other traits (Luger et al., 2016).

These results also have concrete implications for the work conducted in the rest of this thesis. First, our findings suggest that chatbots could, indeed, harness humor from ambiguous grammatical constructs to strengthen anthropomorphic perceptions of traits such as personality and friendliness. Equipping chatbots with ways of exploiting humor arising from ambiguity or incongruity is one strategy that we propose as a part of our conceptual framework. As discussed above, this strategy could help improve the learning experience without distracting the learner from the task at hand. Second, the short, directed interactions harnessed in this experiment provided a base for the design of our rule-based interactions. Following the structure of the interactions used in this experiment, chatbots initiate the exchange with learners in the context of an issue present in a particular learning resource (e.g., text, code) and hook them into the interaction through questions or comments about that issue. Third, certain elements of this initial chatbot interface were adapted for the interface that was embedded in Code Review and used for our empirical evaluations in software engineering education. These include the best practices mentioned in Section 6.2.1. Finally, our results informed a further exploration of how humor can be expressed through emoji in interactions between developers and chatbots on social coding platforms. This exploration served to define another interaction strategy and will be presented in the following chapter.

## **6.6 Conclusion**

In this chapter, we presented a first study aimed at exploring how to integrate humor into the interactions that learners have with educational chatbots. Findings from our controlled online

experiment contribute to the design of research-informed educational applications and, more specifically, to the implementation of strategies for incorporating humor in human-chatbot interaction for education. Possible applications outside language learning—which we will explore in future chapters—include adaptations to introductory programming courses, where humor could be useful to teach best practices in software engineering, given the wide use of chatbots on software development platforms (Wessel, de Souza, et al., 2018).

However, our study has certain limitations that are worth addressing in future work. The first limitation stems from the use of only two phrases with possibly limited intrinsic funniness. Phrases could have also had other unmeasured attributes (e.g., offensive, confusing), introducing biases that were not captured in the preliminary survey. Further investigating the effect of the phrase on our results would provide more rigorous conclusions on the effects of the treatments. A second limitation concerns external validity. To minimize misapprehension, we only recruited native English speakers. Although misplaced modifiers also affect native speakers (Edlund et al., 2012; Joubert et al., 2015), a logical next step would be to validate our current findings with ESL learners. Applicability to other languages and demographic groups should also be tested, as perceptions of human traits, especially humor, might be contingent on cultural and linguistic factors (Alden et al., 1993). Moreover, our questionnaire was explicitly built to complement the short nature of the interaction with our chatbot, but traits such as humor and personality are multifaceted in nature and could be better captured by a more robust questionnaire.

An important limitation to note concerns the interactions participants had with our chatbot. These interactions lacked real-world context, had a fixed script, and only allowed participants to reply using quick response buttons. Future research could benefit from embedding the interaction within a real-world setting and providing more flexibility, such as open-ended typing and conversational scripts that adapt to user inputs. We will investigate interactions in real-world settings in Chapters 8–10, exploring the use of rule-based scripts and LLM-powered chatbots in Chapters 9 and 10, respectively.

## 7 Emoji Reactions

THE widespread use of chatbots to support software development makes social coding platforms such as GitHub a particularly rich source of data for the study of human-chatbot interaction. Software development chatbots are used to automate repetitive tasks, interacting with their human counterparts via comments posted on the various discussion interfaces available on such platforms. One type of interaction supported by GitHub involves reacting to comments using predefined emoji. To investigate how this strategy could be adopted for interactions between chatbots and learners in the context of software engineering education, we explored how developers react to chatbot comments on GitHub. Building on the work presented in the previous chapter, we focus particularly on comments that elicited the laugh reaction. The findings from our analysis suggest that some reaction types are not equally distributed across human and bot comments and that a chatbot's design and purpose influence the types of reactions it receives. Furthermore, while the laugh reaction is not exclusively used to express laughter, it can be used to convey humor when a chatbot behaves unexpectedly. These insights could inform the way chatbots are designed and help developers equip them with the ability to recognize and recover from unanticipated situations. In turn, chatbots could better support the communication, collaboration, and productivity of teams using social coding platforms, as well as of learners using code review notebooks.

By investigating how users interact with chatbots on a social coding platform, this chapter partially addresses **RQ3**:

How can we equip educational chatbots with effective interaction strategies?

The content of this chapter was partially presented in the following publication:

- Farah, J. C., Spaenlehauer, B., Lu, X., Ingram, S., & Gillet, D. (2022). An Exploratory Study of Reactions to Bot Comments on GitHub. *2022 IEEE/ACM 4th International Workshop on Bots in Software Engineering (BotSE)*. <https://doi.org/10.1145/3528228.3528409>

### 7.1 Introduction

The popularity of social coding platforms for software development has been steadily rising over the past decade. These platforms support several tasks related to the software development process, such as flagging issues, proposing changes to address those issues, and reviewing the proposed changes. For some tasks, users can interact using discussion threads, which are associated with a given issue or proposed change. Chatbots that support developers with these tasks have become popular and their participation in discussions is increasingly commonplace (Wessel, de Souza, et al., 2018). Those chatbots—also referred to as *bots* in the context of social coding platforms—interact using natural language via comments. Users can then respond to these comments and—in some platforms (e.g., GitHub)—react to them using predefined emoji (see Figure 7.1).

Although researchers have separately studied the use of bots on GitHub (Wessel, de Souza, et al., 2018; Wyrich et al., 2021) and the role of reactions in discussion threads (Borges et al., 2019; Son et al., 2021), only a few studies have addressed how users interact with bots via the reaction interface, and these have focused on specific bots (C. Brown et al., 2019) or on providing general design principles (Liu et al., 2020). Furthermore—to the best of our knowledge—no study has investigated how users interact with bot comments using the laugh reaction. Humor has been proposed to make interactions with chatbots more enjoyable (Dybala et al., 2009b; Jain et al., 2018)—including in task-oriented settings (Morkes et al., 1999; Niculescu, van Dijk, et al., 2013)—and as a way for chatbots to recover from errors (Niculescu & Banchs, 2015). Hence, a better understanding of what makes GitHub users *laugh* at (or with) bots could help improve chatbot user experiences on social coding platforms and in pedagogical scenarios that resemble these platforms, such as those provided by code review notebooks.

To address this gap, we conducted an exploratory observational study that examined user reactions to over 54 million comments made on GitHub issue threads throughout 2020. Findings from this study provide a second strategy that could be used to configure how educational chatbots interact with learners, making up a part of the fifth contribution in this thesis (C5).

This chapter is structured as follows. In Section 7.2 we present the methodology followed for our observational study. We then present our results in Section 7.3 and discuss their implications in Section 7.4. Finally, we conclude in Section 7.5.









Name	+1	-1	laugh	confused	heart	hooray	rocket	eyes
Emoji								

Figure 7.1: There are eight emoji reactions available on GitHub.



## 7.2 Methodology

The purpose of our study was to address one main research question, which is an extension of RQ3:

- How do developers interact with chatbots using GitHub’s reaction interface? (RQ3b)

Our analysis focused on three aspects of these interactions. First, we looked at the overall differences between how users react to human comments versus how they react to bot comments. Second, we selected ten popular bots and investigated how different bots elicit different reactions from users. For these two aspects, we first considered all reactions and then focused on the `laugh` reaction. A third aspect consisted of a qualitative analysis of randomly selected bot comments with `laugh` reactions and was aimed at understanding the individual characteristics of those comments.

### 7.2.1 Data Acquisition

Our study used data acquired between February and May 2021 from the 2020 GitHub event timeline, which we extracted using GH Archive (Grigorik, 2012). We focused on comments made on issues, extracting events of type `IssueCommentEvent`. These data provided us with 54,394,463 events. To retrieve the reactions to these events, we used GitHub’s REST API. This yielded 3,476,282 comments with at least one reaction and a total of 5,427,039 reactions. Note that some comments were unavailable at the time of access via the API, possibly due to deletion. We then used GitHub users’ `type` attribute to partition these comments into those that were made by a human (3,457,495) and those that were made by a bot (18,787).

### 7.2.2 Data Analysis

To analyze our dataset, we followed a mixed-method approach. We used descriptive statistics across all three aspects of our analysis. Specifically, we report counts ( $n$ ), as well as sample means ( $\bar{x}$ ), medians ( $\tilde{x}$ ), and standard deviations ( $s_x$ ). We also used sentiment analysis and inferential statistics to probe whether bot comments that elicited a `laugh` reaction were different from their counterparts made by humans. For our sentiment analysis, we first filtered the comments to include only those in English and excluded those longer than or equal to 50,000 characters and shorter than or equal to 50 characters. We then applied VADER (Hutto et al., 2014), which assigned a sentiment score ranging from  $-1$  to  $+1$  to each comment, and SentiCR (Ahmed et al., 2017), which classified the comments as *nonnegative* or *negative*. We report the results of both methods. Finally, we manually inspected the issue discussion threads for 100 randomly selected bot comments with `laugh` reactions, performing a qualitative analysis consisting of the following questions:

1. *Was the reason for the laugh reaction evident?*
2. *Was the explanation for the laugh reaction dependent on the comment's context or was it standalone (i.e., only dependent on the comment itself)?*
3. *Was the laugh reaction explicitly sought by the bot comment (i.e., directed at eliciting the laugh reaction)?*
4. *Was the laugh reaction caused by an incongruity (i.e., an inconsistency between what users might expect from the comment and what was actually presented in it)?*

To identify common themes across the different issue discussion threads, we coded each comment, following a strategy similar to paragraph-by-paragraph coding (Urquhart, 2012), and proposed an explanation for the presence of the laugh reaction.

### 7.3 Results

We report our results with respect to the three aspects of our study.

#### 7.3.1 Human versus Bot

Bot comments with at least one reaction were made by 311 distinct bots over 16,370 issues in 6057 repositories. Human comments with at least one reaction were made by 691,361 distinct users, over 2,221,510 issues in 322,390 repositories.

#### All Reactions

We identified 5,397,646 reactions (from 955,515 unique users) to comments made by humans and 29,393 reactions (from 14,264 unique users) to comments made by bots. These reactions were distributed over 3,457,495 human comments and 18,787 bot comments, from a total of 38,744,040 human and 10,322,745 bot comments that were available at the time of access. This means that 8.92% of human comments elicited reactions, while only 0.182% of bot comments elicited reactions.

The proportion of the total number of reactions by reaction type for both comments from human and bot users is shown in Figure 7.2. Most reactions have approximately the same proportion for both types of users. However, while for humans, 71.6% of reactions were a +1, +1 only accounted for 48.1% of reactions to bot comments. At the other end of the spectrum, the proportion of the -1 reaction was higher for bots (14.5%) than for humans (2.19%).

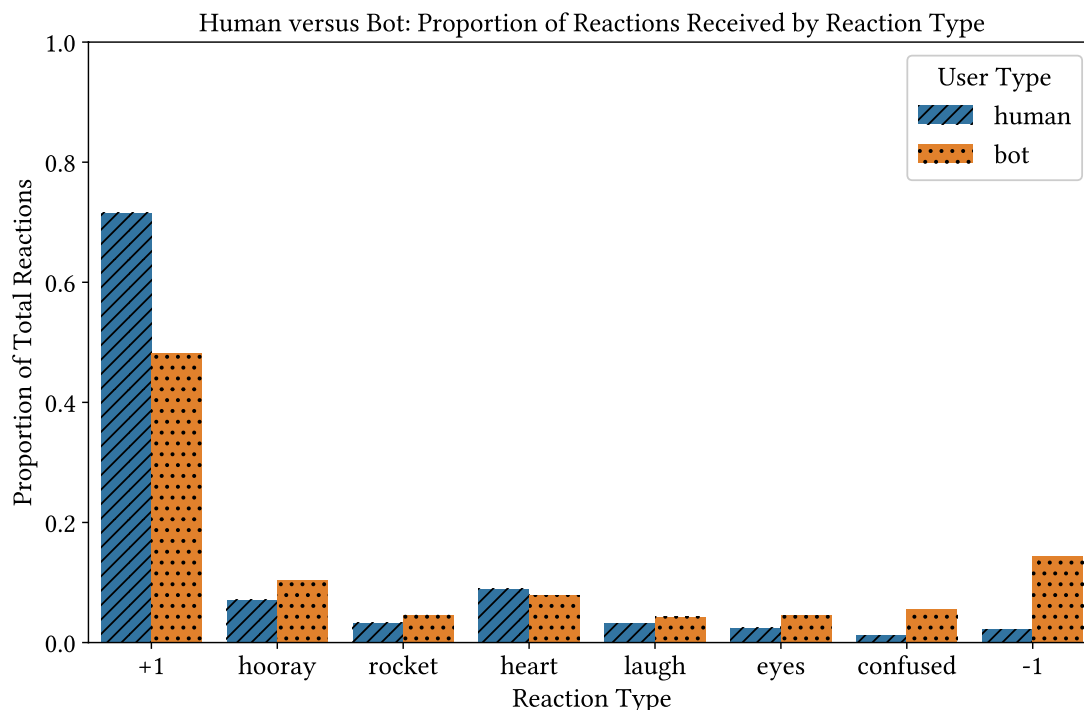


Figure 7.2: Proportion of reactions received—by reaction type—for both human and bot comments.

### Laugh Reaction

A total of 125,442 comments made by humans included a laugh reaction, while 1073 bot comments included a laugh reaction. Of these comments, 83,406 human and 974 bot comments were included in our sentiment analysis. Using VADER, human comments received a mean score of  $\bar{x} = 0.291$  ( $\tilde{x} = 0.368$ ,  $s_x = 0.470$ ), while bot comments received a mean score of  $\bar{x} = 0.290$  ( $\tilde{x} = 0.250$ ,  $s_x = 0.433$ ). As the distributions of sentiment scores were nonparametric, we performed a Mann-Whitney U test to probe for differences between the scores of the human and bot comments, observing no significant differences between them ( $U = 40047367.0$ ,  $p = 0.224$ ). Using SentiCR, 83.4% (69,596) of human comments were classified as *nonnegative* while 16.6% (13,810) were classified as *negative*. For bot comments, 89.6% (873) were classified as *nonnegative* while 10.4% (101) were classified as *negative*.

### 7.3.2 Popular Bots

We selected 10 bots for further analysis based on the number of unique users that reacted to comments posted (see Table 7.1).

Table 7.1: GitHub Bots that Received Reactions from the Highest Number of Distinct Users

Bot	Description	Distinct Users
issue-label-bot [bot]	Automatically labels issues as either a feature request, bug, or question.	3564
github-actions [bot]	Performs automated workflows supported by GitHub actions.	2138
stale [bot]	Closes abandoned issues after a period of inactivity.	1789
github-learning-lab [bot]	Helps users learn how to use GitHub.	1595
vscodebot [bot]	Bot used by the VSCode repository.	966
msftbot [bot]	Microsoft's GitHub bot.	454
allcontributors [bot]	Automatically adds contributor acknowledgments.	415
welcome [bot]	Welcomes new users to a repository.	388
dependabot [bot]	Detects and updates vulnerable dependencies.	388
codecov [bot]	Provides coverage reports and helps with the code review workflow.	302

## All Reactions

The proportion of the total number of reactions by reaction type for our selection of popular bots is shown in Figure 7.3. For certain reaction types, a few bots stand out. A total of 65.0% of reactions to comments by stale [bot], for example, are a -1, while the group mean is  $\bar{x} = 14.7\%$  ( $\tilde{x} = 13.2\%$ ,  $s_x = 18.2\%$ ). This is also the case for the heart reaction ( $\bar{x} = 9.23\%$ ,  $\tilde{x} = 8.03\%$ ,  $s_x = 8.35\%$ ), which is more prominent in comments by welcome [bot] (28.8%) and allcontributors [bot] (19.0%). For the +1 reaction—which has a wide distribution of proportions ( $\bar{x} = 38.7\%$ ,  $\tilde{x} = 31.0\%$ ,  $s_x = 22.8\%$ )—issue-label-bot [bot] stands out, with +1 representing 87.5% of reactions to its comments. Other reaction types, such as rocket ( $\bar{x} = 3.77\%$ ,  $\tilde{x} = 3.62\%$ ,  $s_x = 2.38\%$ ), have narrower distributions.

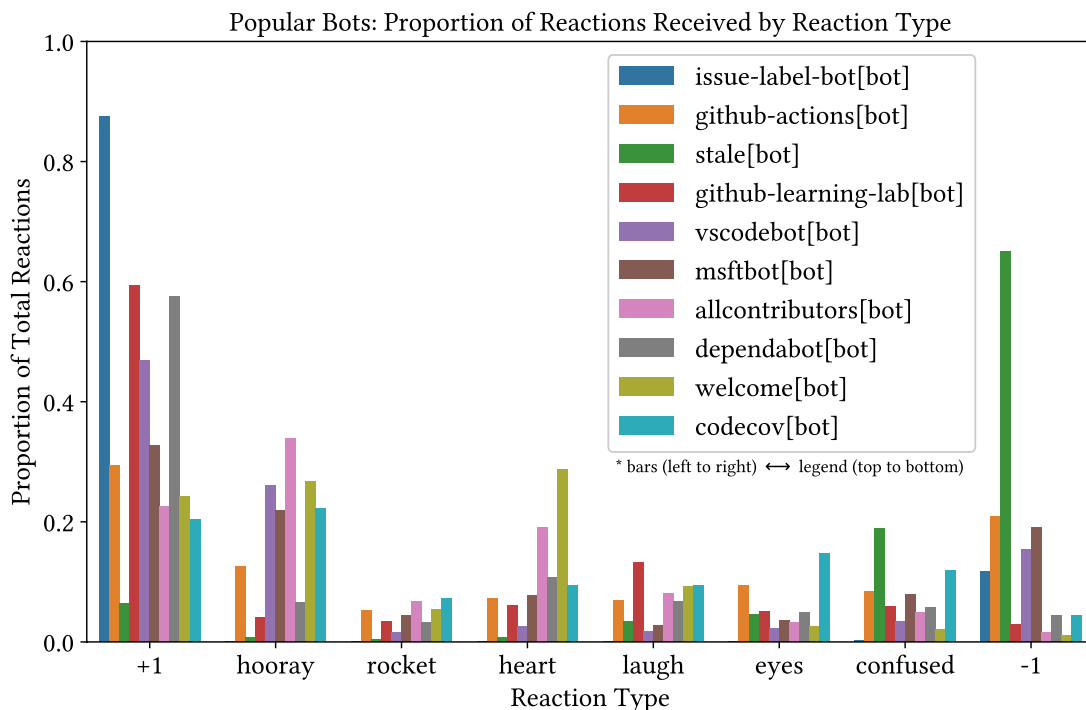


Figure 7.3: Proportion of reactions received—by reaction type—for a selection of popular bots.

### Laugh Reaction

For the laugh reaction, the distribution among these bots had a mean of  $\bar{x} = 6.17\%$  ( $\bar{x} = 6.50\%$ ,  $s_x = 3.86\%$ ). The bot with the highest proportion of laugh reactions was `github-learning-lab[bot]` (13.2%), while the bot with the most laugh reactions was `github-actions[bot]`, with a total of 398 reactions over 315 comments. Finally, the bot with the highest average laugh reactions per comment (including only comments with laugh reactions) was `vscodebot[bot]` ( $\bar{x} = 2.33$ ,  $\tilde{x} = 2$ ,  $s_x = 1.66$ ,  $n = 9$ ).

### 7.3.3 Qualitative Analysis


The comments selected for our qualitative analysis were made by 26 different bots. For 91 of the 100 comments in the sample, the presence of the laugh reaction was evident after going through the issue discussion thread, while for the other 9, this was not the case. For 41 comments, the explanation for the laugh reaction depended on the context, while 57 were standalone, and for 2, this was unclear. Only 2 comments explicitly sought the laugh reaction. Of these 2, one was a joke and the other contained a funny image. Finally, for 29 comments, the laugh reaction could have been caused by an incongruity, while for 61 comments this was apparently not the case, and for 10 comments, this was unclear. Figure 7.4 depicts an example of such incongruity. In this example, a user opened an issue in the GitHub repository of Microsoft's Visual Studio Code—a popular code editor—to presumably post content in support of Donald Trump during the 2020 United States presidential elections. The repository's `vscodebot[bot]`, which is programmed to detect duplicates, suggested that this issue could be a duplicate of another, completely unrelated issue, possibly due to the presence of the word *support* in both issues. This incongruity results in two users reacting with the laugh emoji.

The most prominent themes were *tutorial* (32 comments), *report* (16 comments), and *closing issue* (16 comments). The *tutorial* theme was only present in comments by `github-learning-lab[bot]`, while *report* was present in comments by 10 different bots, and *closing issue* in comments by 4 different bots. Of these three themes, *tutorial* comments were mostly not incongruent (Yes = 1, No = 31, Maybe = 1), while incongruity appeared more in *report* (Yes = 10, No = 5, Maybe = 1) and *closing issue* comments (Yes = 8, No = 3, Maybe = 5). Finally, some noteworthy explanations for the presence of a laugh reaction due to incongruity included the following: (i) reaction to a bot closing an issue that had recently been reported as still present, (ii) reaction to a bot thanking another bot for its contribution, and (iii) reaction from a user to being welcomed by a bot that the user himself created.

## 7.4 Discussion and Implications

Overall, our results show that while a far smaller percentage of bot comments receive reactions when compared to human comments, the proportions of the types of reactions received are mostly similar across both groups. One notable exception is the proportion of -1 reactions.

## Support Donald Trump #94842

 Closed ghost opened this issue on Apr 10, 2020 · 2 comments

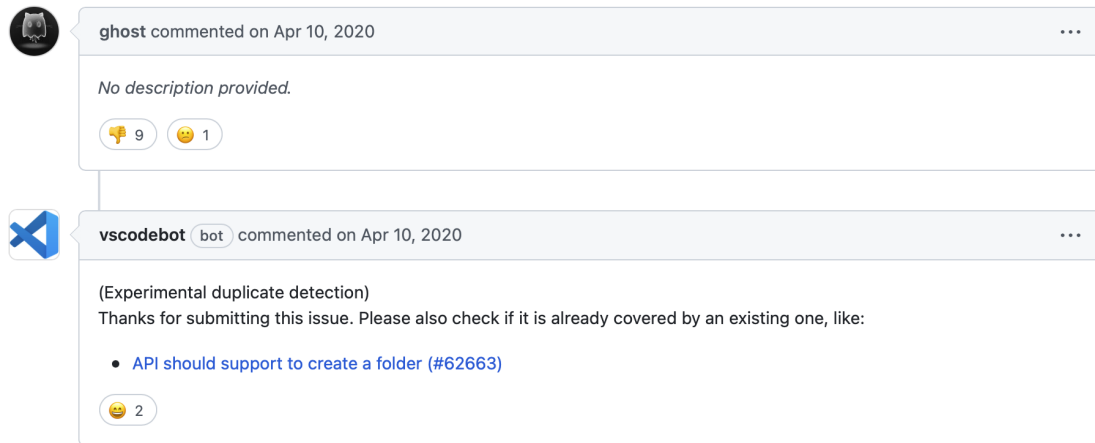


Figure 7.4: Example of the laugh reaction to a bot comment being caused by an incongruity. Although the issue is clearly spam, `vscodebot [bot]` suggests that it could be a duplicate of something completely unrelated, possibly due to the presence of the word *support*.

The fact that bots receive more -1 reactions than humans could be attributed to users being more willing to give negative feedback to bots and to the fact that some bots perform tasks that are regarded as friction points in the software development process (e.g., closing an issue, requesting a review), as noted by Lebeuf et al. (2017). For `stale [bot]`, for example, its role in closing issues results in it getting a disproportionate share of -1 reactions. On the other hand, bots that perform tasks that are regarded as positive, such as `welcome [bot]` and `allcontributors [bot]`, naturally get a higher proportion of heart reactions than the average. Receiving an unusually high (or low) proportion of a given type of reaction can also be due to the design of the bot. A closer inspection of `issue-label-bot [bot]`'s comments, for instance, shows that it explicitly asks for a -1 or +1 to refine the way it labels issues, as was pointed out by Liu et al. (2020).

Focusing on the results for the laugh reaction specifically, we observe that there are no significant differences between how this reaction is used to react to human versus bot comments. Similarly, none of the bots in our selection stood out in terms of the proportion of laugh reactions received. Although the proportion of laugh reactions to `github-learning-lab [bot]` was the highest in the selection, our qualitative analysis showed that most of these reactions were not due to any type of humor being evoked by the bot. In fact, the laugh reaction seems to be used not only to express laughter—or as a reaction to an interaction that could be considered humorous—but also to show that a user is pleased. This could be due to the choice of emoji used to represent the laugh reaction on GitHub, which is described as *Grinning Face with Smiling Eyes* in Unicode's Common Locale Data Repository (Unicode, 2021), and might not be explicitly associated with humor. Indeed, Borges et al. (2019) describe the laugh

reaction as being used to express a “fun situation *or* [emphasis added] happiness” (Borges et al., 2019). Nevertheless, our qualitative results also show that the `laugh` reaction *can* be used to express humor. In these cases, humor is usually not explicitly sought. Instead, it arises from an incongruity and is most commonly related to an unexpected report or a bot trying to close an issue. These findings are once again aligned with the work of Borges et al., who also noted that `laugh` can be used to “express sarcasm or irony in negative situations” (Borges et al., 2019). In these cases, the presence of the unanticipated `laugh` reaction could serve as a signal that the bot’s behavior was not aligned with user expectations. Bots could then follow `issue-label-bot` [bot]’s strategy and refine their behavior accordingly, potentially even harnessing the underlying humor to recover from such situations.

These results also have concrete implications for the work conducted in the rest of this thesis. First, our observational study showed that the reaction interface on GitHub is widely used by developers, including in their interactions with chatbots. Therefore, we integrated a reaction interface into Code Review, allowing learners and educators to quickly react to both human and chatbot comments using emoji. Following our observation that the choice of emoji used to represent the `laugh` reaction on GitHub might not be explicitly associated with humor, our interface includes a ninth emoji (described as *Face with Tears of Joy* in Unicode’s Common Locale Data Repository (Unicode, 2021)) for this purpose. Second, the fact that developers reacted differently to human comments than they did to chatbot comments sheds light on an opportunity for educators to harness chatbots in order to gain further insight into the way learners perceive the activities they take part in. That is, learners might be more willing to provide certain types of feedback to a chatbot than they would to a human interlocutor. Third, the differences that arose in how developers interact with different chatbots highlight the importance of a chatbot’s identity—and the task it is associated with—on the interactions it has with users. This guided our use of neutral yet explicit chatbot identities in the empirical studies presented in Chapters 8–10 and informed certain aspects of the conceptual model for participatory design that will be covered in Chapter 11. Finally, given that when the `laugh` reaction was used to convey the presence of humor, such humor was usually not explicitly sought by the chatbot, we only included a few chatbot comments in our empirical studies that explicitly incorporated humor.

## 7.5 Conclusion

In this chapter, we presented results from an observational study analyzing how users react to comments on GitHub. Our findings suggest that while some reaction types are used differently when reacting to human versus bot comments or across comments made by different bots, this is not the case for the `laugh` reaction. Furthermore, while the `laugh` reaction is not exclusively used to express humor, it *can* be used to express humor arising from an incongruity when a bot behaves unexpectedly. These insights could inform the design of bots that can take into account how users react to their comments in order to align their behavior with user expectations. While this study explicitly considered the `laugh` reaction, extensions to other

reactions could be relevant to the study of how chatbots harness social cues.

This study also has some limitations worth discussing. First, we do not take into account bot users that are not labeled as bots. To address this, we could incorporate methods proposed for identifying bots into our data acquisition pipeline (Dey et al., 2020; Golzadeh et al., 2020). Second, our dataset consists only of comments made on issue threads, even though bots also comment on pull requests (Wyrich et al., 2021). Including those comments would broaden the scope of our analysis. Third, some bots follow predefined behaviors that bias the reactions they receive. These behaviors should be taken into consideration to avoid arriving at misleading conclusions. Finally, our qualitative analysis only included a sample of 100 bot comments. Annotating a larger sample would provide a more solid base for our findings. Our aim is to build on this study to explore this line of research and address the aforementioned limitations in future work.



## 8 Wizard of Oz

IN this chapter, we return our focus to how chatbots can be integrated into software engineering education. For this purpose, we developed a new version of the code review notebooks presented in Chapter 5. This version featured an enhanced Code Review app that allowed educators to interact with learners using chatbot identities. That is, educators could impersonate these chatbot identities in their interactions with students, following the Wizard of Oz technique. We then used this code review notebook to conduct a lesson on software engineering best practices and evaluated it through a controlled in-class experiment. This experiment examined the effect that explaining content via chatbot identities had on three aspects: (i) students' perceived usability of the lesson, (ii) their engagement with the code review process, (iii) their short-term learning gains, and (iv) the feedback they provided. While our findings show that the code review notebook used achieved good usability across all conditions, our quantitative analysis did not yield significant differences between conditions for any of the aspects considered. Nevertheless, our qualitative results suggest that students expect explicit feedback when performing this type of exercise and could thus benefit from automated replies provided by an interactive chatbot.

By investigating the effects of incorporating Wizard of Oz chatbots into a software engineering lesson, this chapter partially addresses **RQ3**:

How can we equip educational chatbots with effective interaction strategies?

The content of this chapter was partially presented in the following publication:

1. Farah, J. C., Spaenlehauer, B., Sharma, V., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices. *2022 IEEE Global Engineering Education Conference (EDUCON)*, 1634–1642. <https://doi.org/10.1109/EDUCON52537.2022.9766793>

### 8.1 Introduction

In the introduction to Part III of this thesis, we noted that while several studies have evaluated the integration of social coding features into educational contexts (Zagalsky et al., 2015), the impact that chatbots could have on supporting these integrations has not been addressed in the literature. In particular, although the code review process has been found to be suitable for educational purposes (X. Li et al., 2005; Y. Wang et al., 2012), no study has assessed the role chatbots could play in supporting code review exercises.

Outside of education, a recent exploratory study showed that adopting software agents to perform code reviews had a positive impact on contributions to open-source software projects and also resulted in decreased communication between maintainers and contributors (Wessel, Serebrenik, et al., 2020a). Though preliminary, these results may imply that chatbots could play a positive role in software engineering education. Drawing a parallel from a social coding platform to an educational context—with contributors as students and maintainers as instructors—we posit that chatbots might be able to increase student engagement during code review exercises while also reducing instructors’ workloads with respect to the guidance required during these exercises. In this chapter, we present the first of a series of empirical studies in which we investigate the effects of using a chatbot to present content related to code quality standards within a simulated code review exercise.

We start by presenting how we enhanced Code Review to allow educators to integrate chatbots into the code review process. Through a new configuration panel, educators could define chatbot identities and then use them to provide comments in the examples selected to illustrate the code review process. Nevertheless, it is important to note that, in this implementation, these chatbots only participate in the code review process when impersonated by users with access to the educator interface (e.g., instructors, teaching assistants). That is, our chatbots cannot interact automatically with learners. This interaction strategy is referred to as the Wizard of Oz technique. In Wizard of Oz studies, “subjects are told that they are interacting with a computer system through a natural-language interface, though in fact they are not”, since there is a human controlling the interface behind the scenes (Dahlbäck et al., 1993).

We then evaluated this interaction strategy in a controlled in-class experiment comprising 30 undergraduate software engineering students. Our objective was to test whether there was an effect on perceived usability, student engagement, learning gains, and feedback if explanations regarding code quality were provided differently. To do this, we evaluated two treatments: (i) providing explanations as the course instructor within Code Review (instructor condition) and (ii) providing explanations within Code Review through a chatbot that the course instructor impersonated (chatbot condition). Finally, to provide a baseline, we also included a condition in which Code Review was only used to display the code snippet, while explanations of the issues therein were provided in the text preceding Code Review (control condition).

Findings from this study provide a third strategy that could be used to configure how educa-

tional chatbots interact with learners, making up a part of the fifth contribution in this thesis (C5). Specifically, our objective is to shed light on whether the simple act of presenting students with information via a static chatbot has an effect on the learning experience. Moreover—to the best of our knowledge—no study has evaluated the impact of incorporating chatbots in a code review exercise within a blended learning scenario. Given the widespread use of chatbots in social platforms and the trend to incorporate these platforms into educational contexts, we believe that our study is both timely and relevant.

This chapter is structured as follows. In Section 8.2, we present the design of our chatbot integration. We outline our methodology in Section 8.3 and report on the results in Section 8.4. These results are discussed in Section 8.5 alongside the implications for the rest of the work conducted in this thesis. Finally, we summarize our conclusions in Section 8.6.

## 8.2 Design

In this section, we show how we integrated a chatbot interface into Code Review and present *Lint Bot*, the chatbot identity used in this study.

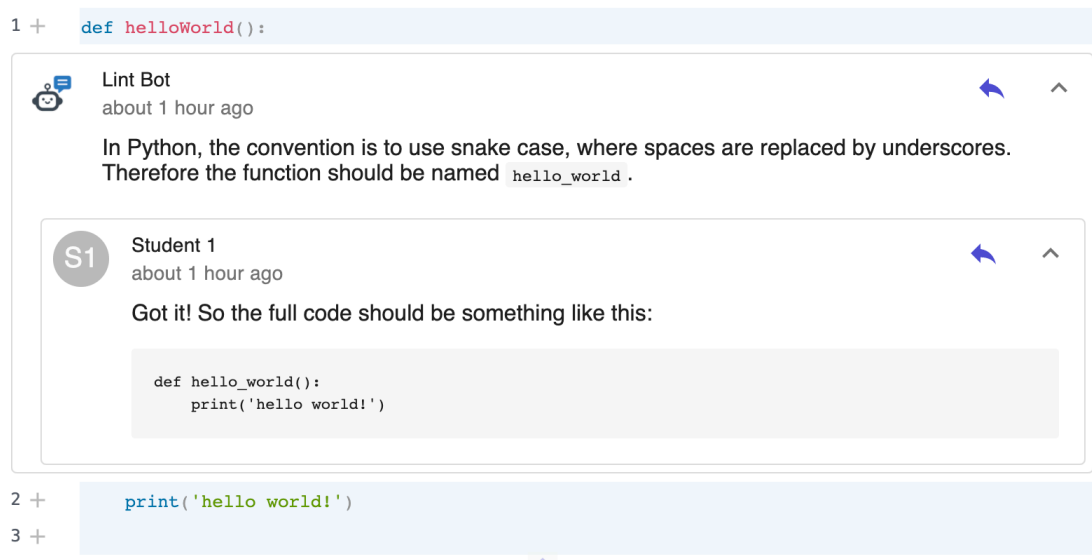


Figure 8.1: The new Code Review app allows students to reply to comments seeded by the educator, leading to thread-like exchanges. As depicted here, the educator can impersonate a chatbot when annotating the code snippet.

### 8.2.1 Chatbot Integration

Our objective in designing our chatbot integration was to provide educators with a way of easily configuring and impersonating chatbots in order to interact with students within a learning experience platform. To achieve this, we designed an interface whereby an educator

can easily create chatbots that can then be used to interact with students through the learning app in which the interface is embedded.

### Bot Users













Avatar	Name	Description	Actions
	Lint Bot	Lint Bot checks the quality of your code.	  
	Test Bot	Test Bot helps you test your code.	  
	Teaching Bot	Teaching Bot helps teachers explain content to their students.	  

Figure 8.2: Educators can create different bot identities that they can then impersonate when interacting with students.

To guide educators, we incorporated three best practices for the design and implementation of chatbots that are relevant to scenarios where chatbots are being impersonated by human users (Chaves et al., 2020; Ferman Guerra, 2018; Følstad & Brandtzaeg, 2020). Namely, we wanted to ensure that educators (i) provide the chatbot with an identity, (ii) explicitly state the chatbot's purpose, and (iii) can interact through messages including content that students might find pleasant, evocative, or playful. First, with respect to (i) and (ii), the interface provided a way to define a chatbot's identity, which was given by its name, an optional avatar, and an optional description, with both the name and description serving as a way to explicitly state the chatbot's purpose. Second, with respect to (iii), the messages sent by the educator under the chatbot's guise can be formatted using Markdown (Gruber et al., 2004), which supports embedding rich text and multimedia, making it possible to easily create engaging content.

We implemented this chatbot integration within Code Review. Figure 8.2 shows an instance of Code Review that has been seeded with three chatbot identities. Once a chatbot identity has been created, educators can impersonate it to provide feedback to students, as shown in Figure 8.3. Students can then reply back, leading to interactions similar to the one shown in Figure 8.1.

### 8.2.2 Lint Bot

As mentioned in Section 8.1, many social coding platforms support using chatbots to perform a variety of tasks (Wessel, de Souza, et al., 2018). One of these tasks is to verify that code

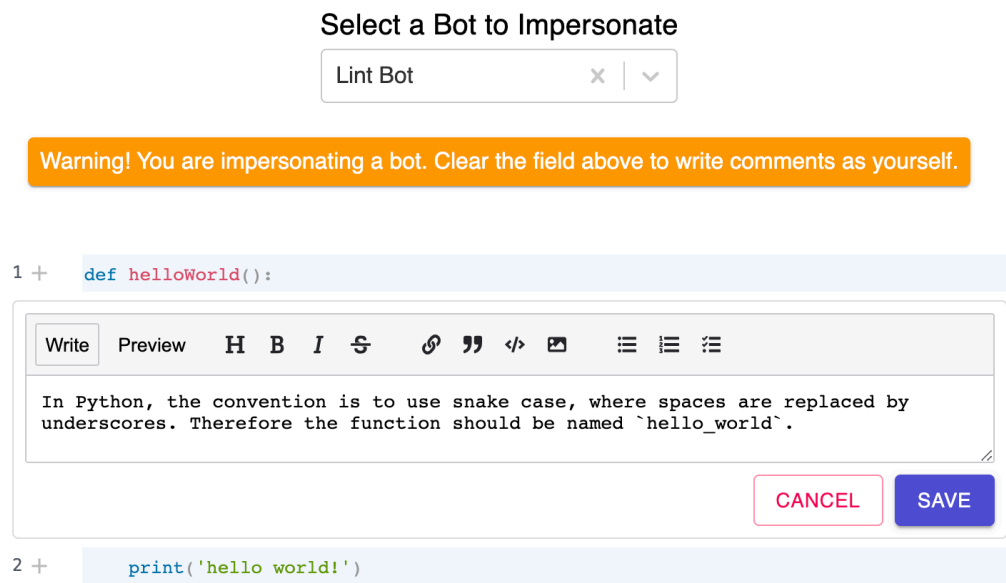


Figure 8.3: Educators can impersonate chatbots to seed a code snippet with comments or to provide feedback to students.

submitted to a repository is formatted according to the repository’s respective style and guidelines. The term *linting*—which we introduced in Chapter 5—refers to the use of static analysis tools (*linters*) to detect bugs and other issues (*lint*) in software programs (Johnson, 1978). Therefore, for our evaluation, we created a bot identity using the name *Lint Bot*. This identity would be used to explain code quality issues to students. As noted in Section 8.1, it is important to note that Lint Bot was simply an identity and did not have any agency or script to respond automatically to student comments. That is, for Lint Bot to interact with students, educators had to manually post comments and replies using Lint Bot’s identity. In our case, the instructor who participated in our evaluation impersonated Lint Bot to seed the comments in one of the treatment conditions (chatbot condition). Aside from the seed comment, there were no further interactions provided. Students assigned to this condition were unaware that the chatbot had been impersonated by the instructor.

## 8.3 Methodology

The purpose of our evaluation was to address one main research question, which is an extension of **RQ3**:

- What are the effects of using static chatbot comments on students’ experiences learning to identify code quality issues in JavaScript? (**RQ3c**)

We addressed this research question by evaluating three variations of our learning scenario.

More specifically, we tested the effect of impersonating chatbots when presenting concepts in software engineering education, conducting a between-subjects controlled experiment comprising one control and two treatments. In all conditions, subjects were presented with the learning scenario described in Section 8.3.1. The conditions differed only in the way we explained the linting errors illustrated by the example code snippets. To frame our evaluation, we focused on four aspects of the learning experience: (i) usability, (ii) engagement, (iii) short-term learning gains, and (iv) feedback. In this section, we explain the methodology of our evaluation in detail.

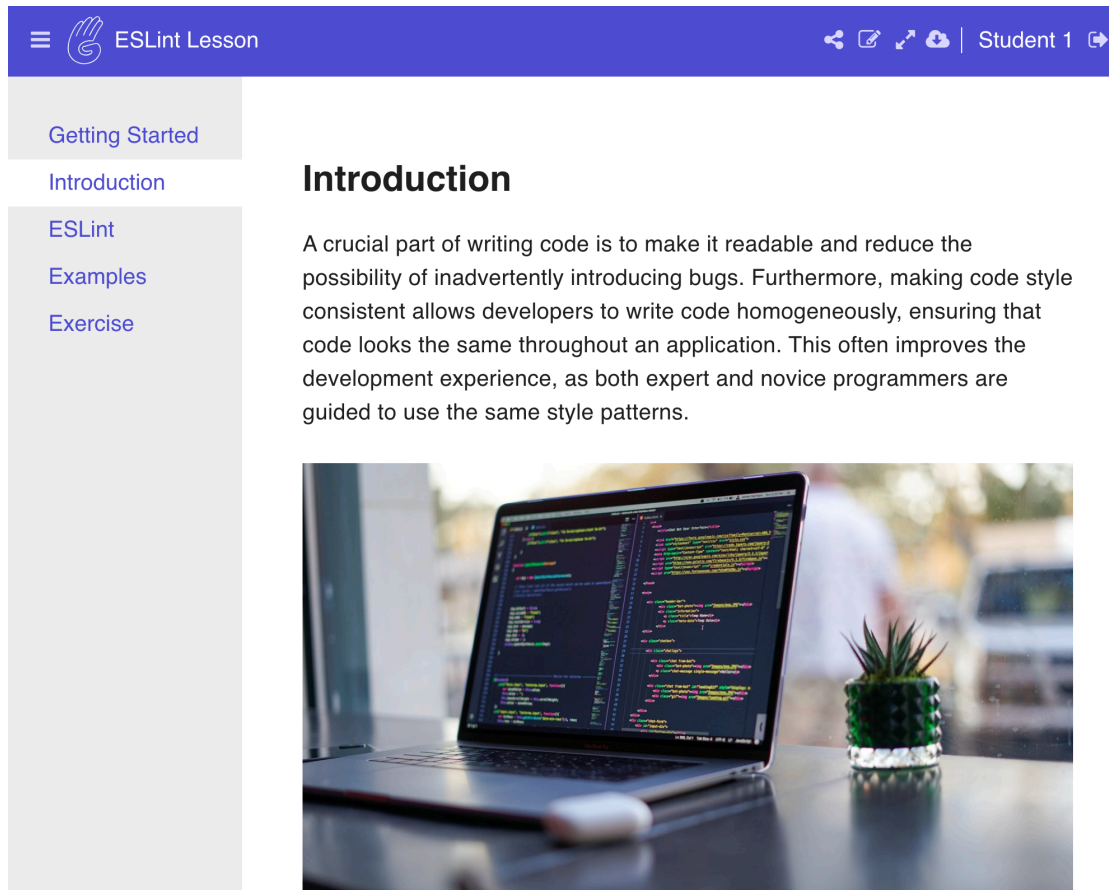


Figure 8.4: We prepared a lesson introducing students to the concept of code linting in JavaScript.

### 8.3.1 Pedagogical Scenario

Our goal was to evaluate Code Review and the effect of equipping it with a chatbot interaction in a real software engineering lesson. To do this, we created a lesson introducing the concept of code linting following a similar format to the one presented in Section 5.3.1. This lesson was structured in five phases. In the first phase, students were presented with a snippet of code. Students were explained how to use Code Review and asked to add comments to the parts of

the code that had potential issues. This exercise served as a pre-test to (i) gauge their level of engagement at the start of the lesson and (ii) see if they could find six issues that had been explicitly seeded in the code. In the second phase, students were introduced to the concept of code linting, while in the third phase, they were introduced to ESLint (Zakas, 2013). In the fourth phase, four specific code quality issues were presented and exemplified in detail using Code Review. This fourth phase was the only part of the scenario that varied between conditions (see Section 8.3.3 for more details). Finally, in the fifth phase, students were again presented with an exercise using Code Review. This exercise served as a post-test to (i) gauge their level of engagement at the end of the lesson and (ii) see if they could find seven issues present in the code snippet. Figure 8.4 depicts the learning scenario, highlighting the second phase of the lesson (*Introduction*).

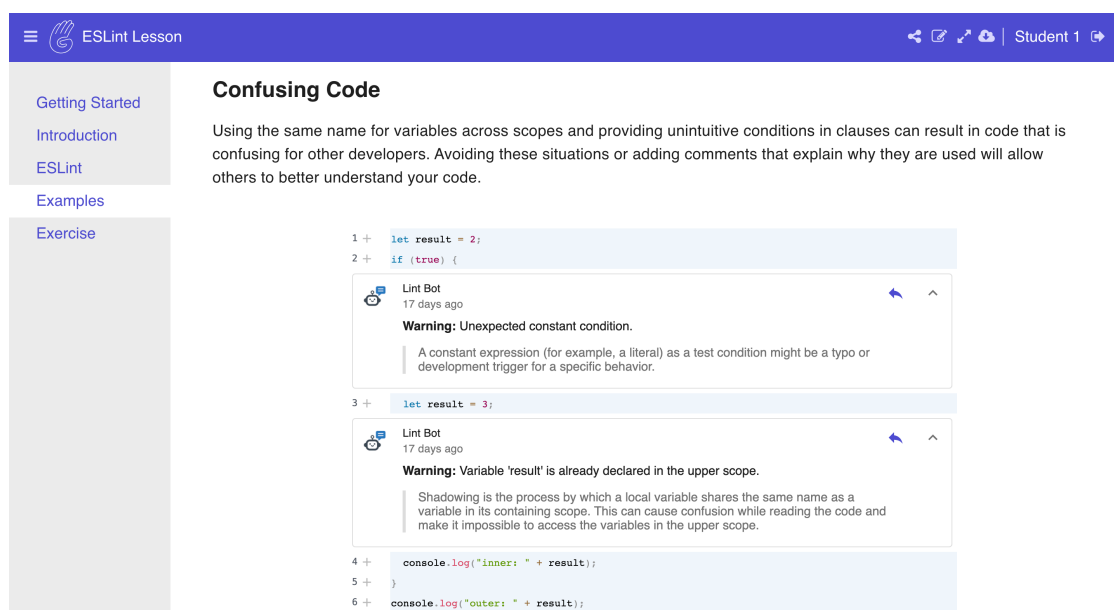


Figure 8.5: The *Examples* phase of the learning scenario varied across experimental conditions. This figure depicts the interface for the chatbot condition, showcasing an example code snippet with linting errors that have been highlighted by comments from an instructor impersonating *Lint Bot*.

### 8.3.2 Participants

We recruited 31 undergraduate software engineering students following a course on frontend web development at the School of Engineering and Architecture of Fribourg, Switzerland. Students consented to participate in this study. Responses from one participant were omitted after controlling for data collection errors, resulting in a total of 30 valid participants (2 female, 28 male), with a distribution of 9 participants in the control condition, 10 participants in the instructor condition, and 11 participants in the chatbot condition.

### 8.3.3 Procedure

The experimental part of our study was conducted in November 2021. As part of an in-class exercise, students were given 30 minutes to complete the learning scenario described in Section 8.3.1. The content of the lesson was identical for all students except for the fourth phase (*Examples*), which differed in the way the code quality issues were explained. In the control condition, the explanation was presented in text alongside—but not embedded within—Code Review (control condition). In the two treatments, we either provided the explanation within comments in Code Review (i) via a user representing the course instructor (instructor condition) or (ii) under the alias of Lint Bot (chatbot condition). All explanations were in English, based on feedback provided by the ESLint CLI (Zakas, 2013). Figure 8.5 depicts the *Examples* phase for the chatbot condition. Interactions with the lesson were recorded using Graasp’s implementation of the LA pipeline architecture described in Chapter 3. Upon completion of the lesson, students were directed to a post-questionnaire.

### 8.3.4 Instruments

To assess the impact of the treatment conditions of our controlled experiment, we operationalized three concepts—(i) usability, (ii) engagement, and (iii) learning gains—using the following instruments. To collect data on usability, we used the System Usability Scale (SUS) (Brooke, 1996).

For engagement and learning gains, we built bespoke instruments using the results of the pre- and post-tests. As explained in Section 8.3.1, the pre-test assessed if students could find six issues seeded in a code snippet, while the post-test assessed if they could find seven issues seeded in a different code snippet. Engagement was operationalized using the length of the comments added by students. The total length  $h_{l,t}$  of the set of comments  $C_{l,t}$  added by a given student  $l$  in test  $t$  can be calculated as the sum of the length of each comment in the set.

$$h_{l,t} = \sum_{c \in C_{l,t}} \text{len}(c)$$

The total length of the comments added by a given student in the pre-test ( $h_{l,\text{pre}}$ ) served as an indicator of how engaged the student was at the beginning of the activity, before being exposed to their respective condition. The total length of comments added in the post-test ( $h_{l,\text{post}}$ ) then served as an indicator of how engaged the student was at the end of the lesson, after being exposed to their respective condition. The engagement resulting from their exposure was then measured as the difference between the total length of comments left by the student in the post-test and the pre-test. We refer to this metric as the *engagement score* ( $e$ ).

$$e_l = h_{l,\text{post}} - h_{l,\text{pre}}$$



As in Chapter 5, learning gains were operationalized using the scores of the post-test and the pre-test, with a possible range from  $-100\%$  to  $100\%$ , inclusive. Finally, we also captured qualitative feedback through an open-ended question asking students to provide comments, suggestions, and general observations in short answer form.

### 8.3.5 Data Analysis

Data collected through the aforementioned instruments were analyzed using mixed methods. Quantitative data were analyzed using descriptive and inferential statistics. In terms of descriptive statistics, we report the sample mean ( $\bar{x}$ ), median ( $\tilde{x}$ ), and standard deviation ( $s_x$ ). In terms of inferential statistics, data on usability, engagement, and learning gains were all normally distributed and homoscedastic within each condition and were therefore compared using ANOVA. Qualitative data—in the form of open-ended comments—were analyzed by articulating emergent themes using line-by-line data coding (Charmaz, 2006).

## 8.4 Results

In this section, we present the main results of our evaluation. These results are also summarized in Table 8.1 and illustrated in Figure 8.6.

### 8.4.1 Usability

The mean SUS scores were  $\bar{x} = 82.250$  ( $\tilde{x} = 82.500$ ,  $s_x = 9.238$ ) in the instructor condition,  $\bar{x} = 75.682$  ( $\tilde{x} = 77.500$ ,  $s_x = 7.508$ ) in the chatbot condition, and  $\bar{x} = 84.167$  ( $\tilde{x} = 85.000$ ,  $s_x = 8.004$ ) in the control condition. All of these scores correspond to usability that can be described as *good* (Bangor, Kortum, et al., 2008). Furthermore, one-way ANOVA showed that there were no significant differences between conditions ( $F(2, 27) = 2.978$ ,  $p = 0.068$ ).

Table 8.1: Descriptive Statistics for Each Condition and ANOVA Results Across Conditions

Aspect	Condition			ANOVA $F$ ( $p$ -value)
	<i>Control</i> $\bar{x}$ ( $s_x$ )	<i>Chatbot</i> $\bar{x}$ ( $s_x$ )	<i>Instructor</i> $\bar{x}$ ( $s_x$ )	
<i>Usability</i>	84.167 (8.004)	75.682 (7.508)	82.250 (9.238)	2.978 (0.068)
<i>Engagement</i>	57.556 (56.593)	38.727 (106.593)	108.500 (97.803)	1.606 (0.219)
<i>Learning Gains</i>	0.071 (0.231)	0.184 (0.179)	0.145 (0.287)	0.577 (0.568)

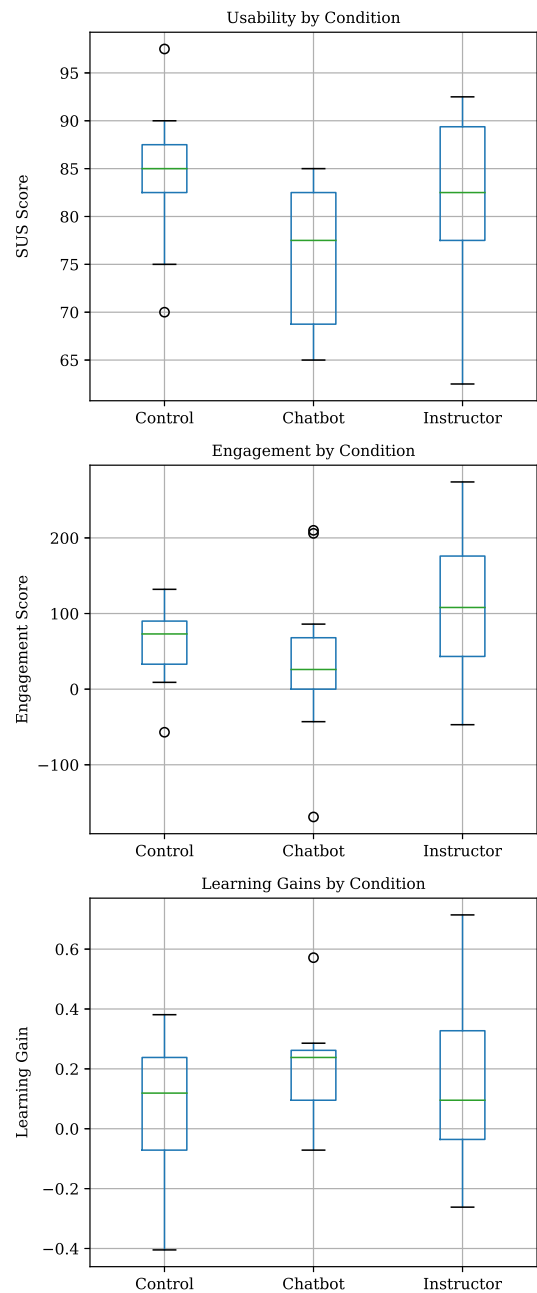


Figure 8.6: Box plots summarizing the descriptive statistics across conditions for the three aspects considered by our quantitative analysis: usability (top), engagement (middle), and learning gains (bottom).

### 8.4.2 Engagement

The mean engagement scores were  $\bar{x} = 108.500$  ( $\tilde{x} = 108.000$ ,  $s_x = 97.803$ ) in the instructor condition,  $\bar{x} = 38.727$  ( $\tilde{x} = 26.000$ ,  $s_x = 106.593$ ) in the chatbot condition, and  $\bar{x} = 57.556$  ( $\tilde{x} = 73.000$ ,  $s_x = 56.593$ ) in the control condition. These descriptive results suggest that engagement was higher in the instructor condition. However, one-way ANOVA showed that there were no significant differences between the conditions ( $F(2, 27) = 1.606$ ,  $p = 0.219$ ).

### 8.4.3 Learning Gains

The mean learning gains were  $\bar{x} = 0.145$  ( $\tilde{x} = 0.095$ ,  $s_x = 0.287$ ) in the instructor condition,  $\bar{x} = 0.184$  ( $\tilde{x} = 0.238$ ,  $s_x = 0.179$ ) in the chatbot condition, and  $\bar{x} = 0.071$  ( $\tilde{x} = 0.119$ ,  $s_x = 0.231$ ) in the control condition. These results show that all conditions led to—on average—positive learning gains for students. Nevertheless, when comparing the conditions, one-way ANOVA showed that there were no significant differences ( $F(2, 27) = 0.577$ ,  $p = 0.568$ ).

### 8.4.4 Qualitative Feedback

A total of 10 students provided open-ended feedback, four from the instructor condition, three from the chatbot condition, and three from the control condition. Two themes were presented by more than one student. First, two students (one in the instructor condition and one in the control condition) described the examples in the lesson as “repetitive”. Second, four students (one in the instructor condition, one in the chatbot condition, and two in the control condition) expressed the need for feedback after having completed the post-test. Specifically, one student expressed the following with respect to the need for feedback: “You explain the concept, yes, but the fact that we do an exercise without being able to see the answer key makes it all frustrating and not at all useful in learning.”<sup>1</sup> This quote illustrates how the second theme—which was the only one seen across all three conditions—was the one that students who provided open-ended comments were most vocal about.

## 8.5 Discussion and Implications

Our evaluation provided inconclusive yet promising results regarding the role chatbots could play in learning scenarios incorporating code review exercises. First, our quantitative analysis showed that, overall, usability was good in all conditions. Similarly, all conditions led to positive learning gains. These findings imply that the learning scenario in and of itself was both usable and effective. Nevertheless, there were no significant differences between the conditions for either usability or learning gains. Second, while descriptive results suggest that engagement was higher in the instructor condition, no significant differences were observed

<sup>1</sup>“On nous explique le truc oui, mais le fait qu’on aie [sic] un exercice sans pouvoir avoir un corrigé rend le tout frustrant et pas du tout utile dans l’apprentissage.” (Translated from French by the authors.)

between conditions. This lack of significant differences across conditions for all of the aspects considered could be due in part to our limited sample size. As illustrated in Figure 8.6, the presence of outliers within various condition/aspect combinations could have affected our ability to draw conclusions from our results. Future work aims to further explore our scenario with a larger group of students.

Moreover, our findings could suggest that the differences across conditions, which were limited to the three code snippets used in the *Examples* phase of the lesson, were not distinct enough and therefore resulted in a very similar learning experience. Indeed, the qualitative analysis we conducted sheds some light on this matter. First, it was reported that the exercises were repetitive. This perception of repetitiveness could have resulted in an adaptation effect, diminishing the impact of the treatments. Second, students expressed disappointment at the absence of feedback provided during the lesson. These remarks point to a possible explanation for the nonsignificant results across conditions. Namely, the mere use of identities (human or chatbot) to present static information adds very little to the learning experience, at least within the scope of our learning scenario. Instead, students expect this type of scenario to include feedback and interactivity.

These results have concrete implications for the work conducted in the rest of this thesis. First, the interface to create chatbot identities, which we added to Code Review, will be harnessed in future chapters. Second, the fact that usability, learning gains, and engagement were not impacted by the use of chatbots suggests that some of the interactivity required for these lessons could be provided by educational chatbots, possibly reducing the workload for educators. The use of educational chatbots could also open up new opportunities to offer these blended and remote learning scenarios to larger cohorts of students without diminishing their educational value. Finally, the qualitative feedback received informed our decision to (i) improve the learning scenario to include a phase in which the solutions to the exercise are presented and (ii) equip the chatbots with mechanisms through which they could interact automatically with learners. While educators might not be able to cope with giving real-time feedback to large numbers of learners, chatbots following rule-based scripts could serve as a first layer of interaction. A promising next step for our research would be to build on these results and assess the role that interactive chatbots could play within exercises simulating code reviews. This interactivity is what we will introduce in the next two chapters.

### 8.6 Conclusion

In this chapter, we explored the role chatbots could play in supporting a lesson on software engineering. Our approach was to first equip Code Review with an interface to enable instructors to impersonate chatbots when presenting content or interacting with their students. Code Review was integrated into a variation of the lesson on code linting presented in Chapter 5 and used to evaluate three ways of explaining linting errors presented via example code snippets. This evaluation was conducted through a controlled in-class experiment with 30

undergraduate software engineering students.

Findings from our evaluation suggest that there are no significant differences between the three conditions we explored—explaining concepts (i) via a user representing the course instructor, (ii) via a chatbot alias, or (iii) in the accompanying text—with respect to perceived usability, engagement, and learning gains. However, suggestions obtained through open-ended responses provide key insights into the possible need for interactivity and feedback in order to better support students in this type of learning scenario. We aim to build on these findings to equip our chatbot identities with interactivity and evaluate them in future work.



## 9 Rule-Based Scripts

IN this chapter, we report on the integration of a rule-based chatbot into an information technology course. Building on the Wizard of Oz chatbot interface presented in the previous chapter, we start by describing how we equipped these chatbots with automated scripts. We then conducted a controlled experiment whereby half of the students were able to engage with an interactive chatbot when taking part in the course's lab sessions, while the other half completed the sessions without the chatbot interaction. Our results show that while the chatbot did not lead to differences in short-term learning gains or perceived relevance of the lesson, it did improve usability. These findings suggest that educational chatbots powered by short, simple, interactive scripts could have a positive impact on students' experiences using blended learning technologies such as code review notebooks. In turn, this improvement in usability could affect other dimensions, such as completion rate, self-regulation, and motivation—as suggested in the literature—and could be pertinent to educators looking to integrate chatbots into their practice.

By investigating the effects of incorporating rule-based chatbots into a software engineering lesson, this chapter partially addresses **RQ3**:

How can we equip educational chatbots with effective interaction strategies?

The content of this chapter was partially presented in the following publications:

1. Farah, J. C., Spaenlehauer, B., Bergram, K., Holzer, A., & Gillet, D. (2022). Challenges and Opportunities in Integrating Interactive Chatbots into Code Review Exercises: A Pilot Case Study. *EDULEARN22 Proceedings*, 3816–3825. <https://doi.org/10.21125/edulearn.2022.0932>
2. Farah, J. C., Spaenlehauer, B., Ingram, S., Purohit, A. K., Holzer, A., & Gillet, D. (2023). Harnessing Rule-Based Chatbots to Support Teaching Python Programming Best Practices [In Submission]

### 9.1 Introduction

In the previous chapter, we described how we improved Code Review to support chatbot identities and how we conducted a study that used these chatbot identities as a way to illustrate the code review process to students. Following the Wizard of Oz interaction strategy, instructors could impersonate chatbots to annotate code snippets with sample comments, but these annotations were static—meaning that students could not engage in real-time conversations with the chatbot—and required manual intervention from the instructor. In this chapter, we present how we enhanced the static interface by equipping it with functionalities to support chatbots following automated interactive scripts.

Using these new functionalities, we then configured *PEP-8 Bot*, a chatbot designed to support a series of lessons on Python programming best practices. PEP-8 Bot followed predefined scripts intended to show students that a particular code style guideline was relevant in practice. The impact of PEP-8 Bot on these lessons was evaluated through a controlled experiment conducted within a university-level information technology course. Our study aimed to shed light on how rule-based chatbots could be harnessed to support students learn Python programming best practices as defined by the PEP-8 standard (van Rossum et al., 2001). The results of our case study contribute to the growing body of research on educational chatbots and may be relevant to practitioners looking to integrate these chatbots into their courses.

Findings from this study provide a fourth strategy that could be used to configure how educational chatbots interact with learners, making up a part of the fifth contribution in this thesis (C5). Specifically, having not found any significant differences in the case where chatbots were used to present static content, our aim was to assess how equipping them with interactivity could have an impact on the learning experience. As such, this study continues our exploration of the effects of incorporating chatbots into a code review exercise aimed at educational contexts.

This chapter is structured as follows. In Section 9.2, we present how we equipped the chatbots embedded in Code Review with rule-based scripts. We then outline the methodology used for our evaluation in Section 9.3 and report on the results in Section 9.4. These results, as well as their implications in the context of this thesis, are discussed in Section 9.5. Finally, we conclude in Section 9.6.

### 9.2 Design

To enable interactive dialog within Code Review, we enhanced the chatbot interface so that conversations between learners and chatbots could follow rule-based scripts. These scripts then define how a chatbot will process student input. In this section, we describe the design of our rule-based engine.



### 9.2.1 Automated Chatbot Responses

We added an *Auto Bot Settings* tab to the educator interface of the Code Review app. Within this tab, educators can activate both automatic replies (which will enable the rule-based script) as well as automatic message seeding, which will add a seed message to the lines where the chatbot is added in order to invite users to interact with the chatbot. Finally, educators can activate human intervention requests, in which learners can request that a human user intervene in the conversation. These settings are shown in Figure 9.1.

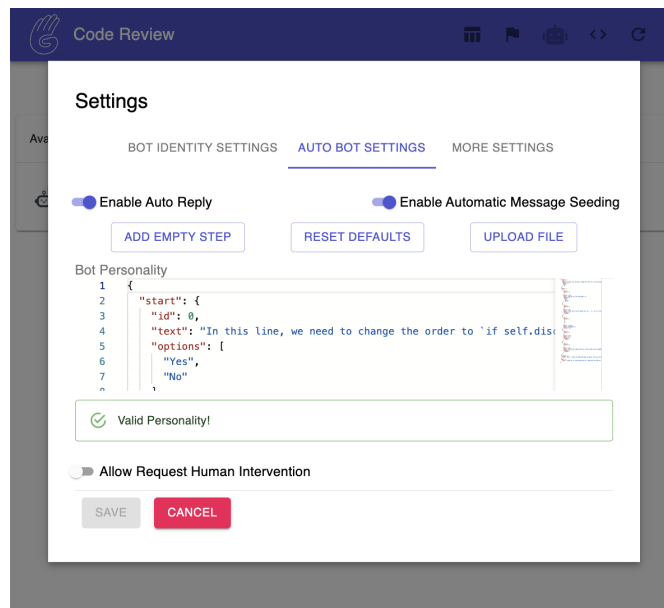


Figure 9.1: Through the *Auto Bot Setting* feature, educators can upload a script defining the rules that the chatbot would follow or edit these rules directly using a built-in editor.

Once activated, automatic replies enable a built-in editor featuring a sample script that educators can try out before defining their own scripts. To facilitate the creation and dissemination of these scripts, we added the possibility for educators and developers to edit the script outside of Graasp and upload the configuration directly. Educators can also create and edit configurations using the built-in editor. As shown in Figure 9.1—via the *Valid Personality* message—this editor includes a verification mechanism to ensure that the script conforms to the format compatible with the dialog engine, which we will present in the following section.

### 9.2.2 Dialog Engine

To support executing rule-based scripts, we equipped Code Review with a browser-based dialog engine that can interpret simple JSON configuration files following a predefined structure (see Figure 9.7, left). For clarity, we refer to chatbot messages as *comments* and learner messages as *responses* or *replies*. Each script starts with a comment that serves as a seed,

## Chapter 9. Rule-Based Scripts

which is a comment used to elicit an interaction with the learner. Depending on the learner's response, different conversational paths can be taken. The dialog engine matches a learner's response to the appropriate next step in the conversational path using regular expressions. For convenience, a subset of the responses that the chatbot would know how to interpret are provided as quick-reply buttons (e.g., the *Yes* and *No* buttons shown in Figure 9.4). There is also a fallback comment that is used when no comment could be matched with the response provided by the learner. Finally, to mark the end of an interaction, the chatbot outputs a comment that does not provide any quick-reply options. If the learner responds to this last comment, a special end step is used, which informs the learner that the chatbot has nothing more to say and provides the learner with a button to restart the interaction.

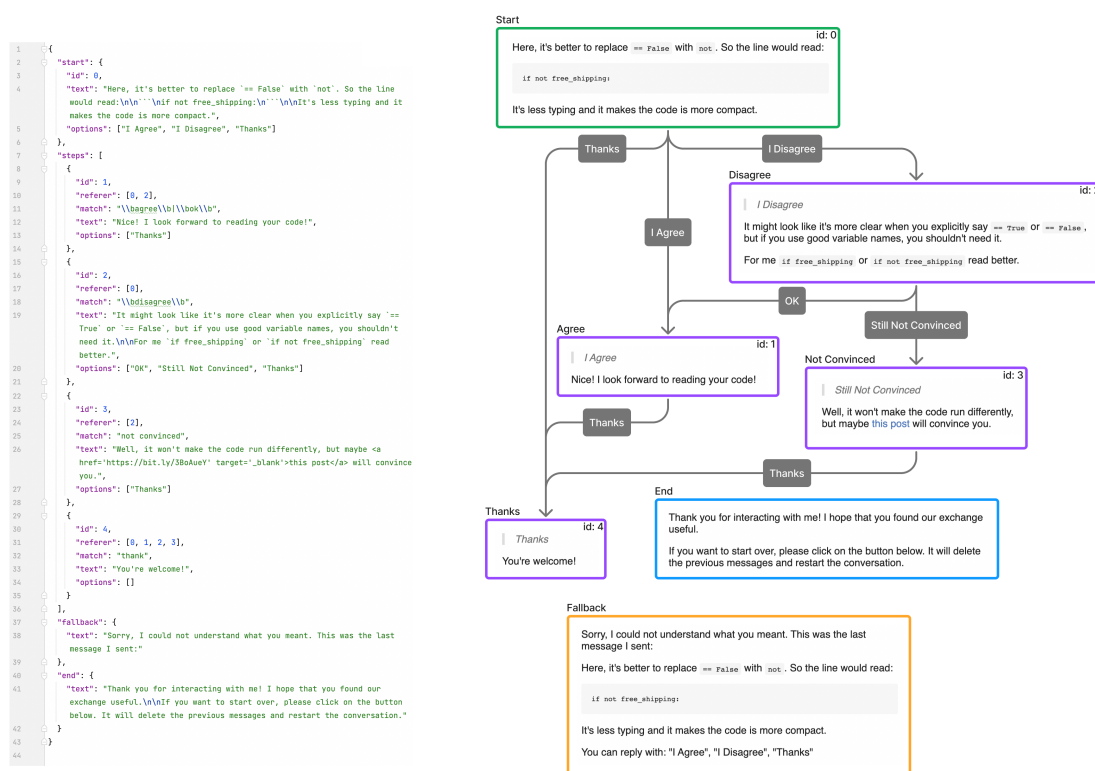


Figure 9.2: Our chatbot interaction script was configured using the JSON format (left) and supported a rule-based conversational flow comprising several potential pathways (right).

### 9.2.3 Selective Audience

A final feature that was developed was the possibility to select which learners were exposed to a given chatbot. As shown in Figure 9.3, this selectivity is achieved through a setting that allows the educator to manually select the learners who will (or will not) be exposed to the chatbot. Educators can also upload a file containing a list of these learners' IDs. For our empirical evaluation, this allowed us to divide participants into a control and treatment group, where

the chatbot was visible only to students in the treatment group. Nonetheless, there are other use cases for this feature, such as personalizing the chatbots for individual learners or cohorts of learners.

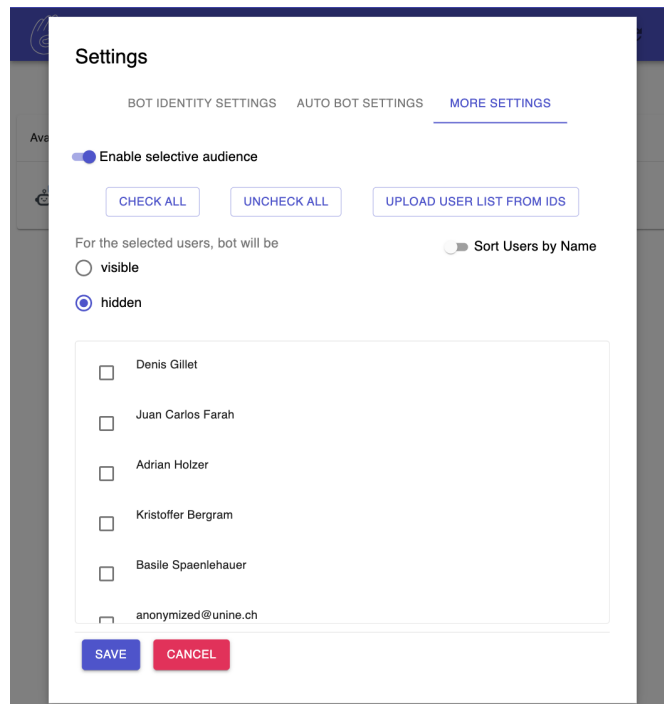


Figure 9.3: Educators could also select to show/hide the chatbot to/from a particular set of learners.

### 9.3 Methodology

Our evaluation aimed to address the following research question, which is an extension of RQ3:

- What are the effects of a rule-based chatbot designed to support Python programming lessons on students' learning experiences? (**RQ3d**)

We addressed this research question by conducting a between-subjects controlled experiment comprising one control and one treatment. In both conditions, students were presented with the same pedagogical scenario. The conditions differed only in the way we explained the code style issues illustrated by the example code snippets. To frame our evaluation, we focused specifically on four aspects of the learning experience: (i) learning gains achieved, (ii) perceived relevance of the material, (iii) usability of the lesson, and (iv) feedback. In this section, we explain the methodology of our evaluation in detail.

### 9.3.1 Pilot

Before conducting our main evaluation, we conducted a week-long pilot in a computational thinking course aimed at students pursuing a master's degree in the Faculty of Economics and Business at the University of Neuchâtel in Switzerland. A total of 28 students registered for the course and 27 completed it. The pilot study was structured as a within-subjects controlled study, which aimed to identify possible challenges that could arise during the main evaluation.

The course was conducted in English and took place over one week. Each day, students attended an in-class lecture in which they were introduced to a Python programming concept. On the same day, they had a lab covering the topics studied in the lecture, as well as an exercise related to Python code style guidelines. Students could complete these labs in person or remotely. Participants completed a total of five labs. The first two labs did not include a chatbot, while the following three labs did.

While we saw a diminishing trend in interactions with the chatbot, participants rated the lesson positively. For each lab featuring the chatbot, students gave a rating based on a question asking them to rate the usefulness of the chatbot on a scale of 1 to 7, with 1 being *Not Useful At All* and 7 being *Very Useful*. The overall mean rating was  $\bar{x} = 6.24$  ( $\bar{x} = 6.00$ ,  $s_x = 0.73$ ,  $n = 17$ ), suggesting that the chatbot was useful in carrying out its task.

### 9.3.2 Pedagogical Scenario

The main evaluation took place within the five-week Python programming component of a 14-week information technology course tailored for students completing a bachelor's in economics and business at the University of Neuchâtel. A total of 97 students were enrolled in the course. Before the beginning of the course, students were randomly assigned to the treatment (chatbot) or control (no chatbot) group. Each week, students took part in an in-class lecture. The same week, they were able to attend a lab session in which the topics covered in the lecture were addressed. Participation, however, was not mandatory. The course was conducted in French and all the material—including the chatbot scripts—was adapted to French. For convenience, the figures shown in this chapter are the English versions used in the pilot.

Table 9.1: Weekly lecture topics included in the Python programming component of the course alongside their corresponding lab session and code style guidelines covered therein.

Week	Lecture	Lab	Code Style Guidelines
1	Conditions	Conditions	Pre-Test
2	Loops	Loops	Indentation, Whitespace, Constants
3	Lists	Lists	Comparisons, Negating, Comparing Booleans
4	Functions	Functions	Max Length, Function Names, Descriptive Names
5	—	Review	Post-Test

Each lab session was conducted in French and included an exercise on Python code style guidelines based on the PEP-8 standard. As in our previous evaluations, these exercises followed the *Fixer Upper* pedagogical pattern both for explanation and evaluation (Bergin, 2000). That is, students were presented with code snippets that included code styling violations and were shown how to correct them (explanation) or asked to identify the issues present (evaluation). The first lab introduced students to the exercises and included a short activity that served as a pre-test. The second, third, and fourth labs each included explanations covering code layout, coding standards, and naming standards, respectively. In the final session, students completed a second activity that served as a post-test. Sessions were supported by Graasp and Code Review.

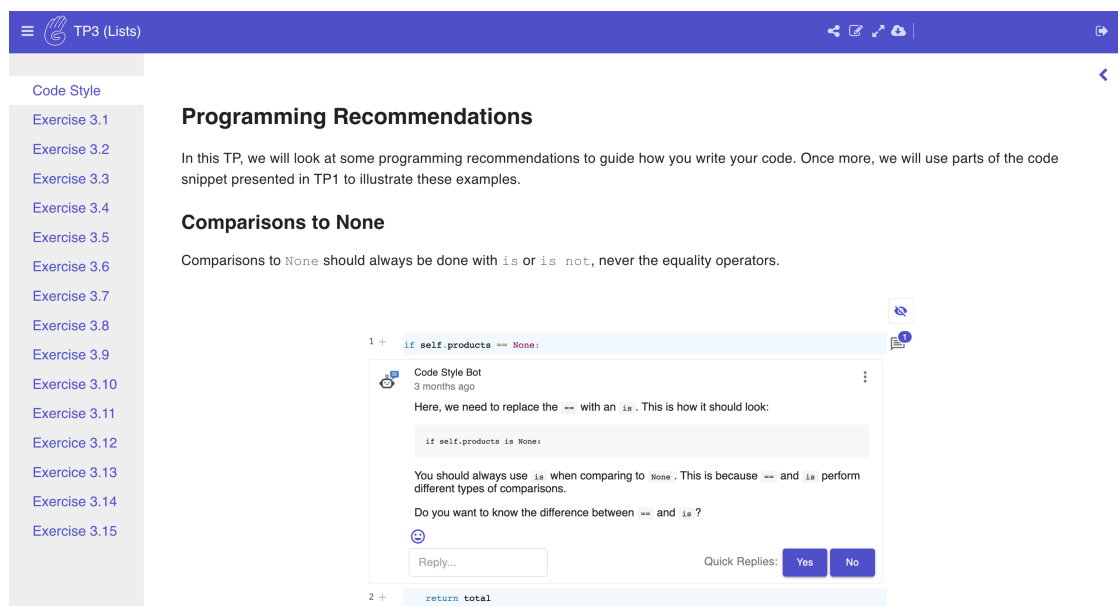


Figure 9.4: Our chatbot was integrated into a code review application that was embedded in a code review notebook aimed at teaching Python programming.

## Chatbot

For this case study, we equipped Code Review with *PEP-8 Bot* (named *Code Style Bot* in the pilot, which was conducted in English). This chatbot was used to annotate the lines of code that contained potential code style issues within a series of Python code snippets used in Labs 2–5. The chatbot was configured to engage students by asking them if they understood and agreed with the logic behind the code style issue at hand, providing explanations of Python programming best practices, and motivating the reasons behind those best practices. An example of the chatbot embedded in the code review notebook is shown in Figure 9.4

Furthermore, building on findings from Chapters 6 and 7, we included interactions featuring emoji and animated gifs, taking advantage of the Markdown support provided by Code Review.

## Chapter 9. Rule-Based Scripts

This allowed our chatbot to express—among other emotions—humor and surprise, as shown in Figure 9.5.

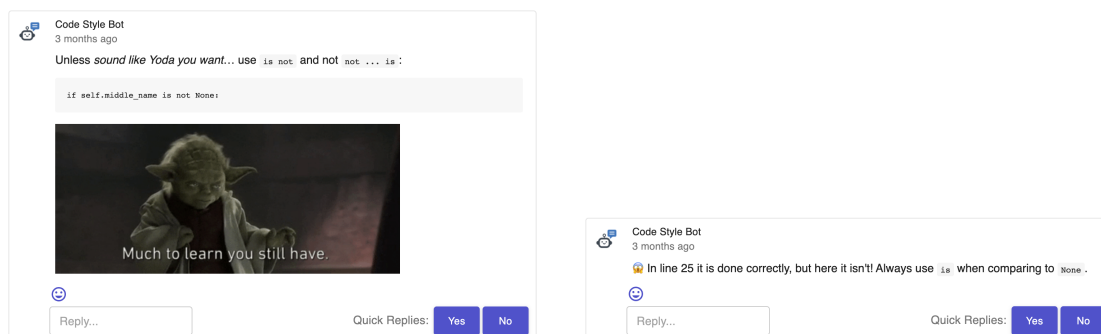


Figure 9.5: Some of the chatbot's comments featured animated gifs (left) and emojis (right) to elicit humor and express surprise, respectively.

### 9.3.3 Procedure

At the beginning of each lab session, students were asked to complete a short code style exercise. Each exercise was meant to take approximately five minutes to complete. Lab 1 included the pre-test and was identical for all students. In Labs 2–4, for all students, the code style exercise covered three guidelines (indentation, whitespace, and constants). For each guideline, students were presented with a short explanation followed by a code snippet containing an issue (incorrect snippet) and a code snippet with a correction of the issue (correct snippet). In the incorrect snippet, on the line containing the issue, students in the treatment group were also shown a comment from PEP-8 Bot that provided a further explanation and prompted the student to start a dialog about the relevance of the guideline. Students in the control condition did not see this comment.

### 9.3.4 Participants

There were 97 students enrolled in the course (44 female, 53 male). A total of 89 students accessed at least one of the exercise sessions and 65 students accessed all the sessions.

### 9.3.5 Instruments

We operationalized the four aspects of the learning experience as follows. Following the same strategy as our previous case studies, learning gains were calculated by taking the difference between students' scores on the pre-test and the post-test. This yielded a learning gain that could range from -100% to 100%. Relevance was measured using a seven-point Likert scale. Students were asked to rate how useful they found each individual guideline from 1 (*not useful at all*) to 7 (*very useful*). After the final exercise, using the same scale, they were asked to

rate the overall usefulness of the code style exercises as a whole. Usability was captured with the User Experience Questionnaire (UEQ), a standard instrument that measures usability across six dimensions (Laugwitz et al., 2008). Note that, unlike the UEQ-S—which we used in Chapter 5 to evaluate Code Review—the UEQ is the full version, comprising 26 items covering six factors: (i) attractiveness, (ii) perspicuity, (iii) efficiency, (iv) dependability, (v) stimulation, and (vi) novelty. Finally, feedback was assessed using the following question: *Do you have any suggestions or comments on the parts of the labs that dealt with code style guidelines?*

### 9.3.6 Data Analysis

We analyzed quantitative data using descriptive and inferential statistics, reporting sample means ( $\bar{x}$ ), medians ( $\tilde{x}$ ), and standard deviations ( $s_x$ ), as well as results from two-sample  $t$ -tests, where applicable. Responses to the UEQ were further analyzed using its toolkit, which includes a comparison against benchmark data covering 21,175 people over 468 studies in various domains (e.g., business, e-commerce, social networks). Qualitative feedback was analyzed using line-by-line data coding (Charmaz, 2006).

## 9.4 Results

In this section, we highlight the main results with respect to each aspect of our study.

### 9.4.1 Learning Gains

A total of 25 students (10 control, 15 treatment) completed the post-test required to calculate learning gains. In both groups, as shown in Figure 9.6, learning gains were positive. The students in the control group achieved a mean learning gain of 36.0% ( $s_x = 31.3\%$ ), while those in the treatment group achieved a mean learning gain of 36.7% ( $s_x = 30.6\%$ ). As expected, the results of a two-sample  $t$ -test did not yield significant results ( $p = 0.958$ ).

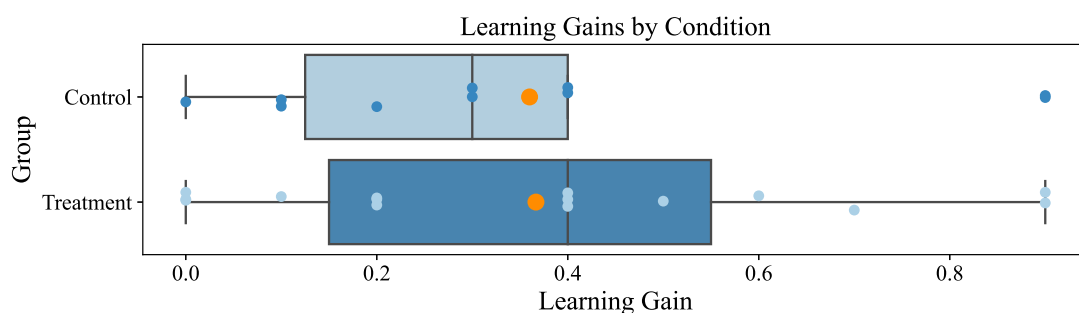


Figure 9.6: Learning gains were positive for both groups, but there were no significant differences across conditions.

### 9.4.2 Perceived Relevance

Regarding perceived relevance, 34 students (17 treatment, 17 control) provided a total of 199 ratings distributed across the nine different guidelines, while 14 students (5 control, 9 treatment) rated the overall usefulness of the code style exercises. As shown in Figure 9.7, ratings were on average positive, both overall and across individual guidelines. However, the two-sample  $t$ -tests did not yield significant differences across the groups. Furthermore, the number of students who provided the ratings decreased over time.

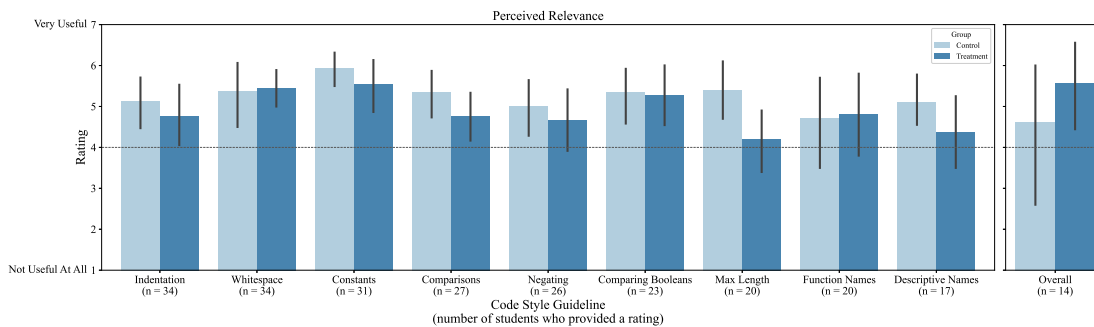


Figure 9.7: On average, students rated how relevant they found each guideline and the overall relevance of the code style exercises positively (above the median rating of 4), but there were no significant differences across groups.

### 9.4.3 Usability

A total of 27 students (14 control, 13 treatment) completed the UEQ. On average, students in the control group provided negative ratings for four of the six dimensions, while students in the treatment group provided positive ratings across all dimensions (see Figure 9.8). This difference was more pronounced in the efficiency ( $p = 0.0127$ ), dependability ( $p = 0.0565$ ), and perspicuity ( $p = 0.0807$ ) dimensions. However, compared to benchmark data, both groups provided ratings in the range of the 25% worst results for most dimensions.

### 9.4.4 Feedback

A total of 10 students (5 control, 5 treatment) provided qualitative feedback in the form of short open-ended responses. Four themes emerged in the responses: (i) *more support*, (ii) *more exercises*, (iii) *useful*, and (iv) *useless*. First, *more support* was requested by four students (one control, three treatment), who suggested that instructors help clarify doubts and provide more information on the exercises. These requests are exemplified by a comment from a student in the treatment condition, who underlined that a “human touch would be nice” when referring to the lesson. Second, two students in the control group requested *more exercises* as a way to better assimilate the material. Third, two students in the treatment condition expressed that the exercises were *useful*, with one student providing the following comment:



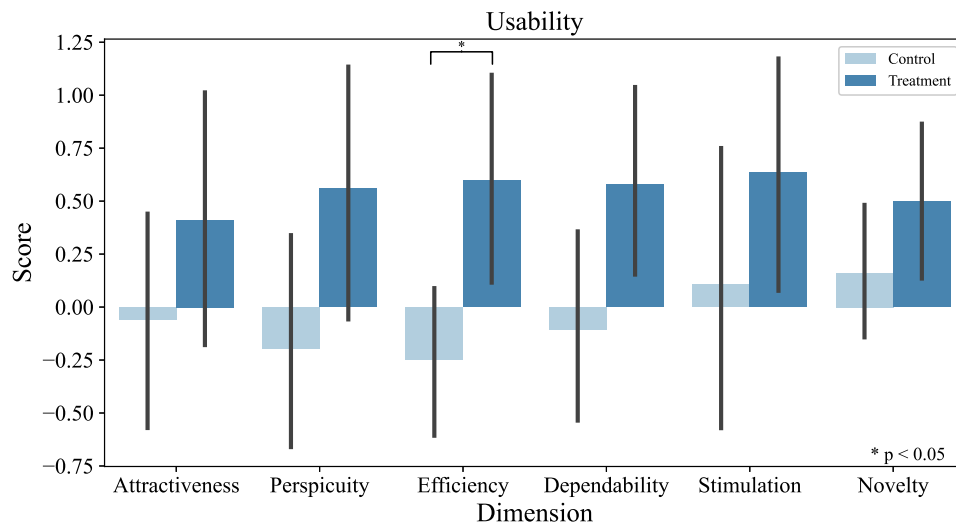


Figure 9.8: User Experience Questionnaire (UEQ) ratings were consistently higher for the treatment group, especially for the efficiency ( $p = 0.0127$ ), dependability ( $p = 0.0565$ ), and perspicuity ( $p = 0.0807$ ) dimensions.

“Guidelines that addressed code style that made code more readable (layout, balance between upper and lower case, convention in writing style) were very helpful. The little touches of humor in [Lab 4] made the [lab] much more interesting. In general, the practical guidelines of the code allowed the course to be more complete.”

Finally, two students in the control group referred to the exercise as *useless*, with one explicitly saying that they were “quite useless and boring compared to normal lab exercises”.

## 9.5 Discussion and Implications

The results of our evaluation show that while our chatbot integration did not affect learning gains or the perceived relevance of the material, it did have an impact on the usability of the lesson. On the one hand, results regarding learning gains could be explained by the fact that the chatbot interaction was primarily designed to reiterate the explanation that was already present in the text and persuade—if needed—the student that the guideline in question was useful in practice. On the other hand, for the same design reason, we would have expected students who were exposed to the chatbot to perceive the code style guidelines as more relevant than students who did not hold those interactions. However, as evidenced by the decreasing number of students who provided ratings as the course progressed, this result could have been influenced by a diminishing novelty effect that curbed student interest in the code style exercises.

As in our previous Wizard of Oz study (see Chapter 8), the lack of significant differences in learning gains between conditions could be interpreted positively. That is, as suggested by Hobert (2020), educational chatbots can serve to support learners when teaching staff are not available, or in cases with large numbers of students. Given that learning is not impacted, educational chatbots could serve as an additional layer of interaction for learners seeking more information.

The differences in usability, however, were evident—albeit not always significant—across all dimensions of the UEQ, and are very promising. Results from the UEQ suggest that incorporating the chatbot into the interface improved student perception of the usability of the lesson, especially in terms of efficiency, dependability, and perspicuity. This improvement could have been mediated by the fact that including the chatbot added interactivity to a lesson that was otherwise primarily explanatory.

Improvements in usability that are not accompanied by improvements in learning gains have been observed in the literature. As discussed by Davids et al. (2014), this lack of correlation could be due to the type of learners that are participating in the learning activity or the interface that is being optimized. In the case of Davids et al., their study was conducted with practicing clinicians who the authors describe as possibly being highly motivated and therefore less affected by the usability improvements the authors were testing. In our case, the fact that the exercises were not mandatory might have led to a selection bias, where the most motivated students took part in code style exercises included as part of their lab work, possibly leading to a similar effect as the one observed by Davids et al. (2014). Nevertheless, an improvement in perceived usability can have an impact on other dimensions, such as task completion rate (Kanuka et al., 1999), self-regulation (Liaw et al., 2013), and motivation (Zaharias et al., 2009).

Further insights were provided by the qualitative feedback. Positive comments regarding the *usefulness* of the exercises were present exclusively in the treatment condition, while negative comments about the *uselessness* of the exercise were present exclusively in the control condition. This contrast suggests that including the educational chatbot made the exercises more meaningful, possibly shedding some light on why perceived usability was higher in the treatment condition. Nonetheless, the fact that more support was requested by four students, including three in the treatment condition, indicates that even with the inclusion of a chatbot, students still require a “human touch” in the learning process.

These results also have concrete implications for the work conducted in the remainder of this thesis. First, the improvement in usability provides a promising direction for understanding how educational chatbots could have a positive impact on the learning experience. The fact that this difference was not present in the previous Wizard of Oz study suggests that interactivity could be a key element in achieving this improvement. Second, the relatively low overall participation in the code style exercises informed a decision taken for the ensuing empirical study, where we incentivized student involvement by providing extra credit for

participation in the exercise. Third, the qualitative feedback received for the lesson, which was predominantly positive in the treatment group, emphasizes how even with an educational chatbot interface, learners place emphasis on the interactions they have with instructors, as underlined by the request for a more “human touch”. Finally, the appreciation by one student of the humorous interactions—albeit anecdotal—is an encouraging confirmation that humor can have a positive impact on the learning experience, even when expressed through a chatbot.

## **9.6 Conclusion**

In this chapter, we presented results from an empirical case study assessing the effects of integrating educational chatbots into blended learning scenarios aimed at teaching Python programming best practices. The findings of our controlled experiment show that there were no significant differences in the learning gains achieved and the perceived relevance of the lessons between students who completed the exercises alone and those who completed them with support from the chatbot. However, students who had access to the chatbot rated the usability of the lesson more positively, particularly in the efficiency, dependability, and perspicuity dimensions of the UEQ. This improved user experience could motivate the integration of chatbots into the type of lessons used in this study.

It is also important to note that this study has some limitations worth considering. First, given that the exercises were not mandatory, there could have been some selection bias in our sample, as possibly only the most motivated students interacted with the lesson. Second, while the rule-based scripts ensured that student interaction with the chatbot was on topic and pertinent to the lessons, the limited scope of these exchanges could have diminished how natural they appeared to the student and, therefore, discouraged students from interacting with the chatbot. These limitations could be addressed by (i) ensuring that all students are exposed to the exercises and (ii) equipping the chatbot with a generative language model. We will address these limitations and explore possible improvements through the use of LLMs in the following chapter.



## 10 Large Language Models

THE recent rise in both popularity and performance of large language models (LLMs) has generated considerable interest with respect to their applicability to educational contexts. However, the integration of these technologies into large-scale learning systems is faced with both technological and pedagogical challenges, such as configuring the systems that interface with these models, ensuring that conversations stay on topic, and providing insight into the learning experience. In this chapter, we present the design of a web-based architecture and a chatbot configuration model that aim to mitigate these challenges. Our architecture serves as a blueprint for integrating chatbots powered by LLMs into learning experience platforms (LXPs), while our configuration model can serve as a guide for educators looking to incorporate these chatbots into their practice. To evaluate our architecture, we created Code Capsule, an app combining the functionalities of Code and Code Review, and equipped it with an educational chatbot powered by GPT-3 that was configured according to our model. We then conducted a controlled experiment with 26 software engineering students. Half of the students were exposed to a version of the application equipped with an educational chatbot, while the other half completed the same lesson without the chatbot. While the results of our quantitative analysis did not identify significant differences between conditions, qualitative insights suggest that learners appreciated the chatbot. These results provide a first validation for our model and serve as a starting point to optimize prompt engineering strategies for integrating large language models into pedagogical scenarios.

By investigating the effects of integrating LLM-powered chatbots into a software engineering lesson and proposing guidelines for this integration, this chapter partially addresses **RQ3**:

How can we equip educational chatbots with effective interaction strategies?

The content of this chapter was partially presented in the following publication:

1. Farah, J. C., Ingram, S., Spaenlehauer, B., Lasne, F. K.-L., & Gillet, D. (2023). Prompting Large Language Models to Power Educational Chatbots [In Submission]

### 10.1 Introduction

In this chapter, we explore a final strategy for designing the interactions that educational chatbots can have with learners. This strategy builds on the findings presented in Chapter 9 and focuses on enhancing the naturalness of these interactions while keeping the configuration process manageable for educators. As we saw in the previous chapter, rule-based chatbots can enhance the interactivity provided by lessons hosted on LXPs. Nevertheless, learners still noted the need for a *human touch* to provide more support during the lessons. Furthermore, configuring these rule-based chatbots in order to avoid repetitive content requires complex scripts. In light of these limitations, researchers have advocated for the use of LLMs to support more natural conversation and overcome the limitations of rule-based systems or systems based on limited training data (D. Song et al., 2017). However, several challenges limit the integration of LLM-powered chatbots into educational contexts. These challenges include—among others—providing the appropriate user interfaces and configuring the LLMs to ensure that the generated text is aligned with the pedagogical scenario in which the educational chatbots are deployed (Kasneci et al., 2023).

To help address these challenges, we first designed a technical architecture that can guide developers looking to integrate these models into LXPs. We then propose a configuration model that outlines a strategy that educators can follow when configuring chatbots powered by LLMs for use in digital education. Given the rising interest in the applications of powerful NLP technologies, such as ChatGPT, our study is timely and relevant to both research and practice. Our architecture serves as a practical blueprint for developers, while our configuration model aims to simplify the complex process that educators must go through when defining how they want a chatbot to interact with learners. This model can also serve developers in education looking to integrate chatbots into digital education platforms. By defining a common configuration language that bridges technological and pedagogical jargon, our approach aims to harness the power of LLMs while ensuring that the interactions are relevant to the scenarios in which these educational chatbots are deployed.

Our architecture and model were implemented on Graasp and their applicability to software engineering education was validated through a controlled experiment with 26 university students. This initial validation puts forth a final strategy for how chatbots can interact with learners when integrated into digital education platforms, completing the fifth contribution in this thesis (C5).

This chapter is structured as follows. We start in Section 10.2 by presenting the design of our architecture to integrate LLMs into LXPs, as well as our model for configuring educational chatbots powered by LLMs. In Section 10.3 we present the methodology used for an empirical case study used to validate the implementation of this architecture and model on Graasp. We present our results in Section 10.4 and discuss their implications in Section 10.5. We then conclude in Section 10.6.

## 10.2 Design

Our design comprises two elements. First, we present an architecture to guide the integration of chatbots powered by LLMs into LXPs. We then outline the model that we propose to structure how educators configure these chatbots. In this section, we present both elements.

### 10.2.1 Architecture

Our architecture consists of six components that interact through three main processes. These components are loosely coupled and abstractly defined in order to allow developers to adapt them as needed for the particular constraints that might be present in different LXPs. Our architecture also builds on some of the concepts presented in Chapter 2 (e.g., ensuring that there is an interface aimed at educators and an interface aimed at learners) and places a strong emphasis on portable apps that can serve to host educational chatbots. This design decision allowed us to take full advantage of our application development architecture (ADA)—and specifically the Graasp ADK—for our implementation.

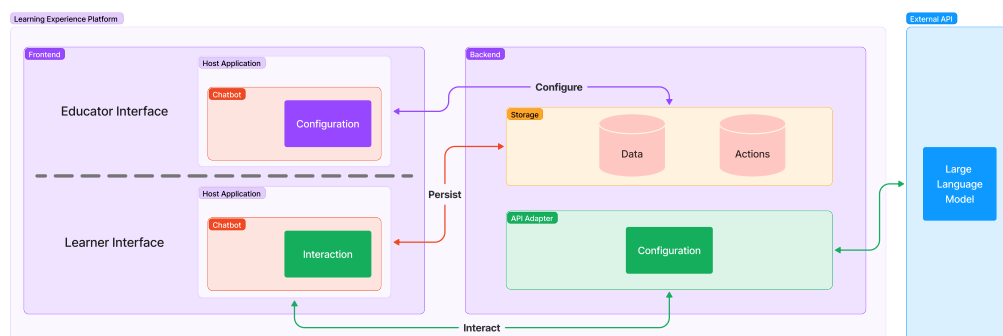


Figure 10.1: Our architecture comprises six loosely-coupled components that interact through three main processes.

### Components

The architecture defines six components that interact to facilitate the integration of chatbots powered by LLMs into educational contexts: (i) LXP, (ii) host application, (iii) chatbot, (iv) API adapter, (v) storage, (vi) external API. In this section, we detail the functionalities provided by each component.

1. **LXP:** The learning activity takes place on a learning experience platform (LXP). As noted in Chapter 2, this platform should provide a way to differentiate content that is visible to educators from content that is visible to learners. Using a dedicated view, educators should be able to create a learning activity by curating multimedia content, such as

images, text, and videos. Most importantly, educators should be able to select interactive learning resources (e.g., web applications) that can host chatbots. That is, chatbots are not directly embedded in the LXP, but in interactive applications that can be added to the learning activity. We refer to these applications as *host applications*.

2. **Host Application:** The host application is the interactive learning resource that can be embedded in LXPs to support learner interactions with chatbots. A host application exposes a configuration panel for educators to activate and configure chatbots. Once activated, these chatbots are visible to learners interacting with the application. For this, the host application should provide an interface through which learners can communicate with the chatbot (e.g., a forum, chat box, or message thread). How a chatbot interacts with learners is entirely defined by its configuration and the interaction affordances provided by the host application.
3. **Chatbot:** A chatbot in our architecture is not an application by default, but a component that can be activated and configured *within* an application. This increases the portability and customizability of a chatbot, which can be developed independently from the host application(s) it can be embedded in, and by educators without technical backgrounds. Hence, the chatbot can be considered to be an open educational resource that is defined by its configuration. The specific elements comprising a chatbot's configuration will be further detailed in Section 10.2.2.
4. **API Adapter:** The API adapter is embedded in the LXP and serves to communicate with the API exposed by the LLM provider. This adapter can also be deployed outside of a single LXP, thus serving multiple platforms. Given that adapters interface with APIs external to the LXP, they can also provide fallback responses in case the APIs are not accessible. Nevertheless, adapters are stateless and should only serve to handle requests from host applications, thus delegating the storage of any information to the LXP.
5. **Storage:** Two types of data are stored by the LXP: (i) application data and (ii) application actions. Application data refer to the content of a learner's interaction with the chatbot, including both the messages provided by the learner and the chatbot's responses. Application actions refer to activity traces of how a learner interacts with the conversational interface in which the chatbot is embedded, including keystrokes, clicks, and other events that can be captured by the browser. These actions can serve to deliver learning analytics (LA) and provide a more nuanced depiction of how learners interact with chatbots.
6. **External API:** Finally, our architecture requires an LLM that is accessible through a public API in order to generate the responses to the queries provided by the learners.



## Process

There are three main processes that define the interactions between the aforementioned components. These processes are outlined below.

1. **Configure:** Following the contextual component of the ADA presented in Chapter 2, educators should be able to configure the chatbot through the educator interface of the host application. The chatbot's configuration is kept in the storage component and allows for the personalization of the chatbot integration for each educator's particular needs. This configuration is also seamlessly reproducible, allowing educators to quickly replicate chatbot integrations across learning activities.
2. **Interact:** The core process defined by our architecture is how learners interact with the chatbot. When a learner interacts with the chatbot, the host application's learner interface connects to an external LLM API provider through the API adapter component hosted in the LXP backend.
3. **Persist:** As interactions take place, the host application ensures that the outcomes of these interactions are *persisted* on the LXP. These outcomes mainly concern the conversation between the chatbot and the learner (and any related content such as emoji reactions) but also include any actions that the learner might take using the chatbot interface. These actions can then be used to provide LA.

### 10.2.2 Configuration Model

As mentioned in Section 10.1, one challenge limiting chatbot adoption in education is the complex configuration process. To guide educators in this process, we propose a configuration model for educational chatbots powered by LLMs. Our model consists of four elements (i) *identity*, (ii) *prompt*, (iii) *cue*, and (iv) *scope*. In this section, we describe these elements in detail.

#### Identity

The importance of providing the appropriate interface for chatbots, as highlighted by Kasneci et al. (2023), was noted in Chapter 1. As discussed in detail in Chapter 8, a key element of this interface is the identity that the chatbot will use in its interactions with users. Educators, therefore, need to provide the chatbot with a minimum number of elements that will constitute its basic identity. These elements should, at a minimum, include their name, avatar, and description, as suggested by best practices for the design and implementation of chatbots (Chaves et al., 2020; Ferman Guerra, 2018; Følstad & Brandtzaeg, 2020).

### Prompt

As discussed in Chapter 1, prompt engineering is a crucial step in configuring an LLM for a specific task. In an attempt to guide prompt engineering for educational chatbots, we propose that prompts follow a fixed structure comprising four parts: (i) *scenario*, (ii) *boundary*, (iii) *continuation*, and (iv) *example*. These parts, which are guided by examples provided by the OpenAI API Documentation (OpenAI, 2023), are detailed in this section.

1. **Scenario:** The scenario provides a high-level description of the pedagogical context in which the interaction is taking place. It should include the subject matter, as well as the nature of the learning activity in which the host application is embedded. *Example: The following is a conversation between a chatbot and a student discussing the correction of an exercise on linting, ESLint, code styling, and best practices in JavaScript.*
2. **Boundary:** The boundary defines the conversational limits that the chatbot should adhere to. These limits are usually subject-related, but can also pertain to language-related items such as diction or the use of emoji. *Example: The chatbot can only discuss topics related to computer science and uses only gender-neutral emoji.*
3. **Continuation:** The continuation aims to provide ways in which the conversation between the chatbot and the learner can continue without having to rely solely on the learner's initiative. *Example: After each response, the chatbot gives the student one or two options to continue the conversation.*
4. **Example:** One or more examples serve to capture the expected outcome of a learner's interaction with the chatbot. These examples are both useful for the model as well as for the educator, who can later use them as a baseline to assess how learners actually interacted with the chatbot. *Example:*
  - [Chatbot] In Python, function names are usually written in lowercase *snake\_case*. Do you want me to provide you with an example of a function name written in Python?
  - [Student] Yes, please.
  - [Chatbot] Here you go! `my_function()`
  - [Student] Why is that called *snake case*?
  - [Chatbot] Good question! Because the words are written in lowercase with under-scores in between, which makes the function name look like a snake!

### Cue

Prompt engineering concerns the LLM that is generating the chatbot's responses. A second element of a chatbot's configuration is how it can *cue* users into interacting with it without interfering with the learning process. More specifically, a cue is a snippet of text that is

presented to learners in order to signal the possibility of *complementing* the learning activity they are currently participating in with a chatbot interaction. Cues should be tightly linked to the pedagogical context in which the chatbot is operating. This aims to increase a chatbot's relevance to the task at hand. A cue consists of two key parts.

1. **Perk:** A perk should summarize what the learner stands to gain from the chatbot interaction. Perks can encapsulate the possibility to learn more or access complementary explanations about a given subject or activity. *Example: In this code snippet, the function name does not follow the camelCase convention. I would suggest that it be written like this: `getFoodItems()` ;.*
2. **Hook:** The hook follows the perk by inviting the user to interact with the chatbot. *Example: Do you agree?*

An example cue is depicted in Figure 10.2.

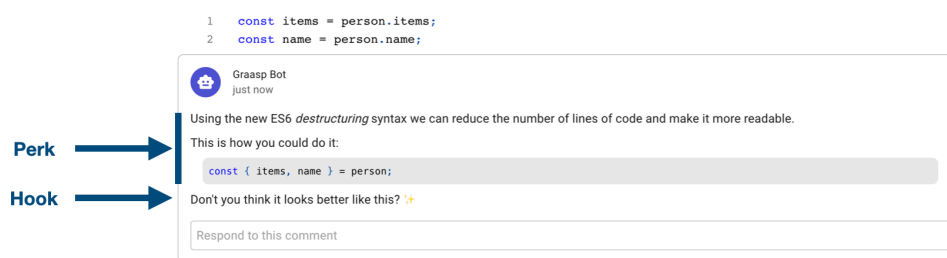


Figure 10.2: The cue is used to invite learners to interact with the chatbot.

## Scope

Finally, it is important to note that when a chatbot interacts with a learner, by default it should only take into consideration the history of the interactions it has held with that learner in particular. This separates the conversations a chatbot has with different learners and provides an element of privacy to the interactions. However, chatbots can also be configured in two other ways. First, to share the content of chatbot-learner conversations with the educator(s). This option allows the educator to harness the chatbot as a conversational interface in order to inquire about individual learners or complete cohorts of learners. Second, to share the content of chatbot-learner conversations with other learners. This option allows for chatbot-mediated collaborative learning to take place, increasing the chatbot's applicability to different pedagogical scenarios. Nevertheless, if a chatbot can share information about its interactions with others, this should be transparent to the learners, in order to adhere to privacy and ethical guidelines.

### 10.3 Methodology

To test the applicability of our architecture to scalable learning scenarios, we implemented our design and conducted an evaluation. Our evaluation focused on addressing one main research question, which is an extension of **RQ3**:

How does incorporating a large language model-based educational chatbot to support a lesson on software engineering best practices affect the learning experience? (**RQ3e**)

This evaluation took the form of a case study consisting of a between-subjects controlled experiment comprising one control (no chatbot) and one treatment (chatbot) condition. For both conditions, we analyzed five aspects of the learning experience: (i) *short-term learning gains*, (ii) *engagement*, (iii) *self-reflection on the learning experience*, (iv) *feedback regarding the lesson*, and (v) *usability*. The small-scale nature of our study allowed us to conduct a mixed-method analysis, involving both quantitative and qualitative methods. In this section, we present the methodology used for this evaluation.

#### 10.3.1 Scenario

To ensure ecological validity, our evaluation took place in a formal education setting. As part of their coursework, students completed an in-class online lesson consisting of an ungraded 45-minute exercise. As with our previous empirical studies, the exercise consisted of a code review notebook covering JavaScript code style standards based on the ESLint and Prettier static analysis tools. In the following sections, we outline the technological support and pedagogical scenario used in our evaluation.

#### Technological Support

Our chatbot was integrated into the Code Capsule app, which was designed for Graasp. Code Capsule is an app that combines the functionalities of Code Review and Code. That is, it allows learners to both review and execute code using the same app. Code Capsule also allows the integration of chatbots when used to review code, supporting the same functionalities provided by Code Review. Figure 10.3 depicts how Code Capsule can be used to execute code (left) and perform code reviews (right).

We configured our chatbot following the model described in Section 10.2.2. To remain gender neutral and maximize consistency with Graasp, we named our chatbot *Graasp Bot* and represented it with a robot avatar, as depicted in Figure 10.4. All chatbot interactions were scoped at the individual level, so students could only see their own interactions and not those of their peers. The prompts for each code snippet were defined following the pattern below. Note that placeholders indicating the parts that would be different for each code snippet are

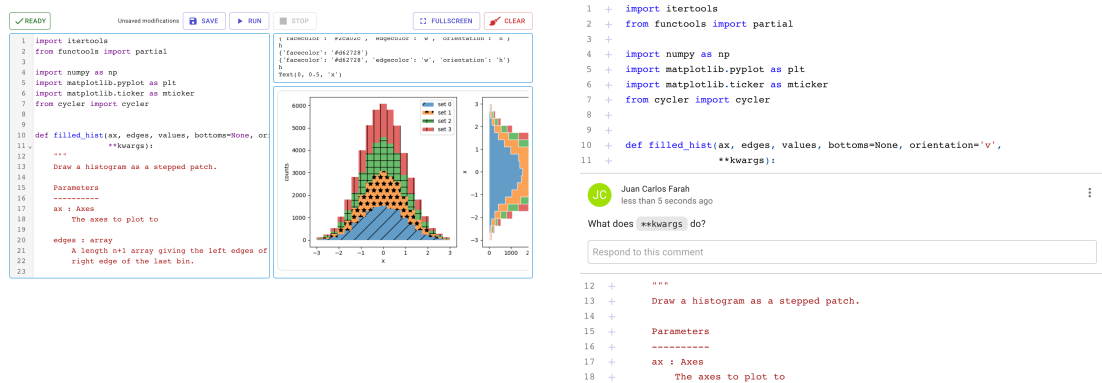


Figure 10.3: The Code Capsule app can be used to write and execute (left) or to review code (right). The code used in these examples has been adapted from the Python Matplotlib library's documentation (Hunter, 2007).

presented between angle brackets (<>). Furthermore, following the Markdown format, text between three backticks (```) would be presented as code.

### Scenario:

The following is a conversation between a chatbot and a student discussing the correction of an exercise about linting, ESLint, code styling, and best practices in JavaScript.

### Boundary:

(None)

### Continuation:

After each response, the chatbot gives the student one or two options to continue the conversation.

### Example:

Chatbot: <EXPLANATION OF WHAT IS WRONG IN THE CODE SNIPPET>. So, I would change the following line of code:

```
...
<INCORRECT CODE>
...
```

To the following:

```
...
<CORRECT CODE>
...
```

<QUESTION ASKING THE STUDENT IF THEY UNDERSTAND THE DIFFERENCE>

Student: OK. <QUESTION ASKING THE CHATBOT SOMETHING RELATED TO THE ISSUE>

Chatbot: <ANSWER>. Do you want me to provide you with an example of <THE ISSUE>?

Student: No, that's OK. Thanks! Any other code issues I should be aware of?

## Chapter 10. Large Language Models

The cue was dependent on each code snippet in which the chatbot was embedded. All of the cues followed the same pattern. The perk consisted of an explanation of an issue present in the code snippet along with a suggestion on how to fix the issue. The hook consisted of a question asking students whether they agreed with some facet of the proposed solution. An example is provided in Figure 10.4.

### Destructuring

With ES6, a new syntax was added for creating variables from an array index or object property, called *destructuring*. Destructuring saves you from creating temporary references for those properties and from repetitive access of the object. Repeating object access creates more repetitive code, requires more reading, and creates more opportunities for mistakes. Destructuring objects also provides a single site of definition of the object structure that is used in the block, rather than requiring reading the entire block to determine what is used.

Variables `items` and `name` can therefore become one line using object destructuring.

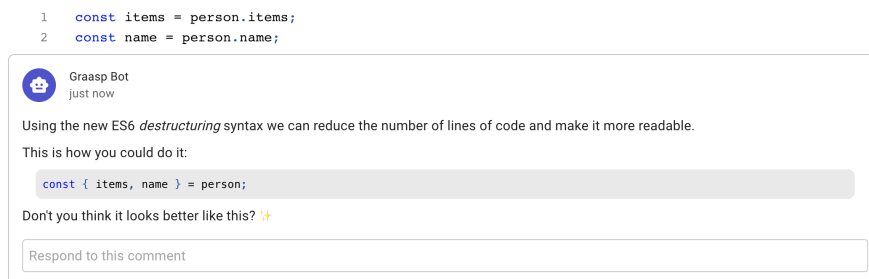


Figure 10.4: The cue consisted of a perk explaining the issue present in the code snippet and a hook asking students whether they agreed with the solution proposed.

When students interacted with our chatbot, Code Capsule interfaced with OpenAI's API following an implementation of the architecture presented in Section 10.2.1, which was integrated into Graasp. The API used the GPT-3 `text-davinci-003` model, with a `temperature` parameter of 0.9 (higher values make completions of the same prompt more random), a `presence_penalty` of 0.6 (higher values penalize new tokens if they have already appeared), a `top_p` parameter of 1 (always returning the best completion to the prompt), and the maximum number of tokens to be included in the chatbot response fixed at 150.

### Pedagogical Scenario

As in our previous studies using code review notebooks, the lesson used in this experiment followed the *Fixer Upper* pedagogical pattern (Bergin, 2000). In our case, we structured the lesson over 10 phases, which students were meant to navigate sequentially (see Figure 10.5). Students were first introduced to the lesson (Phase 1), then asked to complete a short exercise that served as a pre-test to gauge their knowledge of JavaScript code style standards (Phase 2). Phases 3 (*Introduction*) and 4 (*ESLint*) covered the concept of code linting and ESLint specifically, while Phases 5 (*Styling*) and 6 (*Best Practices*) presented examples of code style standards that students should follow when writing JavaScript code. In these phases, 10 code snippets were included using Code Capsule alongside a textual explanation of the issue present in the snippet. For students in the treatment condition, the code snippet also included a cue from Graasp Bot, as shown in Figure 10.4. Phase 7 consisted of an exercise that served as

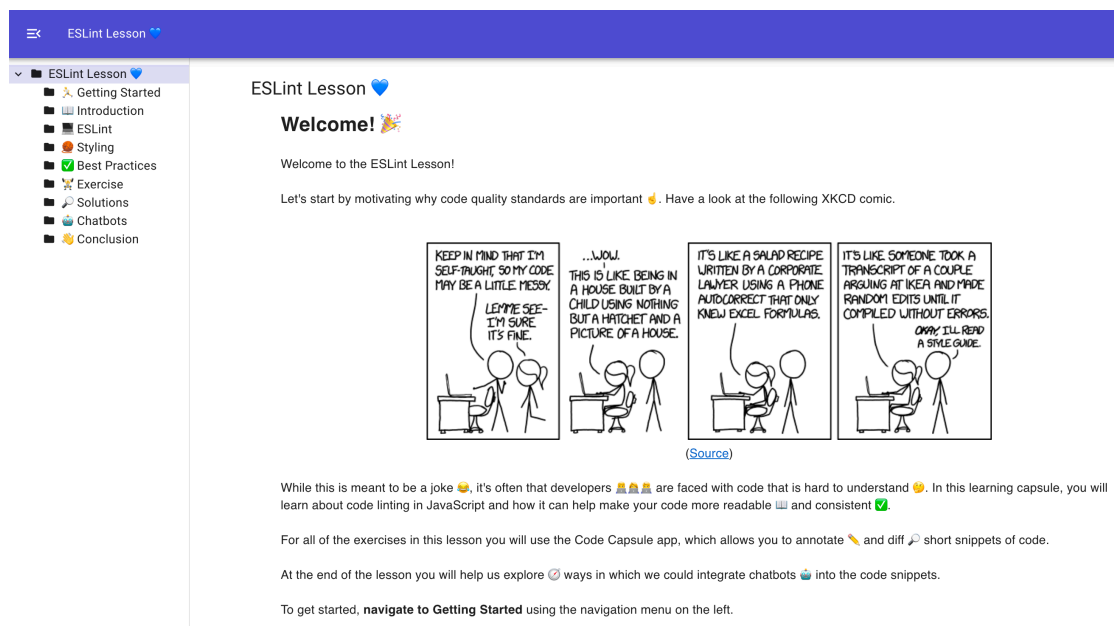


Figure 10.5: The lesson used for this study consisted of an improved version of the code review notebook used in the studies presented in Chapters 5 and 9. (Comic strip by Munroe (2015).)

a post-test, while Phase 8 presented the solutions to the exercise and asked the students to reflect on their performance in the exercise. For students in the treatment condition, Phase 8 also included explanations presented by Graasp Bot. Phase 9 was also different between the control and treatment conditions. Students in the control condition were asked to imagine how they would integrate chatbots into this exercise and to provide sample dialogs that they would envision having with the chatbot. On the other hand, students in the treatment condition were asked to report on their experience interacting with the chatbot. Finally, Phase 10 served as a conclusion to the exercise and provided a link to a User Experience Questionnaire (UEQ) (Laugwitz et al., 2008).

### 10.3.2 Participants

We recruited 28 third-year bachelor students taking part in a course on human-computer interaction at the School of Engineering and Architecture of Fribourg, Switzerland. A total of 26 students—25 male, 1 female—completed the study. Students were informed that this was an ungraded, optional exercise. However, to incentivize participation and limit the effect of sample bias that was possibly observed in previous studies, students were given extra credit for completing the activity.

### 10.3.3 Procedure

We conducted this study in January 2023. Students were given one hour to complete the 45-minute exercise. As explained in Section 10.3.1, the exercise was the same under both conditions except for Phases 5, 6, 8, and 9. In the control condition, explanations were presented in text alongside—but not embedded within—the code review application, while in the treatment condition, these explanations were complemented with explanations from Graasp Bot. At the end of the exercise, students were directed to complete a UEQ.

### 10.3.4 Instruments

As in our previous studies, short-term learning gains were operationalized based on a learner's performance in the pre- and post-tests by calculating the difference between both tests. These gains could range from –100% to 100%, inclusive. Engagement was measured by calculating the total amount of time spent in each phase of the lesson during the one hour time frame that was allocated to the lesson. Self-reflection and feedback were respectively operationalized through the following two open-ended questions:

- *Did you manage to find all of these [issues]? If not, which ones did you miss? Did you find any of them particularly tricky/helpful? Please answer in the text box below.*
- *What did you think about this lesson? Any comments, suggestions, or feedback?*

A second feedback question was relevant only to the students in the treatment group. This second question was only used for our qualitative analysis of the feedback aspect.

- *In a few phrases, describe your experience interacting with the chatbot used in this activity. What did you like about it? What could be improved?*

Finally, usability was measured with the UEQ.

### 10.3.5 Data Analysis

Our data analysis comprised both quantitative and qualitative methods. We applied both descriptive and inferential statistics to our quantitative data, reporting the sample sizes ( $n$ ), means ( $\bar{x}$ ), medians ( $\tilde{x}$ ), and standard deviations ( $s_x$ ), as well as the results of  $t$ -tests for independent samples comparing across the two conditions. To perform sentiment analyses on students' self-reflection and feedback responses, we used VADER (Hutto et al., 2014), which assigns a sentiment score ranging from –1 (negative sentiment) to +1 (positive sentiment), inclusive. The results of the UEQ were analyzed using its data analysis toolkit (Laugwitz et al., 2008). Finally, open-ended responses were analyzed using qualitative methods, following line-by-line data coding (Charmaz, 2006).



## 10.4 Results

In this section, we present our results with respect to each of the five aspects of our evaluation.

### 10.4.1 Learning Gains

The mean learning gains were  $\bar{x} = 0.429$  ( $\tilde{x} = 0.369$ ,  $s_x = 0.212$ ) under the control condition and  $\bar{x} = 0.478$  ( $\tilde{x} = 0.519$ ,  $s_x = 0.177$ ) under the treatment condition. These results—illustrated in Figure 10.6—show that both conditions led to—on average—positive learning gains for students, with all students except one achieving positive learning gains. Although the mean learning gain was higher in the treatment condition, a  $t$ -test for independent samples did not show a significant difference between conditions ( $p = 0.538$ ).

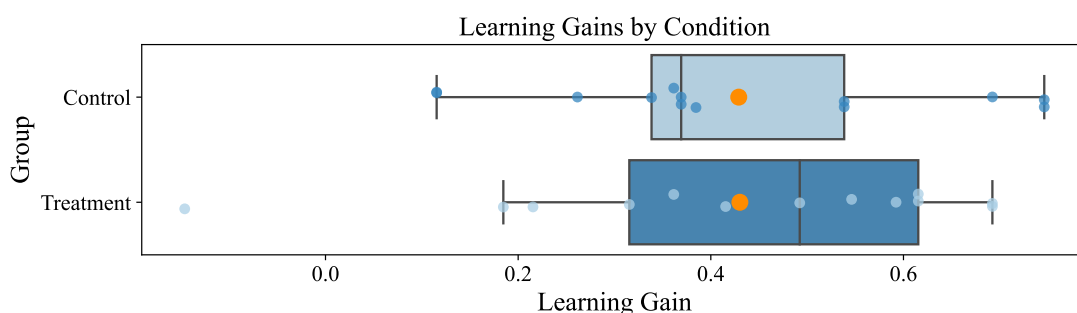


Figure 10.6: Learning Gains

### 10.4.2 Engagement

On average, learners in the treatment condition spent a total of  $\bar{x} = 41.7$  minutes ( $\tilde{x} = 37.8$ ,  $s_x = 9.31$ ,  $x_{\min} = 29.4$ ,  $x_{\max} = 59.5$ ) to complete the lesson, while learners in the control condition did so in  $\bar{x} = 40.1$  minutes ( $\tilde{x} = 36.5$ ,  $s_x = 10.4$ ,  $x_{\min} = 22.2$ ,  $x_{\max} = 55.2$ ). There were no significant differences in the time spent by students either overall (see Figure 10.7) or across the ten phases that constituted our lesson (see Figure 10.8).

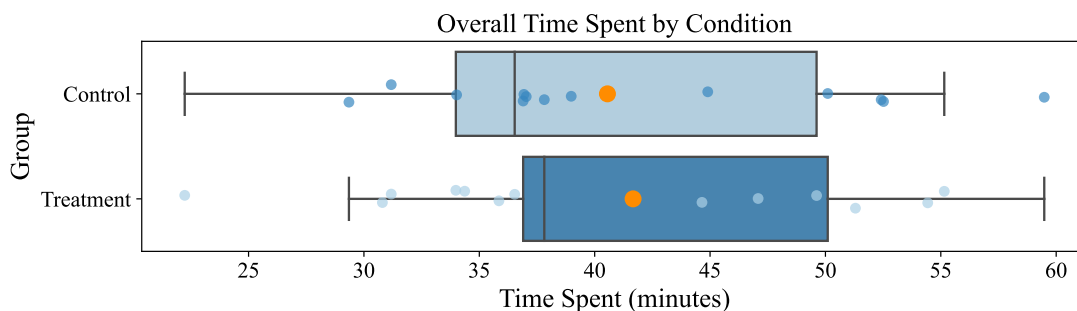


Figure 10.7: Overall Time Spent to Complete the Lesson

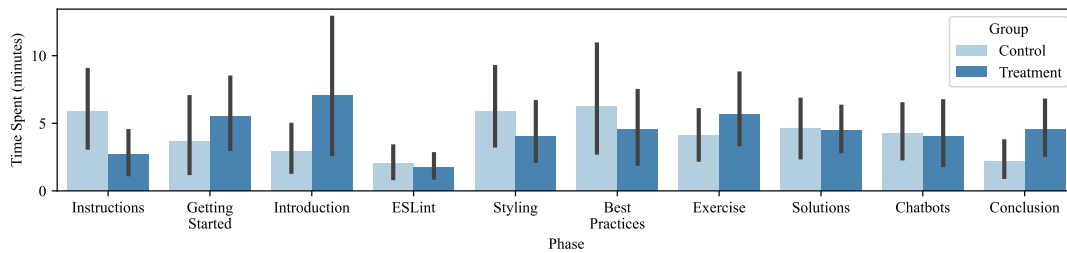


Figure 10.8: Time Spent per Phase

### 10.4.3 Self-Reflection

A total of 22 students (10 control, 12 treatment) provided responses to the self-reflection question. The sentiment analysis performed on student responses to the self-reflection question did not produce any significant differences between conditions. Nevertheless—as shown in Figure 10.9—the distribution of scores in the treatment condition had a more positive tendency than the scores in the control condition. Specifically, the responses of the students in the control group resulted in a mean sentiment score of  $\bar{x} = -0.128$  ( $\tilde{x} = -0.0766$ ,  $s_x = 0.454$ ,  $x_{\min} = -0.743$ ,  $x_{\max} = 0.757$ ), while those in the treatment condition resulted in a mean score of  $\bar{x} = 0.0297$  ( $\tilde{x} = 0.0386$ ,  $s_x = 0.423$ ,  $x_{\min} = -0.595$ ,  $x_{\max} = 0.649$ ).

Our qualitative analysis showed that responses were consistent between both groups, with students providing short answers in which they quickly described what they missed. All students except five (two control, three treatment) specifically listed at least one issue they missed. A typical answer is provided below.

“I forgot a let that had to be const, a comma after the last element of an object and a semicolon at the end of a line.”

Six students (four control, two treatment) provided more detail regarding the issues they missed. In this case, a typical answer is provided below.

“I forgot the first let giftList to const giftList despite seeing it for total at the end. I fell into the trap thinking it was redeclared in the getGiftsTotal(person) function because it had the same name.”

It is also worth noting that two comments in the treatment condition included emoji or emoticons, while no comments in the control condition included these affordances.

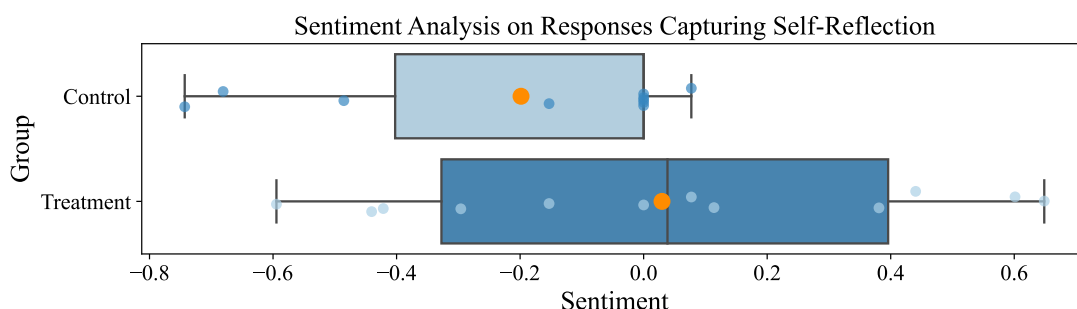


Figure 10.9: Sentiment Analysis on Responses Capturing Self-Reflection

#### 10.4.4 Feedback

A total of 22 students (10 control, 12 treatment) provided responses to the feedback question. The sentiment analysis performed on these responses did not yield any significant differences between conditions. As shown in Figure 10.10, the distribution of the scores was mostly positive in both conditions, with only a few negative outliers. Responses from students in the control group resulted in a mean sentiment score of  $\bar{x} = 0.451$  ( $\tilde{x} = 0.556$ ,  $s_x = 0.400$ ,  $x_{\min} = -0.317$ ,  $x_{\max} = 0.859$ ), while those in the treatment condition resulted in a mean score of  $\bar{x} = 0.460$  ( $\tilde{x} = 0.598$ ,  $s_x = 0.331$ ,  $x_{\min} = -0.356$ ,  $x_{\max} = 0.796$ ).

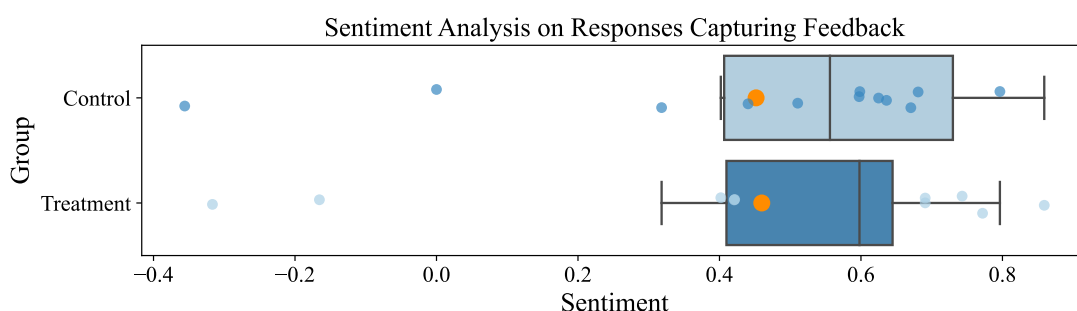


Figure 10.10: Sentiment Analysis on Responses Capturing Feedback

Our qualitative analysis of the first feedback question—which concerned both conditions—showed that all students except two (two treatment) provided a positive comment. Codes such as *good*, *great*, *fun*, *interesting*, and *helpful* were prevalent in answers from students in both conditions. Regarding the negative feedback, while one of the comments provided a minor, off-topic comment regarding the color of the user interface, the other comment questioned the need for a chatbot.

“Was the chatbot REALLY necessary? This lesson needs to end on a link to a lesson/tutorial on how to use/configure/install/firststeps/basics/... on ESLint.”

However, four students specifically provided positive feedback regarding their interactions

with the chatbot. An example of these comments is provided below.

“It was helpful and I liked the interactivity with a bot.”

Finally, one student noted that while the interactivity and response time provided by the chatbot was positive, the chatbot would *never* replace the educator.

“ESLint was interesting. It’s nice to have an answer directly to our questions. After that, it will never replace the answers of a teacher (even if the answer is direct).”

Furthermore, all students in the treatment group ( $n = 13$ ) provided an answer to the second question, which specifically asked about the chatbot and was therefore only visible to students in the treatment group. While eight students provided positive feedback on the chatbot, four of these comments also included a note on how the chatbot had been “repetitive” or “asked too many questions”. One student noted the following:

“It was nice, but sometimes repetitive. He wished me twice a great day and when I asked how to implement something in the linter, it didn’t show me the code. But overall, I find it ludic and it’s a nice way to learn since we have interactions. May be interesting for children too. Maybe less with adults.”

Repetitiveness was also observed in the five negative comments, with students urging the chatbot to “stop asking questions at the end” or characterizing chatbots as “pushy salesmen”. One student provided the following constructive comment:

“Sometimes less interaction is more. In this lesson maybe too many interactions are offered and this could be at some point a bit annoying for the user. But still if correctly dosed it may bring some value for the user!”

### 10.4.5 Usability

Both groups rated the usability of the lesson positively. Compared to the benchmark provided by the UEQ, in the control group, the results achieved were above average (25% of results better, 50% of results worse) for four dimensions—*attractiveness*, *dependability*, *stimulation*, and *novelty*—while they were good (10% of results better, 75% of results worse) for *efficiency* and excellent (in the range of the 10% best results) for *perspicuity*. In the treatment group, the results achieved were above average for *stimulation*, good for three dimensions—*attractiveness*, *efficiency*, and *dependability*—and excellent for *perspicuity* and *novelty*.

When comparing between conditions, however, two-sample *t*-tests did not result in any significant differences across any of the usability dimensions. Nevertheless, it is worth noting that

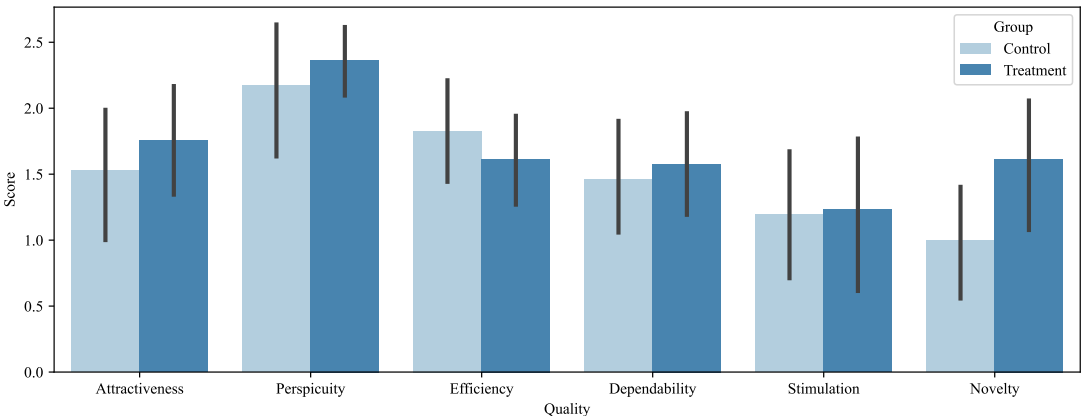


Figure 10.11: Results of the User Experience Questionnaire (UEQ)

the treatment condition achieved a better usability score in all dimensions, except *efficiency*, and specifically for the *novelty* dimension ( $p = 0.0886$ ). These ratings are summarized in Table 10.1 and Figure 10.11.

Table 10.1: UEQ Results Compared to Benchmark Usability Scores

Dimension	Control		Treatment	
	Mean	Benchmark	Mean	Benchmark
Attractiveness	1.53	Above Average	1.76	Good
Perspicuity	2.17	Excellent	2.37	Excellent
Efficiency	1.83	Good	1.62	Good
Dependability	1.46	Above Average	1.58	Good
Stimulation	1.19	Above Average	1.23	Above Average
Novelty	1.00	Above Average	1.62	Excellent

10.5 Discussion and Implications

The results of our evaluation did not surface any significant differences between conditions for any of the five aspects considered. However, there are a few points that stand out and provide interesting insights into the directions that we can explore in future work. In this section, we discuss these points and their implications for the work carried out in the rest of this thesis.

First, results were more positive in the treatment condition across all five aspects. That is, the mean learning gain was higher, students spent—on average—more time in the lesson, were more positive in reflecting on their performance in the post-test, provided more positive feedback, and rated the lesson higher on the UEQ. Although we emphasize that the differences were not statistically significant, these results offer a positive outlook for the integration of this type of educational chatbot into LXPs. At a minimum, these results show that educational

chatbots following our configuration model can complement pedagogical scenarios without interfering with the learning experience. Extensions of these studies with larger cohorts, alternative instruments, and longer exposures could produce more concrete results.

Furthermore, qualitative feedback showed that, overall, learners appreciated the chatbot. Of the 12 students in the treatment condition who provided general feedback through a response to the first feedback question, five explicitly mentioned the chatbot, and only one did so to question whether its integration into the lesson was really necessary. Noting that the chatbot was meant to provide *extra* interaction in the lesson—in order not to compromise the learning experience of the students in the control group—the chatbot was indeed designed to not *really* be necessary. However, in answers to the question specifically asking students in the treatment condition about the chatbot, eight of the 13 students who responded provided a positive comment. The fact that the chatbot was appreciated by the majority of students who were exposed to it is a promising result.

Nevertheless, it is important to acknowledge that the repetitiveness of the questions posed by the chatbot negatively affected the learning experience, as made evident in the qualitative feedback provided by the students in the treatment group. These repeated questions were a result of the chatbot's configuration. Recall from Section 10.3.1:

### Continuation:

```
After each response, the chatbot gives the student one or two options to continue the conversation.
```

Although this continuation strategy was effective in engaging students, it failed to capture the moment when the student wanted to stop interacting with the chatbot. Thus, this continuation quickly backfired and resulted in some students referring to the chatbot as *pushy* and *annoying*. This result sheds light on how what may appear to be a minor detail could potentially have a negative impact on the user experience when using powerful LLMs.

Third, a close inspection of Figure 10.8 shows that the average time spent in the phases that included chatbots (Styling, Best Practices, Solutions, and Chatbots) was actually longer for the control condition than for the treatment condition. While these differences are not significant, the consistency of these results across the four phases stands out. We would have expected students in the treatment condition to spend more time in these phases due to the extra interaction with the chatbot that students in the control condition—who only had to read the accompanying text—did not have to engage with. However, it could be the case that students in the treatment condition favored focusing on the explanation provided by the chatbot, which provided an interactive summary of what was contained in the text. Hence, the chatbot might have provided a faster way of learning the issue that was captured in each code snippet or simply a way to guide a student's focus through the exercise.

Finally, it is worth pointing out that the results from the UEQ show that the differences between conditions were most significant for the *novelty* dimension. For this dimension, the

ratings from students in the treatment condition would be significantly higher at the  $p < 0.1$  level, but not at the  $p < 0.05$  level used in the rest of this thesis. The need for educational technologies to remain novel and attractive is particularly relevant in light of the rapidly changing technological landscape. Learning technologies and LXPs that achieve positive usability results in these dimensions are likely to have an advantage in attracting learners and keeping learners engaged. As mentioned in the previous chapter, positive usability can have an impact on other dimensions of the learning experience (Kanuka et al., 1999; Liaw et al., 2013; Zaharias et al., 2009). The usability aspects of the discussion put forth in Section 9.5—notably that improvements in usability that are not accompanied by improvements in learning gains have been observed in the literature (Davids et al., 2014)—are also applicable here.

These results also have concrete implications for the work conducted in the last part of this thesis. First, the architecture presented in this chapter informs the blueprint that will be proposed in Chapter 12, where we generalize the architecture to include different interaction strategies. Second, our configuration model highlights the importance of collaboration between developers, who are aware of the technical aspects of technologies such as LLMs, and educators who are looking to integrate these technologies into their practice. Without proper guidance, processes such as prompt engineering can be challenging for educators. Guiding the collaboration between these two stakeholders—and also incorporating learners into the design process—is a key motivation behind the conceptual model that will be presented in Chapter 11.

## 10.6 Conclusion

In this chapter, we presented the design of an architecture and a configuration model aimed at supporting developers and educators in integrating chatbots powered by LLMs into LXPs. We implemented this architecture on Graasp and developed a new app—Code Capsule—that featured a connection to OpenAI’s API. We then used Code Capsule to build an improved version of the code review notebook used in Chapters 5 and 8, focusing once more on ESLint and JavaScript best practices. Using this code review notebook, we conducted a between-subjects controlled experiment with 26 software engineering students. Half of the students completed the lesson with support from Graasp Bot—an educational chatbot configured following our model—while the other completed the lesson without support from the chatbot.

While there were no significant differences across the five aspects studied, the results of our study can help optimize our prompt engineering strategy and provide useful examples for educators following our configuration model. Furthermore—as in the previous two chapters—given that learning gains were not impacted by the presence of the chatbot, these findings reinforce the idea that educational chatbots could serve to provide additional information when the educator is not able to provide it.

It is also important to note a number of limitations that could have affected our study. First, while our sample size was appropriate for our mixed methods experiment, expanding our

study to include more subjects could help in the detection of differences between conditions, especially in pedagogical scenarios with shorter durations, where differences might be more subtle than expected. Similarly, increasing exposure by conducting semester-long or longitudinal studies could also serve to better identify the differences that emerge between conditions. Second, while our focus on code review notebooks follows the approach taken in this thesis, exploring this interaction strategy with different pedagogical scenarios and subject matter could help generalize the applicability of our architecture and our configuration model. Finally, incorporating standardized instruments for self-reflection could help reveal more interpretable results regarding how chatbots powered by LLMs can provide support in educational contexts.



## **Design Processes** **Part IV**



---

**I**N this final part of this thesis, we build on the findings of our empirical case studies to address the challenges faced by different stakeholders during the process of designing educational chatbots. As discussed in Chapter 1, our ability to integrate chatbots into educational contexts is limited by challenges that span several domains. Tackling these challenges, therefore, requires interdisciplinary expertise involving multiple stakeholders. Throughout Parts I–III, we have proposed various models and architectures that are relevant to the development and integration of educational chatbots. However, each of these models and architectures was aimed primarily at one type of stakeholder. For example, our application development architecture (ADA) and learning analytics (LA) pipeline were designed to support technical developers, while our code review notebook and configuration model for LLM-powered educational chatbots targeted educators. The contributions proposed in this final part aim to provide bridges between the contributions presented in the previous chapters and to support collaboration among stakeholders. In Chapter 11, we present a conceptual model to guide the participatory design of educational chatbots. This model was validated through an illustrative role-play study conducted with 25 students taking a software design course at the University of Neuchâtel, Switzerland. In Chapter 12, we then present a technical blueprint designed to support developers and educators in integrating educational chatbots into learning apps. In this brief introduction, we highlight related work.

## **Related Work**

Driven by the rising popularity of chatbots, there is a budding line of research proposing guidelines for chatbot design, both in general and specifically for education. Regarding general-purpose chatbots, Ferman Guerra (2018) identified 18 best practices, spanning user-chatbot communication, chatbot features, and human factor concerns. Telang et al. (2018) proposed a domain-independent framework for chatbot engineering, focusing on three *abstractions* verifiable at runtime: goals, plans, and commitments between agents. Shum et al. (2018) proposed a series of design principles to guide the development of social chatbots. These

---

principles comprised broad guidelines, such as equipping chatbots with empathy, social skills, personality, and emotional intelligence.

More recently, Chaves et al. (2020) identified challenges in human-chatbot interaction and suggested ways to address them. Feine, Morana, et al. (2019) designed a chatbot social cue configuration system “to support chatbot engineers in making justified chatbot social cue design decisions”. They evaluated their system with a focus group and two practitioner symposia, receiving positive feedback. In a subsequent publication, Feine, Morana, et al. (2020) also proposed an interactive chatbot development system designed to encourage collaboration between domain experts and chatbot developers. Their evaluation of an implementation of their system through an online experiment in the context of customer service chatbots showed that their system improved subjective and objective engagement.

In education, studies focusing on chatbot design fall mainly under the following lines of research: (i) generic technical architecture, (ii) experiments with developed chatbots, and (iii) reports on a design experience. Villegas-Ch et al. (2020) outlined a comprehensive architecture for a smart campus, including data analysis, decision-making components, and chatbots. Griol et al. (2013) proposed a modular architecture to integrate chatbots into multimodal applications for education, with the ability to easily adapt technical and pedagogical content. They implemented their architecture in their *Geranium* pedagogical system, which focused on speech-based interactions and questionnaire-based learning activities. More generally, Jung et al. (2020) proposed a set of generic educational chatbot design principles derived from a review of various empirical studies.

Nevertheless, few researchers have focused on providing the conceptual tools to guide a chatbot’s design process itself, including the participatory design sessions often championed to design these chatbots. D. Song et al. (2017) adopted a participatory approach to design a conversational agent for online courses in higher education, involving different stakeholders (instructors, computer scientists, and experts in educational technology). Bahja et al. (2020) proposed an iterative step-by-step user-centric methodology for educational chatbots whereby learners and teachers actively collaborate during the requirements analysis, design, and validation phases. Furthermore, Durall Gazulla et al. (2023) adopted a collaborative approach, involving students in the design of chatbots for reflection and self-regulated learning in higher education. The authors focused their study on the challenges encountered during the co-design process of their chatbot and proposed a series of strategies to address these challenges.

While the aforementioned studies have incorporated participatory design sessions for the co-creation of educational chatbots, to the best of our knowledge, no study has proposed chatbot-oriented conceptual tools to structure these sessions. Similarly, there are no standard technical architectures to facilitate the development and integration of educational chatbots that are platform-agnostic and compatible with different subject matter, pedagogical scenarios, and types of interaction. As proposed by Følstad, Araujo, et al. (2021) in the context of the chatbot frameworks and platforms currently available, there is a tangible need to develop new

---

techniques for chatbot design, development, and deployment. The work presented in this part of this thesis is aligned with this line of research.



## 11 Conceptual Model

**D**ESPITE the increasing use of chatbots in educational contexts, few frameworks to structure the design of these chatbots exist. In this chapter, we present a model to guide the design of educational chatbots. Our model aims to structure participatory design sessions in which different stakeholders (educators, developers, and learners) can collaborate to define a chatbot that could be integrated into a learning activity. Specifically, our model focuses on two aspects of this integration: (i) the scenario in which the chatbot will interact with learners and (ii) the interaction the chatbot will hold with learners. The outcomes from these sessions can serve developers involved in the implementation of educational chatbots, as well as educators looking to integrate these chatbots into their practice. Furthermore, we present results from an illustrative study whereby 25 students enrolled in a course on software design took part in a participatory design session to sketch out an educational chatbot. Students were divided into eight groups, assigned the role of one of the different stakeholders, and instructed to use our model. The results of our qualitative analysis of the outcomes from the session and the feedback provided by the students suggest that our model helped structure the design process and align the contributions of the various stakeholders.

By proposing a conceptual model to guide the participatory design of educational chatbots, this chapter partially addresses **RQ4**:

- How can we support the collaborative design of educational chatbots?

The content of this chapter was partially presented in the following publication:

- Farah, J. C., Spaenlehauer, B., Ingram, S., Lasne, F. K.-L., Rodríguez-Triana, M. J., Holzer, A., & Gillet, D. (2023). TRACE: A Conceptual Model to Guide the Design of Educational Chatbots [In Submission]

### 11.1 Introduction

As discussed throughout this thesis, over the past decade, an increasing interest in integrating chatbots into educational contexts has motivated the design, deployment, and evaluation of these chatbots in a wide variety of domains (Wollny et al., 2021). In Parts II and III of this thesis, we explored how to design pedagogical scenarios and interaction strategies for chatbots primarily aimed at software engineering education. While we noted that these designs were possibly applicable to other educational domains, we limited our evaluations to code review notebooks in order to provide a coherent approach when validating our scenarios, strategies, models, and architectures. We will now zoom out of the specific application to software engineering education and explicitly address how to generalize the design processes that we conceived, implemented, and evaluated throughout this thesis.

We start by considering how to provide the appropriate conceptual support to guide the participatory design of chatbot technologies for education. A participatory approach involving end users (e.g., teachers and learners) is particularly important when it comes to designing and proposing new educational tools to increase acceptance and adoption. As chatbots become more prominent in education, it is imperative that their design is aligned both with these end users and with the developers in charge of implementing educational technologies. Furthermore, as noted in Chapter 2, collaboration is essential between developers and educators, given the interdisciplinary nature of the process of designing educational technologies. That is, developers and educators should cooperate and contribute their respective technical and pedagogical skills, which are needed to create educational technologies that are both functional and relevant.

This chapter addresses one particular gap that was presented in the introduction to Part IV. That is the lack of frameworks, models, and design tools to guide the collaborative design processes through which educational chatbots are defined, built, and integrated into pedagogical scenarios. Our work aims to address this gap by proposing a conceptual model that can guide the participatory design of educational chatbots. This model constitutes the sixth contribution put forth in this thesis (C6). By providing the conceptual framework necessary to define the different components underlying the scenario in which an educational chatbot is deployed, as well as the interaction it has with learners, our model could serve to structure participatory design sessions involving educators, developers, and learners. In this chapter, we present our design as well as an illustrative study using this model.

This chapter is structured as follows. We present the design of our model in Section 11.2. In Section 11.3, we outline the methodology followed to validate our model. We present the results of this evaluation in Section 11.4. In Section 11.5, we discuss these results and their implications on the rest of the work conducted in this thesis. Finally, we conclude in Section 11.6.



## 11.2 Design

Our model was conceived to guide educators, developers, and learners in designing task-oriented chatbots for education. The aim is to provide a conceptual tool that can serve to structure participatory design sessions in which the stakeholders collaborate to co-design these chatbots. The model provides two guidelines. First, it proposes five components considered essential in defining a chatbot's integration into a learning activity. Second, it outlines a procedure for how the co-design process should unfold, assigning specific tasks to each of the stakeholders involved. In this section, we present the design of our model, which is summarized in Figure 11.1.

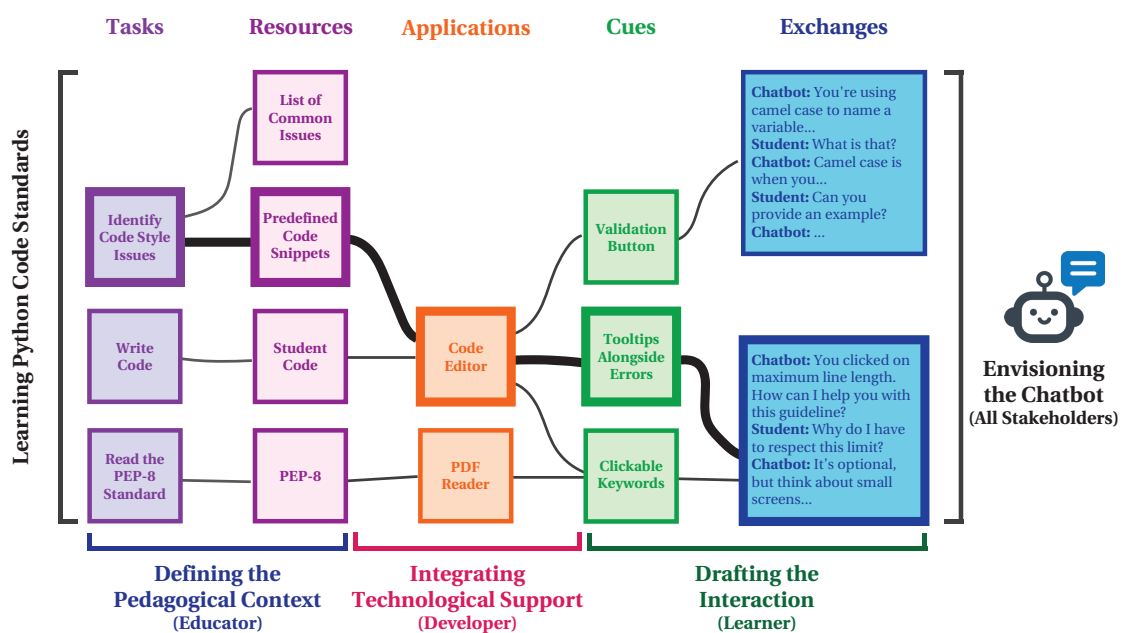


Figure 11.1: Overview of the TRACE model including its components and the tasks associated with each stakeholder. The thick line represents how the stakeholders can then *trace* a line through the components they select for their design.

### 11.2.1 Backdrop

The backdrop for our model's components and the process in which these components are defined is a *learning activity*. Depending on the scope of the design session, this activity can be selected in advance, chosen by all participants, by each group, or ultimately by each individual educator present in the session. The learning activity can be as general (e.g., learning to draw) or as specific (e.g., learning to draw seagulls perched on a boat) as necessary. For illustrative purposes, we will use both *Learning the Python Code Style Standards* and *Discovering the Human Anatomy* as examples of learning activities.

### 11.2.2 Stakeholders

A wide range of stakeholders, from administrators to parents, are involved in the design and implementation of educational technologies. Nevertheless, three key actors stand out given their roles in how these technologies are developed, integrated into the classroom, and exploited. These actors are (i) the developers of these technologies, (ii) the educators who choose which technologies to integrate into their teaching practices, and (iii) the learners who will eventually use these technologies. In this section, we outline their role in the participatory design process structured by our model.

#### Educators

Educators are often the ones who select the material and technology to be used in their practice (Rodríguez-Triana et al., 2016). In our model, educators initiate the design process by selecting the *task* and *resource* that will serve to scaffold the pedagogical scenario. The selection of these two elements serves to frame the chatbot integration with learning scenarios that are relevant to an educator's practice.

#### Developers

Developers in education have the important role of building the technology used in practice, with research suggesting that developers benefit greatly from collaborations with educators when building educational technologies, as was highlighted in Chapter 2. In our model, developers bridge the scenario and interaction aspects of a chatbot's integration. That is, they are the ones in charge of designing the *application* that will both feature the resource(s) selected by the educator and host the chatbot that will interact with the learner.

#### Learners

Learners are the ones who will interact with the chatbot and, therefore, the end users of this technology. Of particular importance is that learners consider the interactions timely and engaging, all the while being relevant to the learning activity at hand. In our model, learners are the ones who will lead the process of defining the *cues* displayed to provide ways to trigger the chatbot interaction, as well as the content of these *exchanges* themselves.

### 11.2.3 Components

Our model focuses on five components to guide the integration of chatbots into educational contexts. Indeed, the model takes its acronym—*TRACE*—from the components it defines. The first two components (Task and Resource) focus on the scenario in which the interaction will take place, while the last two (Cue and Exchange) focus on the interaction itself. The central component (Application) serves as a bridge between the scenario and the interaction and is

responsible for hosting the chatbot. The design decisions made for each component are led by one of the stakeholders involved in the participatory design session. In this section, we present each component, describe it, and provide illustrative examples.

### Tasks

A task in our model can be conceived as a structured step that is aligned with the objectives of the learning activity. This is based on the idea that learning tasks are “an interface between the learners and the information offered in the learning environment” and “serve to activate and control learning processes in order to facilitate successful learning” (Richter, 2012). Educators take the lead in defining the tasks that are best aligned with the learning activity.

*Examples:*

- **Python Code Standards:**

1. Students have to identify code style issues in snippets of code.
2. Students have to write a snippet of code following PEP-8 (van Rossum et al., 2001).

- **Human Anatomy:**

1. Students are asked to label the different parts of the human body.
2. Students are asked to identify the name of a body part based on its function.

### Resources

A resource follows from the definition of a learning object as “any digital resource that can be reused to support learning” (Wiley, 2002), but specifically applied to the learning tasks chosen for a specific learning activity. Furthermore, a resource should be able to be featured in (e.g., embedded in, interfaced through) a software application that can make it accessible to both the learner and the chatbot. In that sense, resources in our model can be any of the examples provided by Wiley (“digital images or photos, live data feeds..., live or prerecorded video or audio snippets, small bits of text, animations, and smaller Web-delivered applications”, as well as “entire Web pages that combine text, images, and other media or applications”) (Wiley, 2002), as long as they can be integrated into a software application compatible with an educational chatbot.

*Examples:*

- **Python Code Standards:**

1. Snippets of code provided by the teacher.
2. Code written by students.

- **Human Anatomy:**

1. An anatomical drawing of the human body.
2. A dictionary of body parts and their functions.

### **Applications**

Given that chatbots are a type of educational technology, it is imperative that the bridge between the pedagogical scenario and the interaction with the educational chatbot be mediated by a software application. Applications can be contextualized to provide different interfaces to different stakeholders. That is, educators can access a dedicated interface where they can configure the application, select the resource(s) that it will feature, and connect the educational chatbot that will be hosted therein. Learners will then use a different interface to access the resource(s) selected by the educator. This learner interface will also provide the affordances through which the learner will interact with the educational chatbot. Given the prevalence of web technologies in digital education, applications are often built to run in web browsers and be compatible with digital learning platforms. Nevertheless, applications can also be standalone desktop or mobile applications. Developers take the lead in designing the application that is most appropriate for the selected resource, as they have the required technical expertise.

*Examples:*

- **Python Code Standards:**

1. An application where students can write, annotate, and execute code.
2. An interactive reader featuring the Python documentation.

- **Human Anatomy:**

1. An application where the human body is shown and students can click on each of the body parts.
2. An application where students can build different body parts similar to the way one builds a puzzle.

## Cues

To integrate the chatbot interaction into the learning activity, we rely on the notion of an *interaction cue*, often employed in educational technology (Hu et al., 2020). Interaction cues serve to inform users of the actions they can take and to guide them toward a particular action (Dillman et al., 2018). The function of a cue in our model is to trigger an exchange between the learner and the chatbot. Cues are digital affordances that are an essential part of the application's learner interface. Cues can be graphical or textual and can be linked to actions that the student takes, featured in elements within the resource embedded in the application, or exposed permanently in the interface. As is often the case, these cues could be paired with visual affordances (e.g., buttons, tooltips) that the learner can interact with (Vermeulen et al., 2013). These cues should be featured in the resource and accessible via the application. Learners take the lead in defining the cues, as they will be the ones triggering and following these cues.

*Examples:*

- **Python Code Standards:**

1. A tooltip appears next to an error as the student types.
2. A button that the student can click on to have the chatbot inspect the code.

- **Human Anatomy:**

1. A popup that appears when a student clicks on a body part.
2. A button to start a tour of the human body mediated by the chatbot.

## Exchanges

The final component serves to illustrate what a conversational exchange between a learner and a chatbot would look like. By providing an example of what a useful interaction with a chatbot could look like, learners can both highlight their expectations of the interaction and indicate the design features, conversational capabilities, and social characteristics they expect the chatbot to have. This is an important aspect of the design of interactive agents and is guided by CASA. As such, understanding—and possibly curbing—learners' expectations of these interactions can serve to design a chatbot that is better adapted to the learning activity. Furthermore, these exchanges could be used to create longer examples that could eventually serve to fine-tune the language models used by the chatbots. Once again, learners take the lead in defining the exchanges, since they will be the ones interacting with the chatbot. Note that ellipses are included for succinctness in the examples below.

*Examples:*

- **Python Code Standards:**

- **Chatbot:** Hey! You're using camel case to name a variable. Did you know that you have to use snake case for variable names?
- **Student:** No. Why snake case and not camel case?
- **Chatbot:** Well, snake case is the default for Python...

- **Human Anatomy:**

- **Chatbot:** You clicked on the organ that pumps blood throughout the body. Can you name this organ?
- **Student:** It's the heart.
- **Chatbot:** Exactly! Did you know that the heart...

### 11.2.4 Process

The following process is a way in which the components proposed by our model can be defined by the stakeholders involved in the participatory design session. This process consists of four phases and can be integrated as the central activity of a participatory design workshop, following initial icebreakers and selection of the learning activities that will serve as a backdrop to the co-design. In this section, we describe these phases, providing sample questions that can serve as a guide for discussion among stakeholders, as well as a list of the outcomes that should be produced in each phase.

#### Defining the Pedagogical Scenario

In this first phase, educators take the design lead. The educator will start by proposing one or more tasks that students could do in relation to the learning activity that has been selected. There is a many-to-many mapping between resources and tasks, as the same resource can support multiple tasks and one task can be supported by many resources. Once the tasks and resources have been mapped, the educator can choose one or more resources that they think will be most relevant to their practice.

*Questions:*

- What tasks can support this learning activity?
- What resources are traditionally used for these tasks?
- Can these resources be delivered digitally?

### *Outcomes:*

- A description of one or more tasks that the learner could engage in.
- A description of one or more resources that can be used to support these tasks.
- A mapping between the tasks and the resources.

### **Integrating Technological Support**

The second phase concerns the technological scaffolding that will serve as a bridge between the pedagogical scenario and the interaction with the chatbot. The developer leads this phase and needs to sketch out an application that could feature the resource(s) selected by the educator. If the resource cannot be embedded in or handled by an application, then a different resource needs to be selected. Once one or more applications have been sketched out, these applications can be mapped back to all the other resources in the list that they can potentially support. Once again, this is a many-to-many mapping.

### *Questions:*

- How can the selected resource(s) be embedded into an application?
- What devices would this application run on?
- What learning platforms is this application compatible with?

### *Outcomes:*

- A description of one or more applications that can feature the resource(s) selected in the previous phase.
- Sketches or mockups of how the application(s) will feature the respective resources.
- A mapping between the resources and the applications.

### **Drafting the Interaction**

The third phase is led by the learner, who will first identify what cues within the application (or the resource embedded therein) should prompt the chatbot to start an interaction with the learner. These cues can also define where the interaction takes place within the interface. The developer needs to ensure that the cues are compatible with the application and can be displayed to students. Once the cues are defined, the learner can choose one or more cues to construct sample exchanges between a learner and the chatbot. These exchanges do not need

## Chapter 11. Conceptual Model

---

to be long or detailed but should contain enough information so as to envision what a dialog would look like.

### *Questions:*

- What elements or affordances in the resource or application can serve to cue the chatbot interaction?
- Are these cues specific to one (type of) resource or are they applicable to other (types of) resources?
- Are the exchanges aligned with the goals of the learning activity?

### *Outcomes:*

- A description of one or more cues that could be present in the selected application(s).
- One or more sample dialogs illustrating an exchange between the chatbot and the learner.
- A mapping between the cues and the exchanges.

## **Envisioning the Chatbot**

At this point in the process, stakeholders will have produced a complete mapping of all the components that could serve to support the integration of chatbots into this learning activity. In the final phase, participants use the five components defined in the previous phases to envision the chatbot. Stakeholders can then highlight the example of each component that is most appropriate for the chatbot integration and *trace* a line from learning activity to chatbot, as shown in Figure 11.1. This line serves to visualize the interactions that learners will have with the chatbot, linking particular examples of exchanges with learning tasks, via the corresponding cues, applications, and resources. Once this link is established, stakeholders can define the chatbot's identity, what social cues it will be equipped with, what it will look like, and what strategies it will use to support these interactions.

All stakeholders are invited to be equally active in this phase, as the chatbot's identity serves to summarize various aspects defined in the previous phases. For example, naming the chatbot *Code Style Bot* makes more sense if the chatbot is actually involved in supporting tasks related to code style, providing a three-dimensional avatar for the chatbot might be more complicated if it is embedded inside a PDF reader application than if it is embedded inside a game, and using rule-based scripts might be prohibitively complex for some exchanges. The final outcome of this phase could be the starting point for future participatory design sessions, an initial prototype, or other iterations of this exercise.



### *Questions:*

- What is the chatbot's name and what does it look like?
- What social cues can the chatbot harness in its interactions with users?
- What technologies power the chatbot so that it can support the sample exchanges?

### *Outcomes:*

- A description of the chatbot's identity.
- A list of the technologies needed to support how the chatbot interacts with learners (e.g., rule-based scripts, AI-based solutions).
- A sketch or mockup of the chatbot embedded in the application.

## 11.3 Methodology

To validate our model, we first conducted two pilot studies comprising a workshop with eight researchers and developers in education and a case study with an undergraduate student completing a semester project on designing educational chatbots. We then conducted an illustrative within-subject study in which 25 students enrolled in a course on software design took part in a simulated participatory design session.

The purpose of these studies was to address one main research question, which is an extension of **RQ4**:

Does the TRACE model help guide educators, developers, and learners in collaboratively designing educational chatbots? (**RQ4a**)

Our analysis focused on two aspects of this research question: (i) alignment between stakeholders and (ii) feedback provided about our model. Qualitative responses addressing the first aspect were captured both before and after participants were presented with our model, while qualitative responses with respect to the second aspect were captured at the end of the study. In this section, we present our two pilot studies before outlining the methodology followed for our illustrative study.

### 11.3.1 Pilot Studies

The purpose of our pilot studies was to refine the model and test its applicability to different educational domains. In this section, we describe the two pilot studies, as well as the main outcomes that emerged from each.

### Pilot Workshop

A first pilot study consisted of a one-hour workshop with eight researchers and developers in education from the École Polytechnique Fédérale de Lausanne, Switzerland. The purpose of this workshop was to present the model and collect preliminary feedback on its usefulness in guiding the design of educational chatbots.

This study took place in October 2022 as a blended learning activity. Using Graasp, participants were first asked to describe how they would integrate a chatbot into a learning activity that consisted in teaching students about the history of their place of birth. Participants reported their answers using the Sticky Notes app, as shown in Figure 11.2.

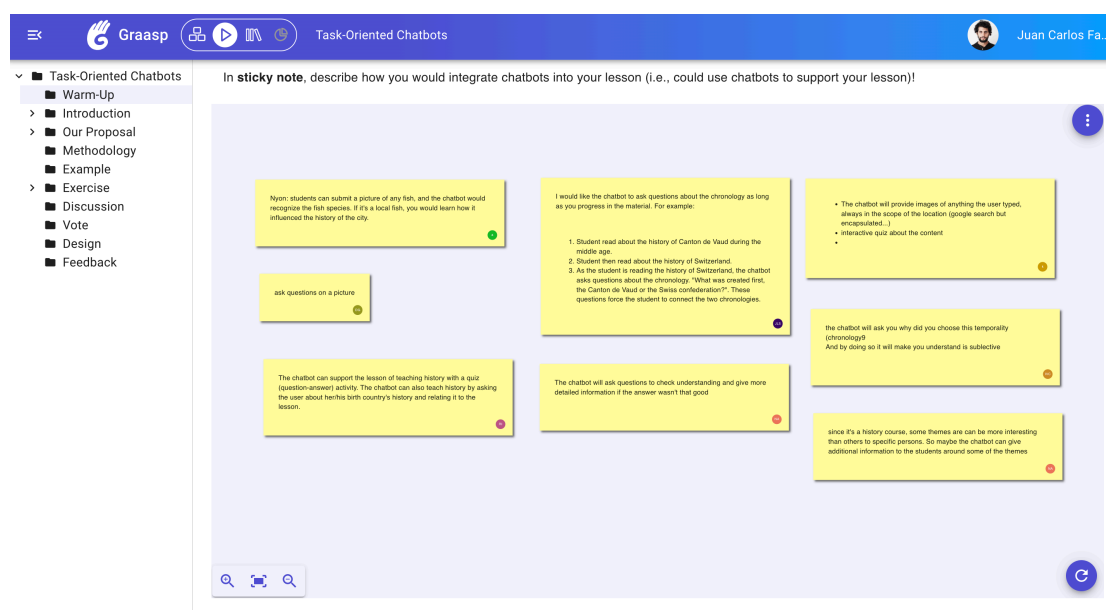


Figure 11.2: Before being exposed to the model, participants were asked to describe how they would integrate a chatbot into a learning activity presenting the history of their place of birth.

We then presented our model, outlining the key components and providing examples. Note that the version of the TRACE model used in the workshop differed slightly from the current version of the model. That is, (i) instead of *applications*, this version of our model guided stakeholders in defining the *actions* that the chatbot could perform and (ii) instead of *cues*, the model guided stakeholders in defining the *concepts* that could guide the chatbot's interaction with learners. Using pen and paper, participants were then asked to use the model to brainstorm, define, and design educational chatbots aimed at supporting two or three learning activities. Participants were then instructed to select one design and outline its TRACE components using Sticky Notes. Examples shared are depicted in Figure 11.3.

At the end of the workshop, participants were invited to provide feedback about the model. Five participants provided written feedback, which was overall positive. However, two partici-

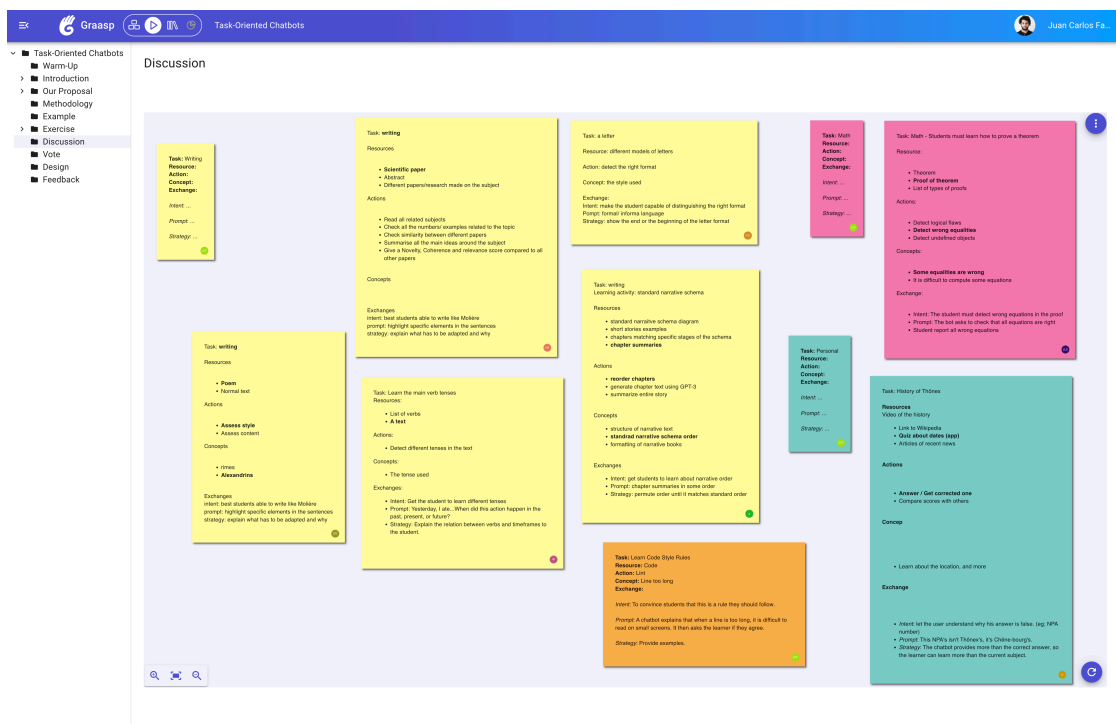


Figure 11.3: Participants in the pilot workshop study were invited to share how they had defined the different components of our model using the Sticky Notes app on Graasp.

pants had trouble understanding the *concepts* component, while two participants reported confusion regarding the *actions* component. Another participant pointed out that an *application* component was missing. This feedback guided our improvement of the model.

## Pilot Case Study

Our model was also used in a pilot case study with an undergraduate student from the School of Computer and Communication Sciences at the École Polytechnique Fédérale de Lausanne, Switzerland. The case study was conducted in the context of a semester project that took place between September 2022 and January 2023. The student used an early version of the TRACE model to design two apps that could be integrated into LXPs and support educational chatbots. Note that the version of the model used by the student was the same version used for the workshop.

This case study served to validate the applicability of outcomes produced during brainstorming sessions that incorporated the TRACE model. Indeed, outcomes from these sessions—which included diagrams similar to the one depicted in Figure 11.4—were then used to guide the implementation of concrete educational chatbot solutions. One of the apps—*Text Analysis*—was later developed with the Graasp ADK, connected to OpenAI’s API, equipped with an educational chatbot, and integrated into Graasp. We will present this app in more detail in

Chapter 12.

Feedback from the student regarding the *action* and *concept* components was aligned with the feedback from the workshop and therefore these components were eventually replaced by *application* and *cue*, respectively.

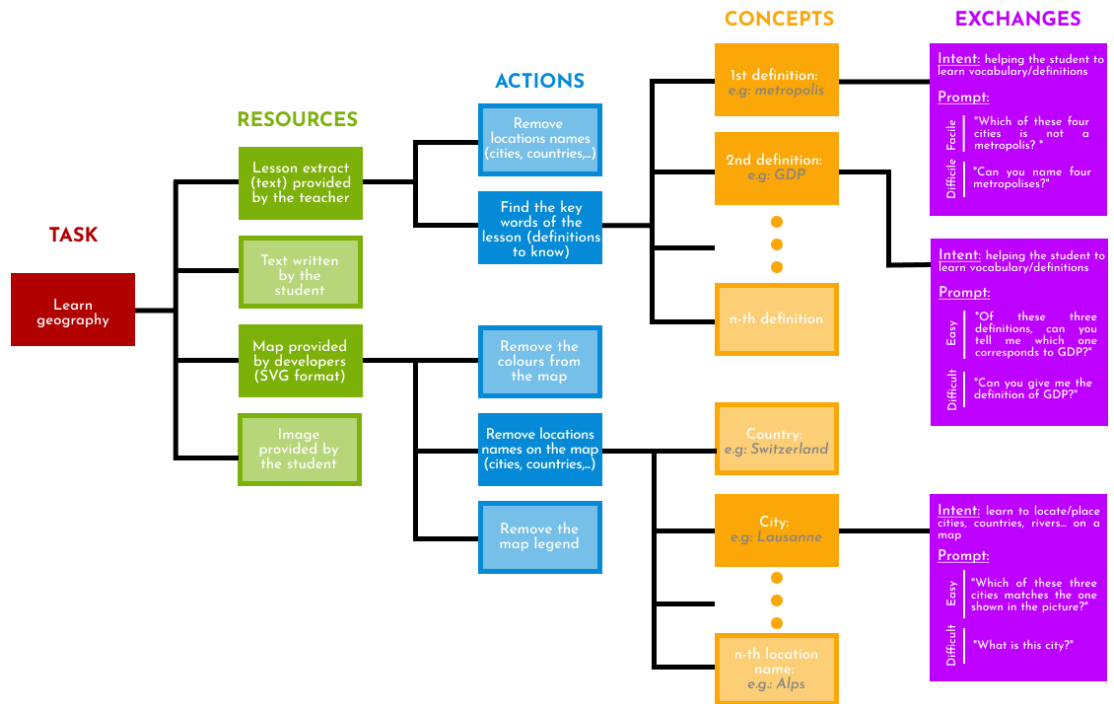


Figure 11.4: TRACE was used to brainstorm the design of educational chatbots during a semester-long pilot case study. Outcomes from these sessions included diagrams similar to the one depicted above, which was produced by an undergraduate student completing a semester project on designing educational chatbots.

11.3.2 Illustrative Study

The purpose of our main study was to demonstrate the feasibility of our proposal and was conducted as an hour-long role-play activity as part of a software design course at the University of Neuchâtel, Switzerland.

11.3.3 Participants

A total of 25 students (7 female, 18 male) were recruited as participants for the session, which took place in December 2022. At the beginning of the session, participants were divided into eight groups of three or four. These groups corresponded to groups in which the students had been working on for one of the course assignments. As such, students within each group were

well acquainted with each other.

#### 11.3.4 Procedure

To motivate the session, after a short introduction, the groups were first asked to come up with a learning activity or choose one from a list (e.g., *how to cook pasta*, *calculating the area of a circle*). Participants were then asked to interact with a chatbot powered by the GPT-3 language model for a few minutes. The topic of this interaction was supposed to be the topic they had come up with or selected from the list.

Within each group, participants were then assigned the role of *educator*, *developer*, or *learner*. They then completed two exercises. In the first exercise, participants were asked to provide a short answer describing—from the point of view of their role—how they would integrate a chatbot into the learning activity chosen by their group. After this exercise, they were introduced to our model through a short presentation. This presentation constituted the intervention in our within-subjects setup.

The short presentation consisted of five slides in which the different components of the model were outlined and examples of each component were proposed. These examples specifically covered the PEP-8 use case. It is important to note that the model used in the presentation featured a slight variation from the one presented in this chapter. That is, this version of our model (*TRA:ACE*) included a second *A* in which stakeholders were invited to design the identity of the *agent* alongside the application. This component has been removed from our current model, as it proved to be redundant with the final outcome produced by the model.

After the intervention, participants completed a second exercise. In this second exercise, participants were instructed to use our model to collaborate on the design of the chatbot integration. At the end of the exercise, the groups provided short descriptions of each of our model's components in the context of their respective learning activity, as well as an optional mockup of the chatbot integration. Finally, qualitative feedback was captured through an open-ended question that asked participants if they found the model useful.

#### 11.3.5 Instruments

We captured qualitative responses through short answers to a series of open-ended questions. In the first exercise—before the presentation of our model—participants were asked to specify how they would integrate the chatbot in a few phrases. Although they worked as a group, each student was asked to provide an answer from the point of view of the role they had been assigned:

*Use the text box below to share your ideas. Mention what your activity and role is and then describe how you would integrate the chatbot in a few phrases. Be as detailed or as general as you would like.*

## Chapter 11. Conceptual Model

---

In the second exercise, participants had to specify each component of our model in a separate input box. For example, for the task, the instructions above the input box read as follows.

*Please describe the Task(s) you will focus on.*

Finally, groups were asked to provide feedback about the model by answering the following questions.

*Did you find the framework useful? Where did it help the most? Where did it help the least? How would you improve the framework to help design chatbot integrations?*

### 11.3.6 Data Analysis

All student responses were analyzed using line-by-line data coding (Charmaz, 2006). Furthermore, responses concerning the alignment aspect were tagged as either *aligned* or *misaligned* depending on whether they were compatible with other responses from the same group. Component descriptions were also tagged as *valid* or *invalid* depending on whether they were applicable to the respective component.

## 11.4 Results

In the first exercise, seven groups provided more than one answer on how they would integrate the chatbot into the chosen learning activity. Only in two of these groups were the answers provided aligned. In the second exercise, all groups provided descriptions of each of the components of our model. For all groups, the descriptions of all components were aligned within each group. Furthermore, four groups provided valid descriptions for all five components, while two groups did so for four components, and the other two groups for only three components. To illustrate the answers provided, Figure 11.5 presents word clouds corresponding to each component.

Finally, four groups provided feedback on the usefulness of the model. Three groups responded positively, while one group described the explanation of the model as *complicated*. This last group provided the following feedback:

“It was complicated to understand what was asked and the explanation of [TRACE] was really fast so we didn’t have time to understand.”

On the other hand, two groups specifically referred to the model’s ability to *structure* the design, while one group appreciated how it was *inclusive* in the sense that it took input from multiple stakeholders into account. This last response is provided below.

“Creates some structure. Allows to think about all points of view and not missing one. For example in marketing it is very important to think about the consumer desires and not only the product. This framework allows therefore to take on every stakeholder.”

## 11.5 Discussion and Implications

Our current findings are promising. While preliminary, results from our illustrative study suggest that our model helped students collaborate more efficiently by aligning the contributions of the different stakeholders and providing a structure with which to reason about the educational chatbot's implementation.

Aligning expectations of different holders was one of the challenges highlighted by Durall Gazulla et al. (2023), who specifically noted that one obstacle for collaboration they faced in the design of the *EDUguia* chatbot was the “challenge [of] addressing diverse needs, while ensuring the relevance of the solutions envisioned”. By providing specific roles to different

stakeholders, but inviting them to participate in the full design process in order to align their different needs, the TRACE model could help address this challenge.

Furthermore, the fact that the model was reported to provide *structure* for the design process addresses another challenge highlighted by Durall Gazulla et al. (2023). Namely, the challenge of translating research into practice. One outcome of a participatory design workshop structured with the TRACE model is a diagram that can serve as a blueprint to further design and develop the educational chatbot in question. While it is not a functioning chatbot, this outcome can be shaped into a sketch, a mockup, and even a prototype implementation. In essence, it serves as a way to translate the research outcome of the participatory design session into actionable tasks for the stakeholders who will implement this chatbot in practice.

However, it is important to note that one group described the model as *complicated* and that four groups did not provide valid descriptions for all components. Although misunderstandings of the TRACE model could be mitigated by providing more time to present the model, participatory design workshops are also limited by time constraints and include stakeholders with different backgrounds and technical aptitudes (Durall Gazulla et al., 2023). Hence, long presentations featuring abstruse terminology and complex definitions should be best avoided. Instead, ensuring that component definitions are clear and accessible to a wide variety of stakeholders could be crucial to maximize adoption in participatory design practices.

Our model could also help educators adapt their teaching practices in light of the impact large language models (LLMs) are having on education. It has recently been highlighted that “occupations in the field of education are likely to be relatively more impacted by advances in language modeling than other occupations” (Felten et al., 2023). The TRACE model could serve as a canvas for educators to collaborate with developers to better understand the opportunities and limitations of LLMs and with learners to better understand how learners envision using chatbots to support their studies.

These results also help us consolidate the work conducted throughout this thesis. First, the TRACE framework captures multiple lessons learned in the design and implementation of the educational chatbots that were used for our empirical studies. Second, certain elements of the components and processes comprising our conceptual model will inform the components that will form the technical blueprint that we will present in the following chapter. Third, TRACE allows us to reflect on the designs, implementations, and evaluations conducted in Part III of this thesis. These reflections could serve to adapt our designs for future evaluations. For example, we could explore supporting the code review tasks captured by our pedagogical scenarios with resources other than code snippets. Indeed, in the next chapter, we will see how *documentation* can be used as a learning resource to support learning tasks related to the code review process and how integrating this resource into code review notebooks requires a different application. Finally, by generalizing our approach to domains outside of software engineering education, the TRACE model serves to explore the applicability of our technological foundations, pedagogical scenarios, interaction strategies, and design processes



to a wider array of educational contexts.

## 11.6 Conclusion

The model proposed in this chapter addresses the lack of bespoke conceptual models for structuring the participatory design of chatbots. By outlining five different components that can be defined over a four-step process, TRACE breaks down the task of specifying how chatbots can support a given learning activity, which could be interpreted differently by different stakeholders. To maximize relevance in practice, the model places educators first, allowing them to define the pedagogical scenario that will guide the design process. Developers, due to their technical expertise, are also central in our model, ensuring that the pedagogical scenario can be supported by the technology in which the chatbot will be embedded, and bridging the scenario with the interactions that will occur between the chatbot and the learner. Finally, learners are tasked with identifying both the timing and content of these interactions, to ensure that the chatbot adds value to the learning experience, rather than being an element of distraction. The result is a model that can be used to produce a blueprint of how an educational chatbot could be integrated into a learning activity. This blueprint could be the input to future participatory design sessions, guide educators in adapting their lesson plans to make room for learner-chatbot interactions, or serve as a starting point for a technical specifications document or prototype.

Nevertheless, our study has limitations worth addressing. First, while we explicitly chose students from a software design course to maximize the number of participants that could play the role of the different stakeholders, our role-playing study is not indicative of how professional educators, developers, and learners might judge our conceptual model. Conducting a formal participatory design workshop with actual stakeholders could help improve the ecological validity of our proposed tool. Second, the limited time that participants had to interact might have affected their ability to efficiently understand and harness the model. Extending the workshop to a two or three-hour session could give participants time to assimilate the concepts presented in TRACE. We aim to address these limitations in future work.



## 12 Technical Blueprint

WE noted in Chapter 1 and in the introduction to Part IV of this thesis that often, the chatbot technologies and architectures that are applied to educational contexts are not necessarily designed for such contexts. While general-purpose chatbot technologies can be used in educational contexts, there are some challenges specific to these contexts that need to be taken into consideration. Namely, chatbot technologies intended for education should, by design, integrate directly into LXPs and focus on achieving learning goals by supporting learners with the task at hand. In this chapter, we propose a blueprint for an architecture specifically aimed at building task-oriented chatbots to support learners in educational contexts. This architecture aims to support collaboration during the technical implementation process by providing a common vocabulary for developers and educators looking to design educational chatbots. We then present a proof-of-concept implementation of our blueprint through *Text Analysis*, an app designed to support a wide range of pedagogical scenarios. Our blueprint is a first step toward an open chatbot architecture explicitly tailored for learning applications. On the one hand, this blueprint could serve as a starting point for developers in education looking to build chatbot technologies targeting educational contexts. On the other hand, it could also serve as a way for educators to get involved in the technical implementation of chatbots that would be best suited for their teaching practice.

By laying out a technical blueprint for a system that facilitates the integration of chatbots into apps, this chapter partially addresses **RQ4**:

- How can we support the collaborative design of educational chatbots?

The content of this chapter was partially presented in the following publication:

- Farah, J. C., Spaenlehauer, B., Ingram, S., & Gillet, D. (2022). A Blueprint for Integrating Task-Oriented Conversational Agents in Education. *4th Conference on Conversational User Interfaces (CUI 2022)*. <https://doi.org/10.1145/3543829.3544525>

### 12.1 Introduction

In the first paragraph of this thesis, we highlighted the fact that nowadays, chatbots come in a variety of architectures. While we can categorize chatbots by whether they are powered by rule-based, statistical data-driven, or end-to-end neural dialog systems (McTear, 2021), there are a wide variety of technological solutions and designs within each of these categories. This variety becomes even greater when elements of HCI are considered. Chatbots can be embodied, disembodied, speech or text-based, task-oriented or open-domain, and so on and so forth. This variety is also present when considering domain-specific chatbots, such as chatbots designed for educational contexts. To repeat the quote from Quiroga Pérez et al. (2020) that was presented in Chapter 1, “there exists as much technology used in the development of chatbots as there are educational chatbots”.

This lack of standard architectures and systems for designing, implementing, and deploying educational chatbots poses a challenge when translating the outcomes of design sessions—such as the workshops described in the previous chapter—into software solutions that can be exploited and evaluated. That is, developers and educators looking to implement these outcomes need to build software solutions from scratch or adapt these outcomes to integrate them using generic chatbot technologies. Indeed, this challenge was highlighted by Durall Gazulla et al. (2023), who underscored the complexity of translating research into practice when designing educational technologies. In their particular case, for example, their chatbot was meant to be integrated into a learning environment and its interactions were to be informed by pedagogical theory. Fulfilling those requirements with generic chatbot technologies might be infeasible, while building the technology from scratch requires complex programming skills.

To address these challenges, we propose a blueprint for a system that facilitates the integration of chatbots into digital learning apps and takes into consideration concepts specifically related to pedagogical contexts. Our blueprint is most aligned with work by Griol et al. (2013), who proposed an architecture for building chatbot interfaces in educational applications and incorporated it into their *Geranium* pedagogical system. Nevertheless, while Griol et al. focused on speech-based systems and questionnaire-based learning applications, our aim is to lay out an architecture that is agnostic with respect to the pedagogical scenario and the modality of the interaction. In this chapter, we provide the design of our blueprint, which is the seventh and final contribution put forth by this thesis (C7), as well as a proof-of-concept implementation of an app designed following this blueprint.

This chapter is structured as follows. In Section 12.2, we outline the design considerations that emerge from the identified challenges and motivate our proposed approach to address these requirements. We then detail our blueprint’s architecture in Section 12.3, defining the key building blocks, components, and processes that make up an interaction between a learner and a chatbot in an educational context. In Section 12.4, we present a proof-of-concept implementation of our architecture to showcase how it can support a learning activity. We then

discuss—in Section 12.5—how our proposed blueprint could serve as an initial step toward defining an open standard for integrating chatbots into applications aimed at educational contexts. We conclude, outlining limitations and future work, in Section 12.6.

## **12.2 Design Considerations**

We build on the aforementioned challenges to define the considerations that will inform the design of our blueprint. The guiding concern was to ensure that these chatbots were (i) integrated and (ii) task-oriented. For each of these aspects, we highlight the requirement that emerges from the literature and our proposed approach to address that requirement in our blueprint.

### **12.2.1 Integrated**

#### **Requirement**

Chatbots should integrate directly with learning applications (Yang et al., 2019).

#### **Proposed Approach**

Our blueprint is centered on the digital learning application that the chatbot is meant to support. This learning application provides the graphical and pedagogical context that will ground the chatbot's interaction. Furthermore, we propose a modular approach that is system-agnostic and does not require any specific technology stack. To maximize reuse, components are loosely coupled and can be composed as needed to support different learning applications. For example, the communication between the learning application and the rest of the architecture is handled by one component in particular. This component could be adapted in order to integrate different learning applications without the need to replace the rest of the architecture. Finally, while we cannot impose that implementations of our proposed blueprint be open-source, we strongly recommend it. Open-source development facilitates uptake, reuse, continuous improvements, and customization starting from a common codebase, and—as discussed in Chapter 2—is possibly favored by developers in education.

### **12.2.2 Task-Oriented**

#### **Requirement**

Chatbots should be designed to support learners in achieving well-defined learning goals (Kumar, 2021).

### Proposed Approach

Our blueprint requires that chatbots be equipped to perform one or more functions. Each function should be relevant to the context in which the chatbot is deployed. That is, a chatbot's function should take as input the specific learning activity that it is meant to support. To ensure useful and unobtrusive support and minimize the possibility of an unsolicited chatbot interfering with the learning process, a chatbot does not perform its task unless it is summoned by the learner. Furthermore, the chatbot takes into consideration feedback from the learner to improve the way it carries out or communicates the results of its performed function(s). This adaptive approach serves to personalize and enhance interactions with the chatbot and the resulting user experience over time. Privacy considerations related to the way personal data is handled by the chatbot should be transparent to the user.

### 12.3 Architecture

In this section, we outline the key building blocks, components, and processes that comprise our blueprint, as well as the way they come together to support a learning activity.

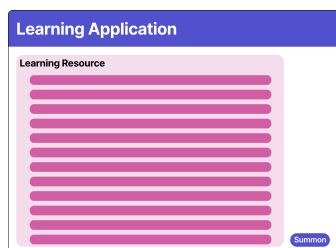


Figure 12.1: The learning application hosts the learning resource (pink) that will be the focus of the interaction and serves as the context for our blueprint.

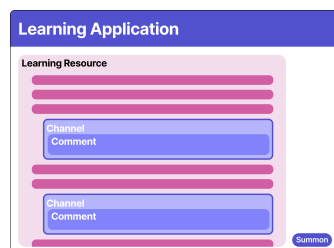


Figure 12.2: When summoned, the chatbot performs a function and opens channels (light purple) through which it can hold an interaction with the learner.

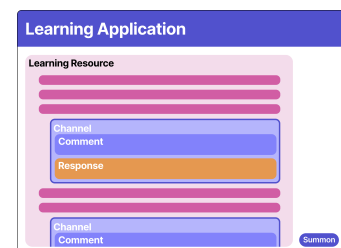


Figure 12.3: A learner can respond (orange) to the chatbot's comments to partake in the interaction that the chatbot has selected for a given channel.

#### 12.3.1 Building Blocks

To illustrate how our architecture fits with the educational context that we are targeting, we define a number of building blocks that make up our components. Specifically, these building blocks show how the chatbot is tasked with supporting learners during a learning activity by performing one or more specific functions. Here, we outline the key building blocks and provide illustrative examples.

**Learning Resource**

The *learning resource* constitutes the pedagogical context for the interactions between the chatbot and the learner. The chatbot should be able to perform its task using this learning resource. An example of a learning resource would be a code snippet that can be analyzed for potential issues.

**Function**

A *function* is performed by the chatbot given a specific pedagogical context, which serves as input to the function. The idea is that the function takes as its main input the learning resource and returns a set of talking points that serve to start the interaction between the learner and the chatbot. An example of a function would be a linter, which is designed to detect bugs and other issues in code (Johnson, 1978). A linter would take as input a code snippet and output the locations in the code snippet where it has detected an issue.

**Concept**

A *concept* refers to a subject that the chatbot can identify in the pedagogical context it is embedded in and about which it can subsequently converse. An example of a concept would be a code styling rule defining how learners should name variables. In JavaScript, for instance, the ESLint linter (Zakas, 2013) can detect when variables have not been written in camelCase. When analyzing a code snippet, the chatbot could detect issues in the code that are related to these concepts and eventually hold an interaction about them with the learner.

**Learning Graph**

Each concept has a corresponding *learning graph*, which represents the states that a learner can be in with respect to the concept at hand. A learner can transition between the different states in the graph by going through an interaction with the chatbot. An example of a learning graph would be a Markov chain with two states (*Don't Know* and *Know*) that indicate whether the learner has understood a concept or not. In its simplest form, this learning graph would not include forgetting, that is, learners can only transition from *Don't Know* to *Know*.

**Channel**

A *channel* is a location in the graphical user interface (GUI) where an interaction between the chatbot and the learner can take place. Channels can be opened by the chatbot after it has performed its function. Once the interaction is complete, the channel is closed. An example of a channel would be a line number in a code snippet. If a chatbot finds an issue in the code snippet, it could open a channel linked to the line number where the issue was located. A dialog box would then appear below the respective line number, allowing the learner and the

## Chapter 12. Technical Blueprint

Table 12.1: An Overview of Our Blueprint's Key Components and Building Blocks Alongside the Functionalities They Support

Component	Building Blocks	Functionalities
Interaction Model	Concepts	Power the dialog between the learner and the chatbot.
Learner Model	Concepts Learning Graphs	Track the state of the <i>learning graph</i> for each <i>concept</i> .
Interface Engine	Learning Resource Channels Feedback	Provide graphical and pedagogical interfaces to interact with the learning application.
Task Engine	Function Concepts	Execute the chatbot's <i>function</i> . Transmit the list of <i>concepts</i> to the <i>knowledge engine</i> .
Knowledge Engine	Concepts Learning Graphs Interaction Models Learner Model	Create a <i>learning graph</i> for each <i>concept</i> given by the <i>task engine</i> . Create a <i>learner model</i> from the <i>learning graphs</i> . Manage the <i>learner model</i> and the <i>interaction models</i> . Update the models when notified. Make knowledge accessible to the <i>interaction</i> and <i>learning engines</i> .
Interaction Engine	Channel Interaction Model Learner Model	Select the <i>interaction models</i> based on the <i>learning graphs</i> . Process responses posted to the <i>channels</i> . Select the next interaction from the <i>interaction model</i> . Notify the <i>knowledge engine</i> of possible updates to the models.
Learning Engine	Feedback	Use <i>feedback</i> to update the strategies used by the <i>interaction engine</i> .

chatbot to hold an interaction. Figures 12.1–12.3 depict how channels can be opened in the context of a learning resource and then serve to hold an interaction between the user and the chatbot.

### Feedback

*Feedback* is a way to signal the quality of an interaction the learner has had with the chatbot. It can be implicit or explicit. Examples of implicit feedback would be whether the learner engaged with the chatbot at all or whether the chatbot was able to achieve its *intent*. Examples of explicit feedback would be emoji reactions that the learner can select, a flag to report an issue with a comment made by the chatbot, or a button that the learner can click on to request that the educator intervene in the interaction.

### 12.3.2 Components

Our blueprint's components build on the blocks presented in Section 12.3.1 to provide the functionalities needed for the chatbot. We divide these components into two types: (i) models and (ii) engines. Models are concerned with keeping the state of the chatbot's interactions with a learner. Engines are concerned with executing the blueprint's processes, which we will outline in Section 12.3.3. A summary of the components and their building blocks is given in Table 12.1.



### Interaction Model

An interaction takes place within a channel and consists of a natural language dialog that the learner can hold with the chatbot. Each interaction is related to a concept and has an *intent*, which defines the state that the chatbot would like the learner to arrive at. Our blueprint does not enforce the exact technical implementation of the dialog employed by the chatbot to fulfill its intent. This allows for flexibility in the selection of the scripting and/or NLP technologies that will power the dialog between the chatbot and the learner. The *interaction model* also has one or more end states, which indicate that the interaction is over. These end states serve as a way to keep the learner focused on the task at hand. An example of a simple interaction model would be a tree-based script in which the chatbot explains a given concept to the learner and then prompts the learner with questions aimed at verifying whether the learner has transitioned from the *Don't Know* to the *Know* state.

### Learner Model

The *learner model* is meant to summarize a learner's grasp of the concepts that the chatbot is designed to support the learner with. When the chatbot performs its function, it will initialize a learner model. For each concept, it will then select the appropriate active state of that concept's learning graph based on the output of the function. For example, if a JavaScript code snippet contains variables that have not been written in camelCase, the resulting learner model generated by the chatbot will contain a learning graph for the camelCase concept where *Don't Know* is the currently active state.

### Interface Engine

The *interface engine* is responsible for connecting the chatbot and the GUI that the chatbot can use to interact with the learners. Some of the functionalities that it handles include: (i) opening the appropriate channels, (ii) posting the chatbot's comments, (iii) exposing the graphical elements required for the learner to respond and provide feedback, and (iv) providing the learner with a way to summon the chatbot.

### Task Engine

The *task engine* is in charge of executing the chatbot's function and translating the output of the function to the list of concepts that will guide the interactions between the chatbot and the learner.

### Knowledge Engine

The *knowledge engine* keeps track of the state of the learner and the interactions between the learner and the chatbot. That is, the knowledge engine is in charge of updating and tracking

the learner and interaction models. Every time the interaction engine processes a learner response, the knowledge engine is notified of any updates that need to be processed. The knowledge engine also communicates with the interaction engine and the learning engine, providing them with the information they need to carry out their tasks.

### Interaction Engine

The chatbot's *interaction engine* is in charge of selecting the appropriate interaction model for a channel given the state of the learner model. The interaction engine can be configured to follow different strategies and communicates with the learning engine to improve or update these strategies. The interaction engine is also in charge of processing the learner's interaction with the chatbot. Every time a learner posts a response or provides feedback, the interaction engine processes these actions accordingly and passes the result on to the knowledge engine so that the knowledge engine can update the learner and interaction models.

### Learning Engine

The *learning engine* is in charge of updating the strategies guiding the chatbot's interaction engine based on feedback from the learner. This allows the chatbot to refine its behavior at the end of a session. Over time, the learning engine should allow chatbots to better select the interaction model used to interact with learners based on the learner's learner model.

#### 12.3.3 Processes

Our blueprint's processes outline the actions that can take place when there is an interaction between a learner and the chatbot. For clarity, we focus on a chatbot interacting with a single learner in a single session. An overview of these processes and how they span the different components of our architecture is shown in Figure 12.4.

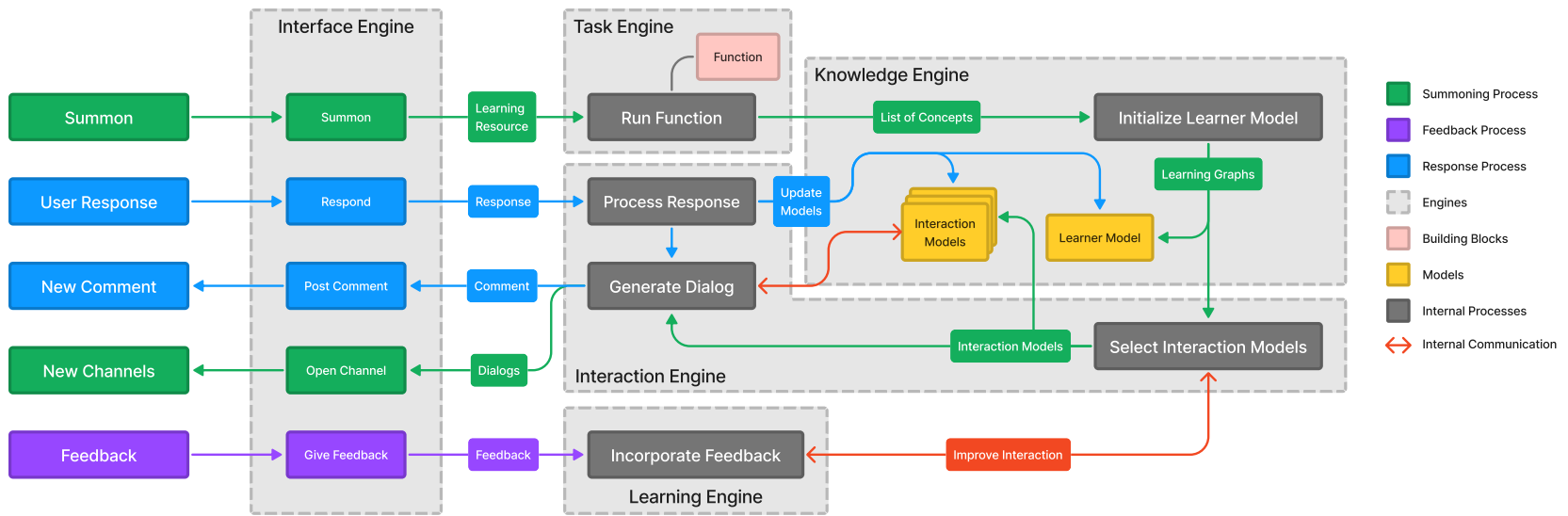


Figure 12.4: Our blueprint consists of multiple loosely coupled components, comprising both engines, which are tasked with executing processes, and models, which keep track of the system's state. Here, we illustrate how the different components and processes come together to support the interaction between a learner and a chatbot in the context of a learning activity.

### Summon Chatbot

To start interacting with the chatbot, a learner summons the chatbot by asking it to perform its function. This could be done through a button on the GUI of the learning application (see Figure 12.1). When the learner clicks on this button, the interface engine will notify the task engine that it should start the process of executing the function. The task engine will execute the function and pass the function's result—a list of concepts—to the knowledge engine. The knowledge engine will then create the learner model based on the output of the task engine. This model is then passed on to the interaction engine, which will select the most appropriate interaction model for each of the concepts that the chatbot will tackle with the learner. These interaction models are then passed over to the interface engine, which will open the appropriate channels for the interactions to take place. Figure 12.2 shows the newly opened channels within the learning resource.

### Hold Interaction

Once a channel is open, the learner can hold an interaction with the chatbot. The interaction is guided by the interaction model selected for that channel and always starts with a message posted by the chatbot. The learner should be able to respond to the initial message through one or more of the following affordances: (i) via natural language through a text input box, (ii) via emoji reactions, and (iii) via quick reply buttons. Text inputs and emoji reactions are commonly available in messaging services and platforms (e.g., WhatsApp (Acton et al., 2009), Slack (Butterfield et al., 2013)), while supporting quick reply buttons has been proposed as a best practice when designing chatbot interfaces (Ferman Guerra, 2018). Once the learner responds via any of these affordances (as in Figure 12.3), the interface engine will forward the response to the interaction engine, which will process the response and select the appropriate next message to post based on the interaction model and its underlying dialog system. The interaction engine will also inform the knowledge engine of the update in the interaction model so that the knowledge engine can update the learner model if the learner has gone through a state transition in the learning graph corresponding to the given interaction's concept. The exchanges between the learner and the chatbot—as well as the corresponding updates—continue until the interaction model reaches an end state.

### Provide Feedback

Once an interaction is over, the learning engine analyzes the interaction to draw conclusions about the quality of the interaction. This is done by analyzing any explicit or implicit actions the learner took throughout the interaction. Based on this analysis, the learning engine provides feedback to the interaction engine. The interaction engine can then use this feedback to change its preferred strategy with respect to a given interaction.

## 12.4 Implementation

To provide a proof-of-concept implementation of our blueprint, we used it to design and implement an app that could (i) highlight words in text provided by the educator and (ii) allow learners to interact with a chatbot with respect to a particular highlighted word. We selected this app for our proof of concept given its applicability to a wide array of domain-agnostic learning activities, including (i) helping learners understand difficult words, (ii) providing learners with further information about complex topics, and (iii) supporting various language learning tasks (e.g., listing synonyms, detecting spelling mistakes).

Our implementation addressed the following research question, which is an extension of **RQ4**:

- Can our technical blueprint support the design and implementation of an app aimed at integrating educational chatbots into LXPs? (**RQ4b**)

In this section, we present the learning scenario we aimed to support as well as the technical implementation of the chatbot integration. Where relevant, we highlight the building blocks and components presented in Section 12.3 as they appear in the implementation.

### 12.4.1 Pedagogical Scenario

While the app we were aiming to design is applicable to a wide array of pedagogical scenarios, to provide a proof-of-concept implementation in line with our previous empirical studies, we focused on the technopedagogical scenario provided by code review notebooks. As explored throughout this thesis, performing code reviews principally consists of a developer submitting code for review and then another developer reviewing the submitted code by adding comments to specific lines of code. Nevertheless, there are other tasks involved in performing code reviews, such as understanding the documentation of a particular library or API that is being used in a code snippet. This documentation is often extensive and complicated for beginner programmers to follow. Hence, educators could select parts of the documentation that they want learners to understand, highlighting terms and concepts that are relevant to a particular code review task. Our app was therefore conceived as a way to allow educators to curate a snippet of text (i.e., the *learning resource*), select a set of keywords (i.e., *concepts*) that the chatbot should be able to highlight (i.e., *function*), and configure the chatbot to hold exchanges centered on those keywords. We named this app *Text Analysis*.

### 12.4.2 Design

Having defined our pedagogical scenario, we began our design process by creating mockups. These mockups served to illustrate how the text would be presented to the learner and how the three main processes outlined in Section 12.3.3 would be supported.

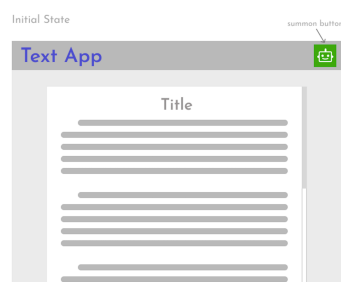


Figure 12.5: Before any interaction takes place, the app features a text snippet selected by the educator.

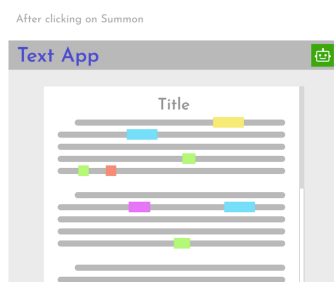


Figure 12.6: When summoned, the chatbot highlights the keywords in the text.

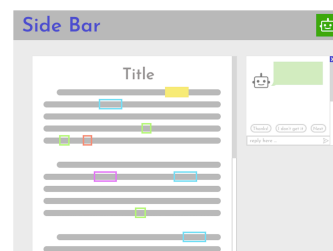


Figure 12.7: Clicking on a highlighted keyword opens a channel in which a learner can hold an interaction with the chatbot.

As shown in Figure 12.5, the app features the text in a central pane. The learner can then summon the chatbot by clicking on a button in the header. This action triggers the chatbot's function, which is to highlight the keywords contained in the text, as shown in Figure 12.6. The learner can then click on one of the keywords to open a channel in which they can hold an interaction with the chatbot. This action opens a channel on the sidebar, as shown in Figure 12.7. After completing an interaction, learners can provide feedback about this interaction by clicking on one of the predefined quick reply buttons shown above the *Reply Here* input box in Figure 12.7.

### 12.4.3 Technical Implementation

We implemented our design following the architecture presented in Section 12.3. Once again, we harnessed the Graasp ADK to create an app compatible with Graasp. As discussed, this app exposes a text snippet (i.e., the *learning resource*) potentially containing some keywords (i.e., *concepts*) that the educator has selected. The chatbot's task is to support the learner in understanding those keywords by highlighting them in the text and providing more information about them. For this implementation, the chatbot is equipped with a highlighter (i.e., the *function*) able to detect a set of keywords by parsing the text using regular expressions and highlighting these keywords by annotating them with HTML. Each keyword is associated with a simple *learning graph* in which a learner can either understand the keyword or not. These keywords are selected by the educator, who can configure the app through Graasp's educator interface (see Figure 12.8).

Once configured, the text snippet predefined by the educator is shown to the learner, as depicted in Figure 12.9. Whenever a learner is ready to analyze the text, they can summon the chatbot by clicking a button. This button will start the *summoning process*. The *interface engine* will fetch the text and send it to the highlighter for analysis. The highlighter will detect the keywords present in the text snippet and annotate them with HTML so that they are

## Prepare Your Lesson

☒ use chatbot

Enter the lesson title

Excerpts from the ESLint Documentation

SAVE

Enter the text students will see

## Command Line Interface

The ESLint CLI is a command line interface that lets you execute `linting` from the terminal. The CLI has a variety of options that you can pass to its commands.

## Node.js API

While ESLint is designed to be run on the command line, it's possible to use ESLint programmatically through the Node.js API. The purpose of the Node.js API is to allow plugin and tool authors to use the ESLint functionality directly, without going through the command line interface.

Note: Use undocumented parts of the API at your own risk. Only those parts that are specifically mentioned in this document are approved for use and will remain stable and reliable. Anything left undocumented is unstable and may change or be removed at any point.

SAVE

Enter the initial prompt describing the conversation (as a template for {{keyword}})

This is a conversation between a student and a chatbot that can help the student understand what {{keyword}} means. After each response, the chatbot gives the student one or two options to continue the conversation.

SAVE

Enter the chatbot's first line (as a template for {{keyword}})

You clicked on {{keyword}}. What would you like to know about this keyword? For instance, I can provide you with its definition, additional information about its relation to ESLint, or highlight specific examples.

SAVE

Enter keyword: definition

SAVE

- api : An application programming interface (API) is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a connection or interface is called an API specification. A computer system that meets this standard is said to implement or expose an API. The term API may refer either to the specification or to the implementation.
- cli : A command-line interpreter or command-line processor uses a command-line interface (CLI) to receive commands from a user in the form of lines of text. This provides a means of setting parameters for the environment, invoking executables and providing information to them as to what actions they are to perform. In some cases the invocation is conditional based on conditions established by the user or previous executables. Such access was first provided by computer terminals starting in the mid-1960s. This provided an interactive environment not available with punched cards or other input methods.
- node.js : Node.js is an open-source server environment. Node.js is cross-platform and runs on Windows, Linux, Unix, and macOS. Node.js is a back-end JavaScript runtime environment. Node.js runs on the V8 JavaScript Engine and executes JavaScript code outside a web browser.
- plugin : In computing, a plug-in (or plugin, add-in, addin, add-on, or addon) is a software component that adds a specific feature to an existing computer program. When a program supports plug-ins, it enables customization.

Figure 12.8: The educator can configure the app and the chatbot interaction using Graasp's educator interface.

displayed with different colors. Note that each keyword corresponds to a different color, as shown in Figure 12.10.

In our current implementation, the *interaction engine* only supports one *interaction model* at a given time. This interaction model is defined by the educator and consists of a prompt configured following the configuration model presented in Chapter 10. That is, the interaction model and the *intent* are captured using natural language by a prompt. This prompt is then sent to OpenAI's API, following the architecture presented in Chapter 10. The intent is defined by the educator, who can—for example—specify the following within the prompt:

*The objective of this interaction is to have the learner acknowledge that they understand the meaning of {{keyword}}.*

Note that the interaction models will be different for each keyword, given that the educator can use a placeholder ({{keyword}}) to refer to the keyword in question. The interaction models are then passed back to the interface engine, which prepares a *channel* for each of these interactions to take place on the sidebar, as illustrated in Figure 12.11.

The screenshot shows the Graasp application interface. At the top, there's a blue header bar with the Graasp logo, navigation icons, and the text 'ESLint Lesson' next to a heart icon. On the right of the header is a user profile for 'Juan Carlos Fa...'. Below the header, a sidebar on the left contains a navigation menu with items like 'ESLint Lesson', 'Getting Started', 'Introduction', 'ESLint', 'Styling', 'Best Practices', 'Exercise', 'Solutions', 'Chatbots', and 'Conclusion'. The main content area is titled 'Excerpts from the ESLint Documentation' and contains three sections: 'Command Line Interface', 'Node.js API', and 'Using Configuration Files'. Each section has a title and a paragraph of text. The 'Command Line Interface' section describes the ESLint CLI. The 'Node.js API' section explains how to use ESLint programmatically. The 'Using Configuration Files' section discusses configuration files like .eslintrc and package.json. The 'Cascading and Hierarchy' section explains how configurations are merged. At the top right of the main content area, there are two buttons: 'HIDE KEYWORDS' and 'SHOW KEYWORDS'.

**Command Line Interface**

The ESLint CLI is a command line interface that lets you execute linting from the terminal. The CLI has a variety of options that you can pass to its commands.

**Node.js API**

While ESLint is designed to be run on the command line, it's possible to use ESLint programmatically through the Node.js API. The purpose of the Node.js API is to allow plugin and tool authors to use the ESLint functionality directly, without going through the command line interface.

Note: Use undocumented parts of the API at your own risk. Only those parts that are specifically mentioned in this document are approved for use and will remain stable and reliable. Anything left undocumented is unstable and may change or be removed at any point.

**Using Configuration Files**

There are two ways to use configuration files.

The first way to use configuration files is via `.eslintrc.*` and `package.json` files. ESLint will automatically look for them in the directory of the file to be linted, and in successive parent directories all the way up to the root directory of the filesystem (`/`), the home directory of the current user (`~/`), or when `root: true` is specified. See Cascading and Hierarchy below for more details on this. Configuration files can be useful when you want different configurations for different parts of a project or when you want others to be able to use ESLint directly without needing to remember to pass in the configuration file.

**Cascading and Hierarchy**

The configuration cascade works based on the location of the file being linted. If there is a `.eslintrc` file in the same directory as the file being linted, then that configuration takes precedence. ESLint then searches up the directory structure, merging any `.eslintrc` files it finds along the way until reaching either a `.eslintrc` file with `root: true` or the root directory.

In the same way, if there is a `package.json` file in the root directory with an `eslintConfig` field, the configuration it describes will apply to all subdirectories beneath it, but the configuration described by the `.eslintrc` file in the `tests/` directory will override it where there are conflicting specifications.

Figure 12.9: The learning task focuses on a snippet of text that is provided by the educator.

The learner can now hold an interaction with the chatbot through any of the channels that have been opened. At each step of the interaction, the learner can respond to the chatbot using free-form text (see Figure 12.11). The interaction model interprets the response to advance the dialog until the learner stops interacting or reaches the maximum number of messages allowed per channel. Currently, learning graph transitions are not performed automatically. That is, the knowledge engine does not automatically update the learner model from the *Don't Know* to the *Know* state. This can be done manually by the educator, who has access to the conversations that learners held with the chatbot. The exercise continues until the learner completes all the interactions that were opened by the chatbot or the learner closes the window hosting the application.

### 12.5 Discussion and Implications

The implementation of Text Analysis following the blueprint outlined in this chapter is a first step towards consolidating the proposed architecture. Our blueprint lays out the key building blocks, components, and processes that a system following our architecture would have to



The screenshot shows the Graasp application interface. At the top, there's a header with the Graasp logo, navigation icons, and the title 'ESLint Lesson' next to a heart icon. On the right of the header is a user profile for 'Juan Carlos Fa...'. Below the header, on the left, is a sidebar with a table of contents for the 'ESLint Lesson', including items like 'Getting Started', 'Introduction', 'ESLint', 'Styling', 'Best Practices', 'Exercise', 'Solutions', 'Chatbots', and 'Conclusion'. The main content area is titled 'Excerpts from the ESLint Documentation'. It contains three sections: 'Command Line Interface', 'Node.js API', and 'Using Configuration Files'. In the 'Node.js API' section, the text is highlighted with blue boxes around the words 'Node.js' and 'API'. Below this, there's a 'Note' about using undocumented parts of the API. The 'Using Configuration Files' section describes two ways to use configuration files. The 'Cascading and Hierarchy' section explains how the configuration cascade works. At the bottom of the main content area, there's a button labeled 'HIDE KEYWORDS' and another labeled 'SHOW KEYWORDS'.

Figure 12.10: When the learner clicks on the *Highlight Keywords* button, the summoning process is executed and the chatbot highlights the keywords selected by the educator.

include. By abstracting as many details as possible, this blueprint could potentially serve as a first step in implementing a standard architecture for integrating task-oriented conversational agents in education. Furthermore, the fact that Text Analysis can be easily used outside of software engineering education illustrates how our blueprint is not closely coupled with a specific educational domain. Indeed, while the pedagogical scenario that motivated our implementation is framed using code review notebooks, we can extend this to any pedagogical scenario that can incorporate an activity mediated by highlighting keywords in text.

Most importantly, our blueprint supports collaboration between the different stakeholders involved in the design of educational chatbots. As explained in Chapter 11, Text Analysis was developed through a semester-long case study with an undergraduate student from the School of Computer and Communication Sciences at the École Polytechnique Fédérale de Lausanne, Switzerland. A part of this app's design process included brainstorming sessions supported by the TRACE model. Outcomes from these sessions were then used to inform the app's implementation. Specifically, we can highlight how concepts from our blueprint

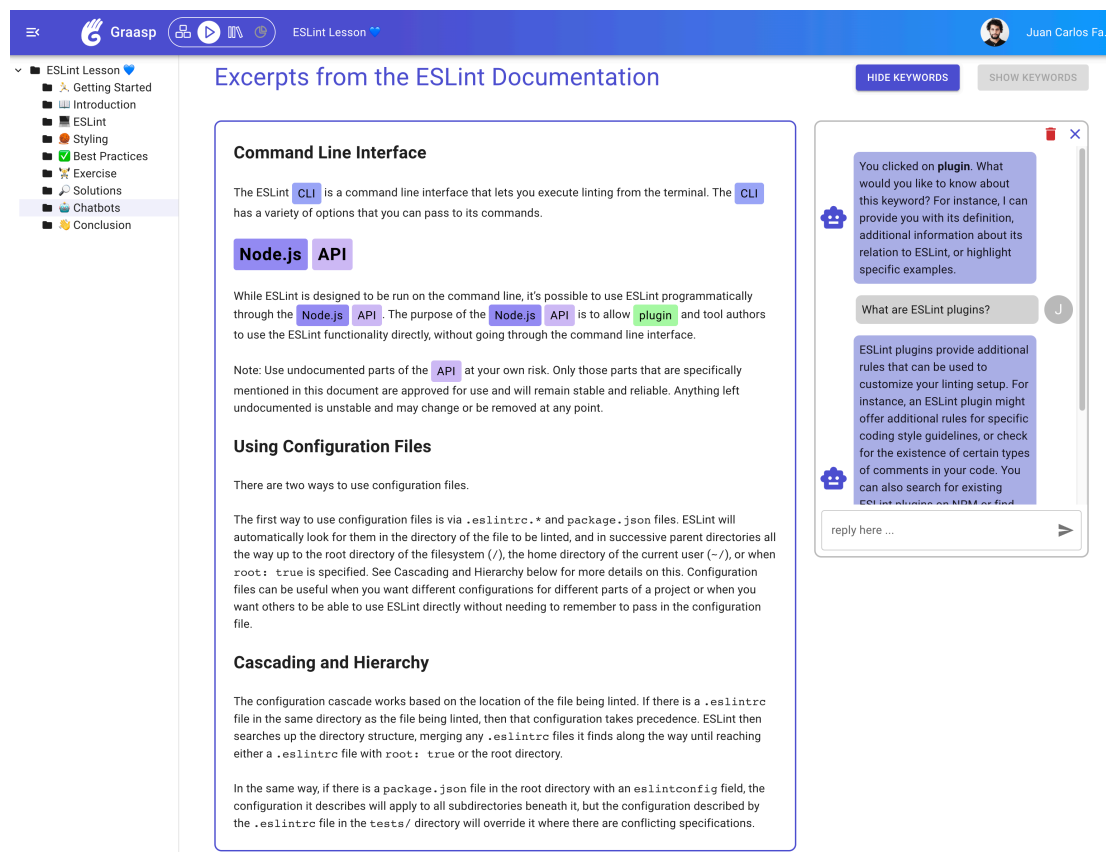


Figure 12.11: Once the channels are open, the learner and the chatbot can hold interactions about the keywords. These interactions are powered by OpenAI's API.

(e.g., learning resources, channels, and interaction models) map to components from the TRACE model (e.g., resources, cues, and exchanges). In fact, our blueprint nicely captures the processes through which components interact. For example, the process of summoning the chatbot illustrates how cues are presented in the learning resource, while holding an interaction illustrates how an exchange can take place to support the learning task at hand. As such, this proof-of-concept implementation also validates the applicability of our blueprint in translating outcomes from design sessions into software.

These results also helped us consolidate the work conducted throughout this thesis. First, by showing how outcomes from design sessions using the TRACE framework can be implemented in software and incorporated into pedagogical scenarios, our blueprint illustrates how we can bridge the gap between research and practice and provides links between the pedagogical scenario presented in Chapter 5, the architecture and configuration model presented in Chapter 10, and the conceptual model presented in Chapter 11. Second, the fact that Text Analysis was bootstrapped using the Graasp ADK shows how apps built with an implementation of the ADA presented in Chapter 2 can feature chatbots by design. Future apps built with the Graasp ADK can follow the same strategy to support educational chatbots.

Furthermore, this blueprint could be integrated by default into the ADA in order to facilitate the development of this chatbot support. Finally, our blueprint builds on our experience implementing educational chatbots for software engineering education and serves to generalize our contributions to other domains. In doing so, our blueprint broadens the impact that the technological foundations, pedagogical scenarios, interaction strategies, and design processes presented in this thesis could have on supporting the integration of chatbots into digital education platforms.

## **12.6 Conclusion**

The blueprint proposed in this chapter is designed to address the challenge highlighted in Section 12.1, namely to support translating the outcomes of design sessions in order to implement educational chatbots that can be directly integrated into learning applications. To test the applicability of our blueprint, we used it to integrate a chatbot into an app that supports students learning programming best practices. This implementation shows that the building blocks of our blueprint covered the needs of our use case application and facilitated its design and implementation.

Nevertheless, the current version of the blueprint has some limitations that should be highlighted. First, for this chapter, we focused on the case of a single learner interacting with a chatbot over the course of one learning activity or session. Further components are required to handle a learner's interaction with the same chatbot over multiple sessions, as well as interactions supporting multiple learners. Second, so far, we have only validated the applicability of our blueprint with our implementation of Text Analysis. Our aim is to target proof-of-concept implementations in diverse learning contexts and use case scenarios. Third, the need to support educators in the configuration and deployment of chatbots built with our blueprint was not addressed. An extension of our blueprint that incorporates mechanisms for configuration, testing, and deployment would ensure that educators can easily integrate such chatbots into their practice. We aim to further develop our blueprint to address these limitations and consolidate its architecture in future work.



## 13 Conclusion

THE presence of chatbots in educational contexts has been steadily increasing over the past decade. This widespread interest in integrating chatbots into pedagogical scenarios has driven research highlighting the tangible benefits and promising applications of educational chatbot technologies. Nonetheless—as illustrated throughout this thesis—designing, developing, deploying, and evaluating educational chatbots is a complex process that involves several stakeholders and faces a number of challenges across various dimensions, including technological, pedagogical, interaction, and design dimensions. To help address some of these challenges, the overarching objective of this thesis was to design, implement, and validate a conceptual framework for integrating conversational agents into domain-specific learning contexts (**O★**). Our approach toward this objective was incremental.

We started by laying the technological foundations to support the technical solutions that would be used in our empirical studies with educational chatbots. We then designed the pedagogical scenarios in which we embedded these chatbots, specifically focusing on supporting the code review process in software engineering education. In a series of empirical studies, we then explored different strategies that educational chatbots could harness in their interactions with learners. Finally, we investigated how to support collaboration among stakeholders, both in the conceptual design and in the technical implementation of educational chatbots. To summarize the work conducted in this thesis, we revisit the diagram illustrated in Figure 13.1, which was presented in Chapter 1.

As highlighted in Figure 13.1, our review of the research context allowed us to identify research opportunities at the intersections of the fields of digital education, software engineering, human-computer interaction (HCI), and artificial intelligence (AI). This review highlighted opportunities to study the integration of educational chatbots into learning experience platforms (LXPs) (**RO1**), develop tools to support software engineering education (**RO2**), create models to guide a chatbot's design (**RO3**), and incorporate the functionalities offered by large language models (LLMs) in the design of educational chatbots (**RO4**). In turn, addressing these opportunities informed our problem statement, which identified a lack of conceptual and technical design tools for integrating educational chatbots into domain-specific learning

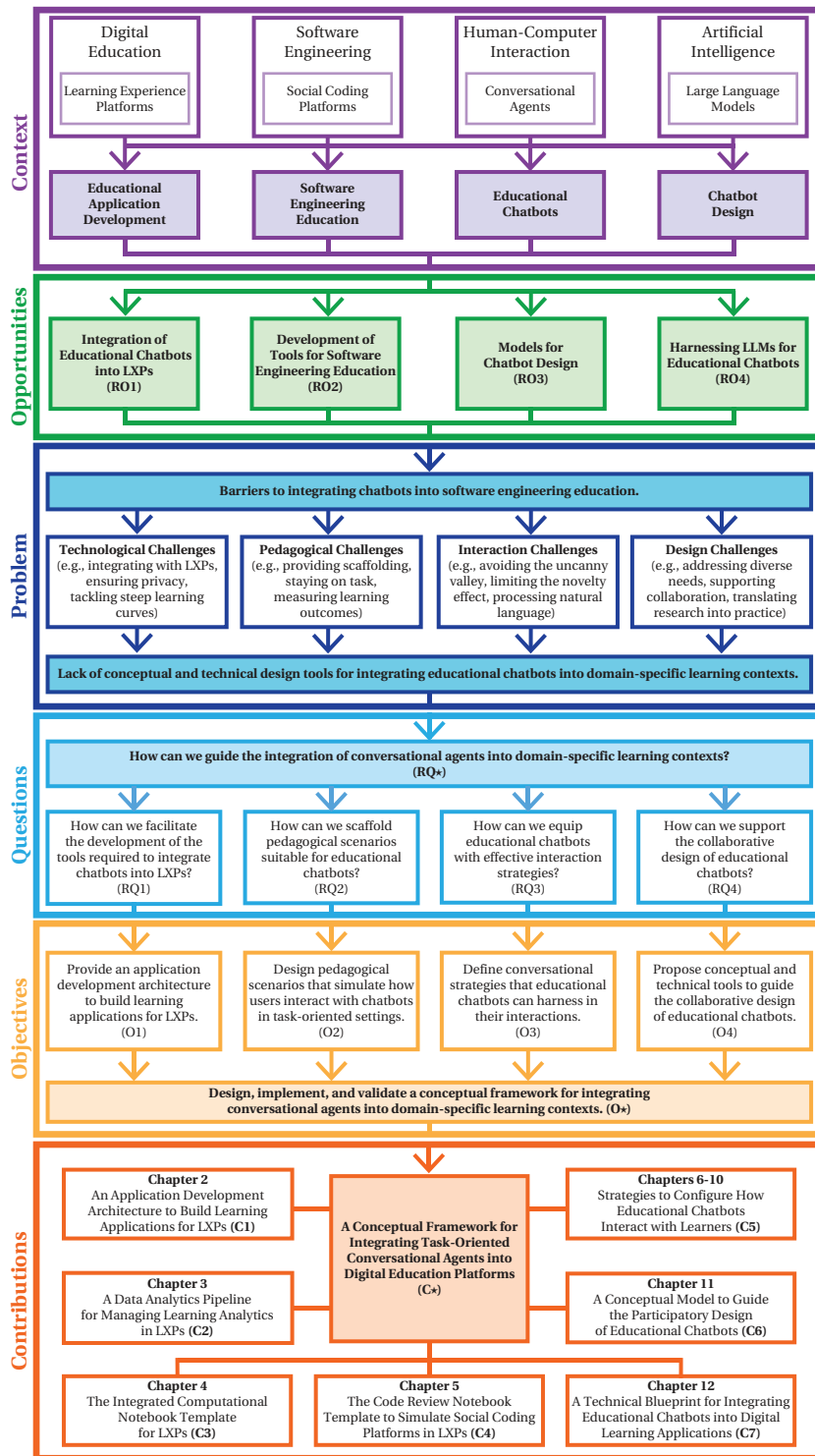


Figure 13.1: An Overview of Our Research Context, Research Opportunities, Problem Statement, Research Questions, Objectives, and Contributions

contexts. With this problem statement in mind, we formulated the following overarching research question:

- How can we guide the integration of conversational agents into domain-specific learning contexts? **(RQ★)**

To approach this overarching research question, we broke it down into four—more focused—research questions, each addressing a different set of challenges that we had identified in the literature.

- *Technological Challenges*: How can we facilitate the development of the tools required to integrate chatbots into LXPs? **(RQ1)**
- *Pedagogical Challenges*: How can we scaffold pedagogical scenarios suitable for educational chatbots? **(RQ2)**
- *Interaction Challenges*: How can we equip educational chatbots with effective interaction strategies? **(RQ3)**
- *Design Challenges*: How can we support the collaborative design of educational chatbots? **(RQ4)**

Tackling each of these questions in order to support educational chatbot design constitutes the overarching contribution put forth in this thesis:

- A conceptual framework for integrating task-oriented conversational agents into digital education platforms. **(C★)**

Guided by our research questions, our framework captures four dimensions of educational chatbot design: (i) *technological foundations*, (ii) *pedagogical scenarios*, (iii) *interaction strategies*, and (iv) *design processes*. Within each dimension, our framework features one or more individual contributions. In the following section, we summarize these contributions, outline how they address their respective research question, and link them to our overarching conceptual framework.

## 13.1 Contributions

Our overarching contribution—as mentioned above—is a conceptual framework for integrating task-oriented conversational agents into digital education platforms. This overarching contribution comprised seven individual contributions that together constitute the framework. As a whole, our framework is a complex design object covering the aforementioned four

dimensions of educational chatbot design. Each of these dimensions was addressed in its corresponding part of this thesis. In this section, we review our individual contributions and specify how they are connected and related to our overarching contribution.

### 13.1.1 Technological Foundations

We started in Part I of this thesis by addressing technological challenges through our first research question: *How can we facilitate the development of the tools required to integrate chatbots into LXPs?* Our approach to this question resulted in two contributions: (i) an application development architecture (ADA) to support the implementation of apps that can be integrated into LXPs (C1) and (ii) an end-to-end LA pipeline to support the full data management cycle for research in education (C2).

The ADA defined key components and a software engineering process to facilitate the development of apps specifically aimed at deployment within a digital learning environment. We validated this architecture through the proof-of-concept implementation of the Graasp ADK, which we then evaluated by conducting semi-structured interviews with 12 developers who used the ADK to create apps for education (Chapter 2). Moreover, we used the Graasp ADK to develop the apps used in our empirical evaluations, including those designed to host educational chatbots.

The LA pipeline was aimed at supporting open research in education. The pipeline's architecture defined five stages to structure how data generated by educational platforms should be captured, visualized, extracted, and anonymized. We validated the pipeline through a proof-of-concept implementation on Graasp (Chapter 3). Throughout this thesis, we employed the features implemented in our proof of concept to collect, extract, and manage the data used in our analyses.

To answer our first research question: we can facilitate the development of the tools required to integrate chatbots into LXPs through standard architectures to support developers in creating apps that can host educational chatbots and pipelines that can help manage the data required to power and evaluate these chatbots.

### 13.1.2 Pedagogical Scenarios

In Part II of this thesis, we addressed pedagogical challenges through our second research question: *How can we scaffold pedagogical scenarios suitable for educational chatbots?* Our approach to this question resulted in two contributions: (i) the integrated computational notebook template to provide browser-based computational notebook solutions within digital education platforms (C3) and (ii) the code review notebook template to simulate social coding platforms in educational contexts (C4).

The integrated computational notebook template aimed to make computational notebooks



more accessible for less technical learners and educators by supporting technopedagogical scenarios that are fully encapsulated within digital education platforms. Integrated computational notebooks center around *Code*, an app that leverages the *Pyodide* library to execute Python directly on the browser without any additional dependencies. We validated this template in an eight-week computational thinking course comprising 67 university students (Chapter 4). We also used integrated computational notebooks to embed and evaluate educational chatbots following rule-based scripts (Chapter 9).

The code review notebook template resembles a computational notebook but focuses on reviewing and annotating code, and can guide educators in creating technopedagogical scenarios for teaching subjects related to programming education through the use of the code review process. Code review notebooks center around *Code Review*, an app focused on introducing learners to the code review process. We validated this template through a case study comprising an online lesson on code quality standards completed by 25 university students (Chapter 5). Furthermore, we used code review notebooks to evaluate educational chatbots supported by the Wizard of Oz technique (Chapter 8) and by LLMs (Chapter 10).

To answer our second research question, we can scaffold pedagogical scenarios suitable for educational chatbots by designing templates that are informed by how chatbots are used in industry and feature apps that can host the interactions chatbots will have with learners. Educators can then use these templates to guide the design of technopedagogical scenarios that best fit their teaching practices.

### 13.1.3 Interaction Strategies

In Part III of this thesis, we addressed interaction challenges through our third research question: *How can we equip educational chatbots with effective interaction strategies?* Our approach to this question resulted in one contribution comprising five different strategies that could be used to configure how educational chatbots interact with learners (C5). These strategies addressed both the content (humor, emoji) and mechanisms (Wizard of Oz, rule-based scripts, LLMs) shaping the interactions between learners and chatbots in educational settings.

The first strategy consisted in equipping chatbots with a way of exploiting humor arising from ambiguity or incongruity. We evaluated this strategy through an online experiment with 400 participants to explore the use of a chatbot to harness such humor (Chapter 6). Findings from this experiment suggested that chatbots could indeed harness humor from ambiguous grammatical constructs to strengthen anthropomorphic perceptions of traits such as personality and friendliness.

The second strategy involved incorporating an emoji interface through which learners could potentially signal their reaction to a particular comment from a chatbot. To understand the effect that this strategy could have, we conducted an observational study comprising over 54

million GitHub comments (Chapter 7). Our analysis showed that some reaction types are not equally distributed across human and bot comments and that a chatbot's design and purpose influence the types of reactions it receives.

The third strategy explored the effects of impersonating educational chatbots following the Wizard of Oz technique. We evaluated this strategy in a controlled in-class experiment comprising 30 undergraduate software engineering students (Chapter 8). On the one hand, our quantitative analysis did not yield significant differences between conditions for any of the aspects considered. On the other hand, our qualitative results suggested that students expect explicit feedback and could benefit from automated replies provided by an interactive chatbot.

We started our exploration of dynamic interactivity through a fourth strategy involving rule-based scripts. We evaluated this strategy through a controlled experiment conducted with 97 students as part of the five-week Python programming component of a 14-week information technology course (Chapter 9). Findings from our experiment showed that while the chatbot did not lead to differences in short-term learning gains or perceived relevance of the lesson, it did improve usability.

A final strategy involved furnishing educational chatbots with the ability to generate their responses using LLMs. To equip educational chatbots with this strategy, we proposed (i) a technical architecture to integrate LLM-powered chatbots into LXPs and (ii) a configuration model to help educators define how these chatbots should interact (Chapter 10). We validated our proposals and evaluated this strategy in a controlled experiment with 26 university software engineering students. Although we did not find any significant differences between the treatment and control conditions, our results suggested that integrating an educational chatbot powered by an LLM could improve usability and help engage students to reflect on their learning.

To answer our third research question: we can equip educational chatbots with a wide variety of interaction strategies. Nevertheless, the effectiveness of these strategies depends on the pedagogical scenario in which they are embedded and on the aspects and metrics used to evaluate the impact of these strategies. Educators and researchers in education should aim to investigate interaction strategies of interest by conducting studies similar to the empirical evaluations presented in Part III of this thesis.

### 13.1.4 Design Processes

Finally, in Part IV of this thesis, we addressed design challenges through our fourth research question: *How can we support the collaborative design of educational chatbots?* Our approach to this question resulted in two contributions: (i) the TRACE model to guide the participatory design of educational chatbots (**C6**) and (ii) a technical blueprint for a system that facilitates the integration of chatbots into digital learning applications (**C7**).

The TRACE model serves to structure participatory design sessions in which different stake-

holders can collaborate on the design of educational chatbots. The model first defines five components that are essential for structuring a chatbot's integration into a learning activity and then outlines a procedure for how the co-design process should unfold, assigning specific tasks to each of the stakeholders involved. We validated this model through two pilot studies and an illustrative case study in which 25 students enrolled in a course on software design took part in a participatory design workshop to ideate an educational chatbot (Chapter 11).

The blueprint outlines an architecture specifically aimed at integrating chatbots into educational contexts, defining the key building blocks, components, and processes that make up an interaction between a learner and an educational chatbot. This blueprint also serves to translate the conceptual outcomes of design processes into tangible software solutions. To validate our blueprint, we used it to develop a proof-of-concept implementation of *Text Analysis*, an app designed to support integrating educational chatbots into a wide range of pedagogical scenarios involving text snippets (Chapter 12).

To answer our fourth research question: we can support the collaborative design of educational chatbots by providing stakeholders with conceptual models and technical blueprints that can guide participatory design processes and support the implementation of outcomes from these processes.

### 13.1.5 Conceptual Framework

To unify the findings from each of these dimensions and link them to our conceptual framework, we revisit our overarching research question: *How can a conceptual framework guide the integration of conversational agents into domain-specific learning contexts?*

As mentioned in Chapter 1, a crucial problem that emerges when attempting to integrate chatbots into digital education platforms is how to address the challenges limiting the design, development, deployment, and evaluation of these chatbots *holistically* and in a way that satisfies the needs of all stakeholders involved. We propose our conceptual framework as a guide to address this problem. Our objective is not to require the involvement of all stakeholders in every single dimension of the framework but to increase awareness and collaboration, and provide bridges between the contributions encapsulated in the different dimensions. Given this objective, our conceptual framework can be visualized as shown in Figure 13.2.

As illustrated in Figure 13.2, the first three dimensions covered in this thesis define the *outline* of our conceptual framework. Technological foundations and pedagogical scenarios define the *base*, as these two dimensions together provide the context that will host the educational chatbots. This base is primarily defined by the developers and educators designing and implementing educational technologies. At the top of our model, we place interaction strategies, which are contingent on the affordances and features provided by this base. Interaction strategies are particularly important, as they define how learners will interact with the chatbot and are therefore the most user-facing of these three dimensions. Finally, to bridge these three

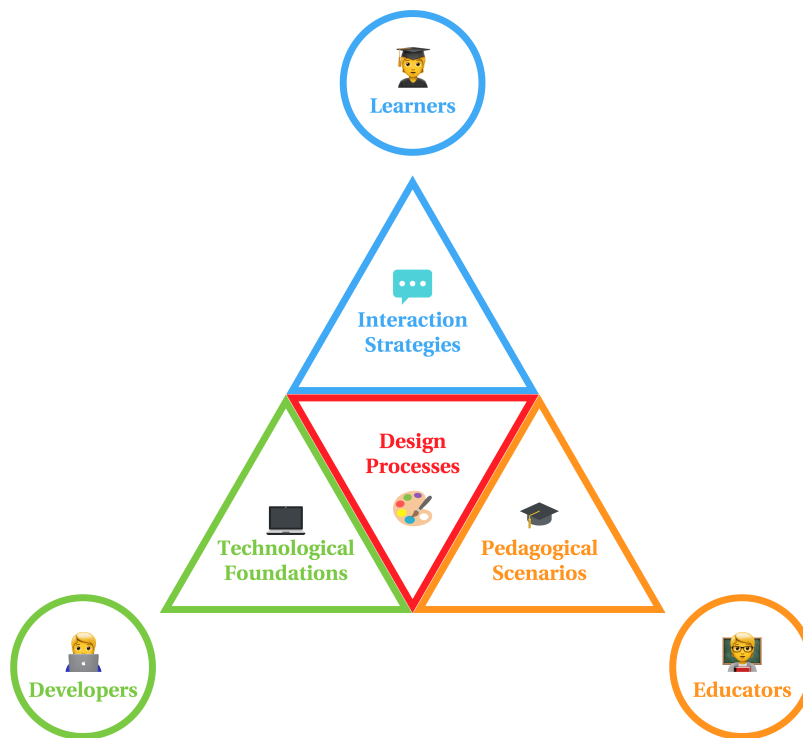


Figure 13.2: A Visual Representation of Our Conceptual Framework

dimensions, our framework emphasizes the role played by the design processes underpinning collaborative practices, which constitute a fourth and central dimension. Both conceptual models—such as TRACE—and technical architectures—such as the blueprint presented in Chapter 12—can serve to create a common vocabulary and a landscape of collaboration between the different stakeholders and across the other dimensions of our framework.

Taking the TRACE model as an example, educators looking to integrate chatbots into a particular pedagogical scenario can frame the discussions of participatory design workshops by defining learning tasks based on these pedagogical scenarios. At the other end of the model, how learners envision the exchanges that they will be having with a chatbot can inform the choice of interaction strategies with which to equip an educational chatbot. Similarly, the ability to translate the outcomes of design sessions into software solutions through technical architectures well-adapted to educational chatbots can help developers in the implementation of those outcomes.

The importance of these models and architectures is even more evident when domain-specific knowledge decisions have to be designed and implemented by different stakeholders. For instance, both educators and developers need to understand how a chatbot's function (e.g., linter, highlighter) should operate on a learning resource (e.g., code snippet, documentation) in order to ensure that outcomes from this function identify the concepts that will guide the interactions that the chatbot will support, which in turn should be related to the learning task

at hand.

To conclude, the core outcome of this thesis posits that the process of integrating educational chatbots into digital learning platforms can benefit greatly from the scaffolding provided by our conceptual framework. That is—to answer **RQ★**—our conceptual framework can support the collaborative design of the technical architectures, pedagogical scenarios, and interaction strategies required to efficiently integrate chatbots into educational contexts. This collaborative design is mediated by models and blueprints that teams of stakeholders can use to guide and translate the results of design sessions into technical solutions.

This framework is by no means final. While we proposed and validated a variety of models, architectures, strategies, and apps that give our conceptual framework its current form, other design artifacts could be generated and incorporated into the framework. Exploring how to consolidate and extend our framework will be the subject of future work.

## **13.2 Limitations and Future Work**

Throughout this thesis, we have highlighted the different limitations that might have affected our research. These limitations were mentioned at the end of each chapter. In this section, we identify general limitations that could have hindered the reliability and generalizability of our findings and chart possible future lines of research to address these limitations and build on the work conducted in this thesis.

One limitation of our work that spanned our empirical field studies concerns the number of students who participated in our experiments. While our field studies aimed to maximize ecological validity by taking place within formal education settings, the sample size of the cohorts of students might have limited our ability to identify significant differences between the treatments we tested. This was particularly the case for our experiments with Wizard of Oz and LLM-powered chatbots (30 and 26 students, respectively). Even in the case where the overall sample size was relatively large (97 students for the controlled experiment with rule-based chatbots), the fact that the interactions were optional diminished participation (only 65 students accessed all of the exercises and only 25 students completed the post-test), possibly resulting in sample bias and limiting the interpretability of our results. Although we addressed some of these shortcomings incrementally throughout the thesis, revisiting our experimental design could help consolidate our findings in future work.

A second limitation is our focus on software engineering education and specifically on the code review process. As motivated in Chapter 1, carrying out our work in the context of software engineering education allowed us to explore pedagogical scenarios that were based on how users interact with chatbots in real-world contexts (e.g., social coding platforms). Nevertheless, results obtained in software engineering settings may have limited applicability outside of this domain. While we attempted to address this limitation by conducting studies with both technical (e.g., software engineering) and nontechnical (e.g., business and economics)

students, most of our empirical evaluations were framed by the use of code review notebooks. Broadening our investigations to cover other educational domains and pedagogical scenarios could help us understand the generalizability of our framework.

Another limitation arises from our use of one LXP. Our work centered on the use of Graasp. As mentioned in Chapter 2, Graasp was identified as an appropriate educational platform on which to scaffold our technological environment and carry out our empirical studies. Nevertheless, the fact that our models, architectures, pipelines, and apps were all implemented on Graasp limits our ability to understand the generalizability of these design artifacts to other LXPs. To address this limitation, we could (i) explore the portability of the artifacts designed in this thesis to other platforms and (ii) conduct reproducibility studies that would aim to create new implementations of our models and architectures within other digital education platforms.

A factor that possibly limited our ability to achieve significant results was our choice of instrumentation. Our empirical work focused on assessing the *learning experience*, which we characterized with aspects such as learning gains, engagement, usability, self-reflection, perceived relevance, and feedback. To operationalize some of these aspects, we employed standard instruments (e.g., usability), while for others, we used custom metrics (e.g., learning gains), qualitative responses (e.g., self-reflection, feedback), individual Likert scales (e.g., perceived relevance), or criteria based on learning analytics (e.g., engagement). Improving our methodology by adopting—or developing—instruments designed specifically for the study of educational chatbots could enhance our ability to conduct analyses and draw more concrete conclusions from our quantitative and qualitative data.

Finally, while we collaborated closely with professors and teaching staff at the École Polytechnique Fédérale de Lausanne, the University of Neuchâtel, and the School of Engineering and Architecture of Fribourg, we did not explicitly investigate the needs of educators outside of these institutions. Indeed, our evaluations focused on developers and learners. Conducting evaluations specifically aimed at educators would serve to further examine the validity of our conceptual framework.

The development of our conceptual framework and the contributions comprised therein also provide a solid starting point for future explorations with conversational agents. Other promising research directions include examining the applicability of our conceptual framework to domains outside of education, such as the development of chatbots aimed at healthcare or customer service. Similarly, investigations targeting the design, development, and evaluation of other interaction strategies (e.g., speech-based) and types of agents (e.g., embodied, physical) could also be explored using our framework. Moreover, assessing the effects of interacting with these agents on factors captured through other modalities—such as physiological signals or neuroimaging data—could deepen our understanding of how educational chatbots can affect the learning experience. We aim to address the aforementioned limitations and pursue these research opportunities in future work.

## Bibliography

- Abdellatif, A., Badran, K., & Shihab, E. (2020). MSRBot: Using Bots to Answer Questions from Software Repositories. *Empirical Software Engineering*, 25(3), 1834–1863. <https://doi.org/10.1007/s10664-019-09788-5>
- Abu Shawar, B., & Atwell, E. (2007a). Chatbots: Are They Really Useful? *LDV-Forum*, 22(1), 29–49.
- Abu Shawar, B., & Atwell, E. (2007b). Fostering Language Learner Autonomy Through Adaptive Conversation Tutors. *Proceedings of the Fourth Corpus Linguistics Conference*.
- Acton, B., & Koum, J. (2009). *WhatsApp*.
- Adams Becker, S., Cummins, M., Davis, A., Freeman, A., Hall Giesinger, C., & Ananthanarayanan, V. (2017). *NMC Horizon Report: 2017 Higher Education Edition*. The New Media Consortium.
- Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., Lu, Y., & Le, Q. V. (2020, February 27). *Towards a Human-like Open-Domain Chatbot*. arXiv: 2001.09977. Retrieved January 20, 2021, from <http://arxiv.org/abs/2001.09977>
- Ahmed, T., Bosu, A., Iqbal, A., & Rahimi, S. (2017). SentiCR: A Customized Sentiment Analysis Tool for Code Review Interactions. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 106–111. <https://doi.org/10.1109/ASE.2017.8115623>
- Airbnb. (2012). Airbnb JavaScript Style Guide.
- Alario-Hoyos, C., Bote-Lorenzo, M. L., Gómez-Sánchez, E., Asensio-Pérez, J. I., Vega-Gorgojo, G., & Ruiz-Calleja, A. (2013). GLUE!: An Architecture for the Integration of External Tools in Virtual Learning Environments. *Computers & Education*, 60(1), 122–137. <https://doi.org/10.1016/j.compedu.2012.08.010>
- Alden, D. L., Hoyer, W. D., & Lee, C. (1993). Identifying Global and Culture-Specific Dimensions of Humor in Advertising: A Multinational Analysis. *Journal of Marketing*, 57(2), 64–75. <https://doi.org/10.1177/002224299305700205>
- Alves, E. (2013). *Jogos Sérios para Ensino de Engenharia de Software* (Master's thesis). Universidade do Porto. Porto, Portugal.
- Amali, L. N., Kadir, N. T., & Latief, M. (2019). Development of E-learning Content with H5P and iSpring Features. *Journal of Physics: Conference Series*, 1387(1), 012019. <https://doi.org/10.1088/1742-6596/1387/1/012019>

- Amaral, L. A., & Meurers, D. (2011). On Using Intelligent Computer-Assisted Language Learning in Real-Life Foreign Language Teaching and Learning. *ReCALL*, 23(1), 4–24. <https://doi.org/10.1017/S0958344010000261>
- Anderson, T., & Shattuck, J. (2012). Design-Based Research: A Decade of Progress in Education Research? *Educational Researcher*, 41(1), 16–25. <https://doi.org/10.3102/0013189X11428813>
- Anewalt, K. (2005). Using Peer Review as a Vehicle for Communication Skill Development and Active Learning. *Journal of Computing Sciences in Colleges*, 21(2), 148–155.
- Ardıç, B., Yurdakul, İ., & Tüzün, E. (2020). Creation of a Serious Game for Teaching Code Review: An Experience Report. *Proceedings of the 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 204–208. <https://doi.org/10.1109/CSEET49119.2020.9206173>
- Attardo, S. (2008). A Primer for the Linguistics of Humor. In V. Raskin (Ed.), *The Primer of Humor Research* (pp. 101–155). Mouton de Gruyter.
- Bacchelli, A., & Bird, C. (2013). Expectations, Outcomes, and Challenges of Modern Code Review. *2013 35th International Conference on Software Engineering (ICSE)*, 712–721. <https://doi.org/10.1109/ICSE.2013.6606617>
- Bach, S. H., Sanh, V., Yong, Z.-X., Webson, A., Raffel, C., Nayak, N. V., Sharma, A., Kim, T., Bari, M. S., Fevry, T., Alyafeai, Z., Dey, M., Santilli, A., Sun, Z., Ben-David, S., Xu, C., Chhablani, G., Wang, H., Fries, J. A., ... Rush, A. M. (2022, March 29). *PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts*. arXiv: arXiv:2202.01279. Retrieved March 9, 2023, from <http://arxiv.org/abs/2202.01279>
- Bahja, M., Hammad, R., & Butt, G. (2020). A User-Centric Framework for Educational Chatbots Design and Development. In C. Stephanidis, M. Kurosu, H. Degen, & L. Reinerman-Jones (Eds.), *HCI International 2020 - Late Breaking Papers: Multimodality and Intelligence* (pp. 32–43). Springer. [https://doi.org/10.1007/978-3-030-60117-1\\_3](https://doi.org/10.1007/978-3-030-60117-1_3)
- Bangor, A., Kortum, P., & Miller, J. (2009). Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*, 4(3), 114–123.
- Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6), 574–594. <https://doi.org/10.1080/10447310802205776>
- Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning & Leading with Technology*, 38(6), 20–23.
- Bartneck, C., Kulić, D., Croft, E., & Zoghbi, S. (2009). Measurement Instruments for the Anthropomorphism, Animacy, Likeability, Perceived Intelligence, and Perceived Safety of Robots. *International Journal of Social Robotics*, 1(1), 71–81. <https://doi.org/10.1007/s12369-008-0001-3>
- Baum, T., & Schneider, K. (2016). On the Need for a New Generation of Code Review Tools. In P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, & T. Mikkonen (Eds.), *Product-Focused Software Process Improvement* (pp. 301–308). Springer. [https://doi.org/10.1007/978-3-319-49094-6\\_19](https://doi.org/10.1007/978-3-319-49094-6_19)



- Beattie, A., Edwards, A. P., & Edwards, C. (2020). A Bot and a Smile: Interpersonal Impressions of Chatbots and Humans Using Emoji in Computer-Mediated Communication. *Communication Studies*, 71(3), 409–427. <https://doi.org/10.1080/10510974.2020.1725082>
- Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623. <https://doi.org/10.1145/3442188.3445922>
- Berg, A., Scheffel, M., Drachsler, H., Ternier, S., & Specht, M. (2016). Dutch Cooking with xAPI Recipes: The Good, the Bad, and the Consistent. *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)*, 234–236. <https://doi.org/10.1109/ICALT.2016.48>
- Bergin, J. (2000). Fourteen Pedagogical Patterns. In M. Devos & A. Rüping (Eds.), *EuroPLoP 2000*. Universitaetsverlag Konstanz.
- Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational Thinking and Tinkering: Exploration of an Early Childhood Robotics Curriculum. *Computers & Education*, 72, 145–157.
- Bibauw, S., François, T., & Desmet, P. (2019). Discussing with a Computer to Practice a Foreign Language: Research Synthesis and Conceptual Framework of Dialogue-Based Call. *Computer Assisted Language Learning*, 32(8), 827–877. <https://doi.org/10.1080/09588221.2018.1535508>
- Binkis, M., Kubiliūnas, R., Sturienė, R., Dulinskienė, T., Blažauskas, T., & Jakštienė, V. (2021). Rule-Based Chatbot Integration into Software Engineering Course. In A. Lopata, D. Gudonienė, & R. Butkienė (Eds.), *ICIST 2021: Information and Software Technologies* (pp. 367–377). Springer. [https://doi.org/10.1007/978-3-030-88304-1\\_29](https://doi.org/10.1007/978-3-030-88304-1_29)
- Bisong, E. (2019). Google Colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 59–64). Springer.
- Blumenthal, J. C. (1981). *English 3200: A Programmed Course in Grammar and Usage* (3rd). Harcourt Brace Jovanovich, Publishers.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., . . . Liang, P. (2021, August 18). *On the Opportunities and Risks of Foundation Models*. arXiv: 2108.07258 [cs]. Retrieved February 24, 2022, from <http://arxiv.org/abs/2108.07258>
- Bonwell, C. C., & Eison, J. A. (1991). *Active Learning: Creating Excitement in the Classroom*. School of Education and Human Development, George Washington University.
- Borges, H., Brito, R., & Valente, M. T. (2019). Beyond Textual Issues: Understanding the Usage and Impact of GitHub Reactions. *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 397–406. <https://doi.org/10.1145/3350768.3350788>
- Borowski, M., Zagermann, J., Klokmoose, C. N., Reiterer, H., & Rädle, R. (2020). Exploring the Benefits and Barriers of Using Computational Notebooks for Collaborative Programming Assignments. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 468–474.

## Bibliography

---

- Boyle, T., Bradley, C., Chalk, P., Jones, R., & Pickard, P. (2003). Using Blended Learning to Improve Student Success Rates in Learning to Program. *Journal of Educational Media*, 28(2-3), 165–178.
- Brandtzaeg, P. B., & Følstad, A. (2017). Why People Use Chatbots. In I. Kompatsiaris, J. Cave, A. Satsiou, G. Carle, A. Passani, E. Kontopoulos, S. Diplaris, & D. McMillan (Eds.), *Internet Science* (pp. 377–392). Springer. [https://doi.org/10.1007/978-3-319-70284-1\\_30](https://doi.org/10.1007/978-3-319-70284-1_30)
- Brooke, J. (1996). SUS: A ‘Quick and Dirty’ Usability Scale. In *Usability Evaluation In Industry*. CRC Press.
- Brown, C., & Parnin, C. (2019). Sorry to Bother You: Designing Bots for Effective Recommendations. *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, 54–58. <https://doi.org/10.1109/BotSE.2019.00021>
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models Are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (pp. 1877–1901). Curran Associates, Inc.
- Brusilovsky, P., Malmi, L., Hosseini, R., Guerra, J., Sirkiä, T., & Pollari-Malmi, K. (2018). An Integrated Practice System for Learning Programming in Python: Design and Evaluation. *Research and Practice in Technology Enhanced Learning*, 13(1), 18.
- Butterfield, S., Costello, E., Henderson, C., & Mourachov, S. (2013). *Slack*.
- Cahn, J. (2017). *CHATBOT: Architecture, Design, & Development* (Bachelor's Thesis). University of Pennsylvania. Philadelphia, PA, USA.
- Cardoso, A., Leitão, J., & Teixeira, C. (2018). Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses. *International Conference on Interactive Collaborative Learning*, 227–236.
- Chapman, B. E., & Irwin, J. (2015). Python as a First Programming Language for Biomedical Scientists. *Proceedings of the 14th Python in Science Conference*.
- Charmaz, K. (2006). *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Sage.
- Chattopadhyay, S., Prasad, I., Henley, A. Z., Sarma, A., & Barik, T. (2020). What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12.
- Chaves, A. P., & Gerosa, M. A. (2020). How Should My Chatbot Interact? A Survey on Human-Chatbot Interaction Design. *International Journal of Human-Computer Interaction*. <https://doi.org/10.1080/10447318.2020.1841438>
- Chen, Y., Jensen, S., Albert, L. J., Gupta, S., & Lee, T. (2022). Artificial Intelligence (AI) Student Assistants in the Classroom: Designing Chatbots to Support Student Success. *Information Systems Frontiers*. <https://doi.org/10.1007/s10796-022-10291-4>
- Chicaiza, J., Cabrera-Loayza, M. C., Elizalde, R., & Piedra, N. (2020). Application of Data Anonymization in Learning Analytics. *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*. <https://doi.org/10.1145/3378184.3378229>

- Chu, Q., Yu, X., Jiang, Y., & Wang, H. (2018). Data Analysis of Blended Learning in Python Programming. *International Conference on Algorithms and Architectures for Parallel Processing*, 209–217.
- Ciechanowski, L., Przegalinska, A., Magnuski, M., & Gloor, P. (2019). In the Shades of the Uncanny Valley: An Experimental Study of Human–Chatbot Interaction. *Future Generation Computer Systems*, 92, 539–548. <https://doi.org/10.1016/j.future.2018.01.055>
- Clarizia, F., Colace, F., Lombardi, M., Pascale, F., & Santaniello, D. (2018). Chatbot: An Education Support System for Student. In A. Castiglione, F. Pop, M. Ficco, & F. Palmieri (Eds.), *Cyberspace Safety and Security* (pp. 291–302). Springer. [https://doi.org/10.1007/978-3-030-01689-0\\_23](https://doi.org/10.1007/978-3-030-01689-0_23)
- Cockrill, A. (2021). From Learning Management Systems to Learning Experience Platforms: Do they keep what they promise? Reflections on a rapidly changing learning environment. <https://doi.org/10.25401/cardiffmet.14611932.v1>
- Coniam, D. (2014). The Linguistic Accuracy of Chatbots: Usability from an ESL Perspective. *Text & Talk*, 34(5), 545–567. <https://doi.org/10.1515/text-2014-0018>
- Coronado, M., Iglesias, C. A., Carrera, Á., & Mardomingo, A. (2018). A Cognitive Assistant for Learning Java Featuring Social Dialogue. *International Journal of Human-Computer Studies*, 117, 55–67. <https://doi.org/10.1016/j.ijhcs.2018.02.004>
- Crow, T., Luxton-Reilly, A., & Wuensche, B. (2018). Intelligent Tutoring Systems for Programming Education: A Systematic Review. *ACE 2018: 20th Australasian Computing Education Conference*, 53–62. <https://doi.org/10.1145/3160489.3160492>
- Cui, L., Huang, S., Wei, F., Tan, C., Duan, C., & Zhou, M. (2017). SuperAgent: A Customer Service Chatbot for E-Commerce Websites. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics-System Demonstrations*, 97–102. <https://doi.org/10.18653/v1/P17-4017>
- Cumbo, B., & Selwyn, N. (2022). Using Participatory Design Approaches in Educational Research. *International Journal of Research & Method in Education*, 45(1), 60–72. <https://doi.org/10.1080/1743727X.2021.1902981>
- Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*, 1277–1286. <https://doi.org/10.1145/2145204.2145396>
- Dahlbäck, N., Jönsson, A., & Ahrenberg, L. (1993). Wizard of Oz Studies—Why and How. *Knowledge-Based Systems*, 6(4), 258–266.
- Davids, M. R., Chikte, U. M. E., & Halperin, M. L. (2014). Effect of Improving the Usability of an E-Learning Resource: A Randomized Trial. *Advances in Physiology Education*, 38(2), 155–160. <https://doi.org/10.1152/advan.00119.2013>
- De Angeli, A., Johnson, G. I., & Coventry, L. (2001). The Unfriendly User: Exploring Social Reactions to Chatterbots. In M. G. Helander, H. M. Khalid, & M. P. Tham (Eds.), *Proceedings of the International Conference on Affective Human Factors Design* (pp. 467–474). ASEAN Academic Press.

## Bibliography

---

- de Jong, T., Sotiriou, S., & Gillet, D. (2014). Innovations in STEM Education: The Go-Lab Federation of Online Labs. *Smart Learning Environments*, 1(1), 3. <https://doi.org/10.1186/s40561-014-0003-6>
- del Blanco, A., Serrano, A., Freire, M., Martinez-Ortiz, I., & Fernandez-Manjon, B. (2013). E-Learning Standards and Learning Analytics: Can Data Collection Be Improved by Using Standard Data Models? *2013 IEEE Global Engineering Education Conference (EDUCON)*, 1255–1261. <https://doi.org/10.1109/EduCon.2013.6530268>
- Demir, L., Kumar, A., Cunche, M., & Lauradoux, C. (2018). The Pitfalls of Hashing for Privacy. *IEEE Communications Surveys Tutorials*, 20(1), 551–565. <https://doi.org/10.1109/COMST.2017.2747598>
- Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., & Mockus, A. (2020). Detecting and Characterizing Bots that Commit Code. *Proceedings of the 17th International Conference on Mining Software Repositories*, 209–219. <https://doi.org/10.1145/3379597.3387478>
- Dietze, S., Siemens, G., Taibi, D., & Drachsler, H. (2016). Editorial: Datasets for Learning Analytics. *Journal of Learning Analytics*, 3(2), 307–311. <https://doi.org/10.18608/jla.2016.32.15>
- Dillman, K. R., Mok, T. T. H., Tang, A., Oehlberg, L., & Mitchell, A. (2018). A Visual Interaction Cue Framework from Video Game Environments for Augmented Reality. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. <https://doi.org/10.1145/3173574.3173714>
- Drachsler, H., & Greller, W. (2016). Privacy and Analytics: It's a DELICATE Issue a Checklist for Trusted Learning Analytics. *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge - LAK '16*, 89–98. <https://doi.org/10.1145/2883851.2883893>
- Drever, E. (1997). *Using Semi-Structured Interviews in Small-Scale Research: A Teacher's Guide* (Reprint). Scottish Council for Research in Education.
- Duffy, B. R. (2003). Anthropomorphism and the Social Robot. *Robotics and Autonomous Systems*, 42(3-4), 177–190. [https://doi.org/10.1016/S0921-8890\(02\)00374-3](https://doi.org/10.1016/S0921-8890(02)00374-3)
- Duijst, D. (2017). Can We Improve the User Experience of Chatbots with Personalisation? <https://doi.org/10.13140/RG.2.2.36112.92165>
- Durall, E., & Kapros, E. (2020). Co-Design for a Competency Self-Assessment Chatbot and Survey in Science Education. In P. Zaphiris & A. Ioannou (Eds.), *Learning and Collaboration Technologies. Human and Technology Ecosystems* (pp. 13–24). Springer. [https://doi.org/10.1007/978-3-030-50506-6\\_2](https://doi.org/10.1007/978-3-030-50506-6_2)
- Durall Gazulla, E., Martins, L., & Fernández-Ferrer, M. (2023). Designing Learning Technology Collaboratively: Analysis of a Chatbot Co-Design. *Education and Information Technologies*, 28(1), 109–134. <https://doi.org/10.1007/s10639-022-11162-w>
- Dybala, P., Ptaszynski, M., Rzepka, R., & Araki, K. (2009a). Activating Humans with Humor—A Dialogue System That Users Want to Interact With. *IEICE Transactions on Information and Systems*, E92-D(12), 2394–2401. <https://doi.org/10.1587/transinf.E92.D.2394>

- Dybala, P., Ptaszynski, M., Rzepka, R., & Araki, K. (2009b). Humoroids: Conversational Agents that Induce Positive Emotions with Humor. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2, 1171–1172.
- Edlund, J. R., & Griswold, O. (2012). Non-Native Speakers of English. In I. L. Clark (Ed.), *Concepts in Composition: Theory and Practice of the Teaching of Writing* (2nd, pp. 317–338). Routledge.
- Edwards, J., Clark, L., & Perrone, A. (2021). LGBTQ-AI? Exploring Expressions of Gender and Sexual Orientation in Chatbots. *CUI 2021 - 3rd Conference on Conversational User Interfaces*, 1–4. <https://doi.org/10.1145/3469595.3469597>
- Edwards, S. H., Tilden, D. S., & Allevato, A. (2014). Python: Improving the Introductory Python Programming Experience. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 641–646.
- European Union. (2016). Regulation 2016/679 of the European Parliament and the Council of the European Union. *Official Journal of the European Communities*, 2014(April), 1–88.
- Facebook. (2008). *Messenger*.
- Farah, J. C., Ingram, S., & Gillet, D. (2022). Supporting Developers in Creating Web Apps for Education via an App Development Framework. *HEAd'22 Conference Proceedings*, 893–890. <https://doi.org/10.4995/HEAD22.2022.15652>
- Farah, J. C., Ingram, S., Spaenlehauer, B., Lasne, F. K.-L., & Gillet, D. (2023). Prompting Large Language Models to Power Educational Chatbots [In Submission].
- Farah, J. C., Moro, A., Bergram, K., Purohit, A. K., Gillet, D., & Holzer, A. (2020). Bringing Computational Thinking to non-STEM Undergraduates through an Integrated Notebook Application. In T. Broos & T. Farrell (Eds.), *Proceedings of the Impact Papers at the 15th European Conference on Technology-Enhanced Learning (EC-TEL 2020)*.
- Farah, J. C., Sharma, V., Ingram, S., & Gillet, D. (2021). Conveying the Perception of Humor Arising from Ambiguous Grammatical Constructs in Human-Chatbot Interaction. *Proceedings of the 9th International Conference on Human-Agent Interaction (HAI '21)*, 257–262. <https://doi.org/10.1145/3472307.3484677>
- Farah, J. C., Soares Machado, J., Torres da Cunha, P., Ingram, S., & Gillet, D. (2021). An End-to-End Data Pipeline for Managing Learning Analytics. *2021 19th International Conference on Information Technology Based Higher Education and Training (ITHET)*. <https://doi.org/10.1109/ITHET50392.2021.9759783>
- Farah, J. C., Spaenlehauer, B., Bergram, K., Holzer, A., & Gillet, D. (2022). Challenges and Opportunities in Integrating Interactive Chatbots into Code Review Exercises: A Pilot Case Study. *EDULEARN22 Proceedings*, 3816–3825. <https://doi.org/10.21125/edulearn.2022.0932>
- Farah, J. C., Spaenlehauer, B., Ingram, S., & Gillet, D. (2022). A Blueprint for Integrating Task-Oriented Conversational Agents in Education. *4th Conference on Conversational User Interfaces (CUI 2022)*. <https://doi.org/10.1145/3543829.3544525>
- Farah, J. C., Spaenlehauer, B., Ingram, S., Lasne, F. K.-L., Rodríguez-Triana, M. J., Holzer, A., & Gillet, D. (2023). TRACE: A Conceptual Model to Guide the Design of Educational Chatbots [In Submission].

## Bibliography

---

- Farah, J. C., Spaenlehauer, B., Ingram, S., Purohit, A. K., Holzer, A., & Gillet, D. (2023). Harnessing Rule-Based Chatbots to Support Teaching Python Programming Best Practices [In Submission].
- Farah, J. C., Spaenlehauer, B., Lu, X., Ingram, S., & Gillet, D. (2022). An Exploratory Study of Reactions to Bot Comments on GitHub. *2022 IEEE/ACM 4th International Workshop on Bots in Software Engineering (BotSE)*. <https://doi.org/10.1145/3528228.3528409>
- Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Toward Code Review Notebooks. *2022 International Conference on Advanced Learning Technologies (ICALT)*, 209–211. <https://doi.org/10.1109/ICALT55010.2022.00068>
- Farah, J. C., Spaenlehauer, B., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2023). Integrating Code Reviews into Online Lessons to Support Software Engineering Education. In M. E. Auer, W. Pachatz, & T. Rüttmann (Eds.), *Learning in the Age of Digital and Green Transition* (pp. 815–826). Springer. [https://doi.org/10.1007/978-3-031-26190-9\\_84](https://doi.org/10.1007/978-3-031-26190-9_84)
- Farah, J. C., Spaenlehauer, B., Sharma, V., Rodríguez-Triana, M. J., Ingram, S., & Gillet, D. (2022). Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices. *2022 IEEE Global Engineering Education Conference (EDUCON)*, 1634–1642. <https://doi.org/10.1109/EDUCON52537.2022.9766793>
- Feine, J., Gnewuch, U., Morana, S., & Maedche, A. (2019). A Taxonomy of Social Cues for Conversational Agents. *International Journal of Human-Computer Studies*, 132, 138–161. <https://doi.org/10.1016/j.ijhcs.2019.07.009>
- Feine, J., Morana, S., & Maedche, A. (2019). Designing a Chatbot Social Cue Configuration System. *Proceedings of the 40th International Conference on Information Systems (ICIS)*.
- Feine, J., Morana, S., & Maedche, A. (2020). Designing Interactive Chatbot Development Systems. *Proceedings of the 41st International Conference on Information Systems (ICIS)*.
- Feliciano, J., Storey, M.-A., & Zagalsky, A. (2016). Student Experiences Using GitHub in Software Engineering Courses: A Case Study. *Proceedings of the 38th International Conference on Software Engineering Companion*, 422–431. <https://doi.org/10.1145/2889160.2889195>
- Felten, E., Raj, M., & Seamans, R. (2023, March 18). *How will Language Modelers like ChatGPT Affect Occupations and Industries?* arXiv: 2303.01157 [econ, q-fin]. Retrieved May 19, 2023, from <http://arxiv.org/abs/2303.01157>
- Ferguson, R. (2012). Learning Analytics: Drivers, Developments and Challenges. *International Journal of Technology Enhanced Learning*, 4(5/6), 304. <https://doi.org/10.1504/IJTEL.2012.051816>
- Ferguson, R., Hoel, T., Scheffel, M., & Drachsler, H. (2016). Guest Editorial: Ethics and Privacy in Learning Analytics. *Journal of Learning Analytics*, 3(1). <https://doi.org/10.18608/jla.2016.31.2>
- Ferman Guerra, M. A. (2018). *Towards Best Practices for Chatbots* (Master's thesis). University of Victoria. Victoria, BC, Canada.

- Fiksel, J., Jager, L. R., Hardin, J. S., & Taub, M. A. (2019). Using GitHub Classroom To Teach Statistics. *Journal of Statistics Education*, 27(2), 110–119. <https://doi.org/10.1080/10691898.2019.1617089>
- Fogg, B. J., & Nass, C. (1997). Silicon Sycophants: The Effects of Computers That Flatter. *International Journal of Human-Computer Studies*, 46(5), 551–561.
- Følstad, A., Araujo, T., Law, E. L.-C., Brandtzaeg, P. B., Papadopoulos, S., Reis, L., Baez, M., Laban, G., McAllister, P., Ischen, C., Wald, R., Catania, F., Meyer von Wolff, R., Hobert, S., & Luger, E. (2021). Future Directions for Chatbot Research: An Interdisciplinary Research Agenda. *Computing*, 103(12), 2915–2942. <https://doi.org/10.1007/s00607-021-01016-7>
- Følstad, A., & Brandtzaeg, P. B. (2020). Users' Experiences with Chatbots: Findings from a Questionnaire Study. *Quality and User Experience*, 5(1). <https://doi.org/10.1007/s41233-020-00033-2>
- Fordyce-Ruff, T. (2011). Laughing All the Way to Court: Avoiding the Humor and Headaches Created by Misplaced Modifiers. *The Advocate*, 54(11/12), 37–38.
- Fryer, L. K., Nakao, K., & Thompson, A. (2019). Chatbot Learning Partners: Connecting Learning Experiences, Interest and Competence. *Computers in Human Behavior*, 93, 279–289. <https://doi.org/10.1016/j.chb.2018.12.023>
- Gabrielli, S., Rizzi, S., Carbone, S., & Donisi, V. (2020). A Chatbot-Based Coaching Intervention for Adolescents to Promote Life Skills: Pilot Study. *JMIR Human Factors*, 7(1), e16762. <https://doi.org/10.2196/16762>
- Gatsby. (2022, February). *Gatsby* (Version 4.7).
- Gillet, D., de Jong, T., Sotirou, S., & Salzmann, C. (2013). Personalised Learning Spaces and Federated Online Labs for STEM Education at School. *2013 IEEE Global Engineering Education Conference (EDUCON)*, 769–773. <https://doi.org/10.1109/EduCon.2013.6530194>
- Gillet, D., Vonèche-Cardia, I., Farah, J. C., Phan Hoang, K. L., & Rodríguez-Triana, M. J. (2022). Integrated Model for Comprehensive Digital Education Platforms. *2022 IEEE Global Engineering Education Conference (EDUCON)*, 1586–1592. <https://doi.org/10.1109/EDUCON52537.2022.9766795>
- Glazunova, O. G., Parhomenko, O. V., Korolchuk, V. I., & Voloshyna, T. V. (2021). The Effectiveness of GitHub Cloud Services for Implementing a Programming Training Project: Students' Point of View. *Journal of Physics: Conference Series*, 1840(012030). <https://doi.org/10.1088/1742-6596/1840/1/012030>
- Go, E., & Sundar, S. S. (2019). Humanizing Chatbots: The Effects of Visual, Identity and Conversational Cues on Humanness Perceptions. *Computers in Human Behavior*, 97, 304–316. <https://doi.org/10.1016/j.chb.2019.01.020>
- Goldberg, L. R. (1993). The Structure of Phenotypic Personality Traits. *American Psychologist*, 48(1), 26–34.
- Golzadeh, M., Legay, D., Decan, A., & Mens, T. (2020). Bot or Not? Detecting Bots in GitHub Pull Request Activity Based on Comment Similarity. *Proceedings of the IEEE/ACM 42nd*

## Bibliography

---

- International Conference on Software Engineering Workshops*, 31–35. <https://doi.org/10.1145/3387940.3391503>
- Gomes, N. R. C., Farah, J. C., Doran, R., & Gillet, D. (2019). A Light Pollution Simulator. *EPSC Abstracts*, 13.
- Google. (2012). *DialogFlow*.
- Graesser, A., Chipman, P., Haynes, B., & Olney, A. (2005). AutoTutor: An Intelligent Tutoring System With Mixed-Initiative Dialogue. *IEEE Transactions on Education*, 48(4), 612–618. <https://doi.org/10.1109/TE.2005.856149>
- Graham, E. E., Papa, M. J., & Brooks, G. P. (1992). Functions of Humor in Conversation: Conceptualization and Measurement. *Western Journal of Communication (Includes Communication Reports)*, 56(2), 161–183.
- Grandell, L., Peltomäki, M., Back, R.-J., & Salakoski, T. (2006). Why Complicate Things? Introducing Programming in High School Using Python. *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, 71–80.
- Greller, W., & Drachsler, H. (2012). Translating Learning into Numbers: A Generic Framework for Learning Analytics.
- Grigorik, I. (2012). *GH Archive*.
- Griol, D., & Callejas, Z. (2013). An Architecture to Develop Multimodal Educative Applications with Chatbots. *International Journal of Advanced Robotic Systems*, 10(3). <https://doi.org/10.5772/55791>
- Gruber, J., & Swartz, A. (2004). Markdown.
- Guo, P. J. (2013). Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 579–584.
- Gursoy, M. E., Inan, A., Nergiz, M. E., & Saygin, Y. (2017). Privacy-Preserving Learning Analytics: Challenges and Techniques. *IEEE Transactions on Learning Technologies*, 10(1), 68–81. <https://doi.org/10.1109/TLT.2016.2607747>
- Haaranen, L., & Lehtinen, T. (2015). Teaching Git on the Side: Version Control System as a Course Platform. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 87–92. <https://doi.org/10.1145/2729094.2742608>
- Hansen, K. (1983). The Anals of History: Unintentional Humor from Freshman Compositions. *The English Journal*, 72(7), 44. <https://doi.org/10.2307/816303>
- Haristiani, N. (2019). Artificial Intelligence (AI) Chatbot as Language Learning Medium: An Inquiry. *Journal of Physics: Conference Series*, 1387(1), 012020. <https://doi.org/10.1088/1742-6596/1387/1/012020>
- Häsel, M. (2011). OpenSocial: An Enabler for Social Applications on the Web. *Communications of the ACM*, 54(1), 139–144. <https://doi.org/10.1145/1866739.1866765>
- Hassan, F., Sánchez, D., Soria-Comas, J., & Domingo-Ferrer, J. (2019). Automatic Anonymization of Textual Documents: Detecting Sensitive Information via Word Embeddings. *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And*



- Engineering (TrustCom/BigDataSE)*, 358–365. <https://doi.org/10.1109/TrustCom/BigDataSE.2019.00055>
- Hayashi, Y. (2015). Social Facilitation Effects by Pedagogical Conversational Agent: Lexical Network Analysis in an Online Explanation Task. *Proceedings of the 8th International Conference on Educational Data Mining*, 484–487.
- Hedberg, H. (2004). Introducing the Next Generation of Software Inspection Tools. In T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, M. Y. Vardi, F. Bomarius, & H. Iida (Eds.), *Product Focused Software Process Improvement* (pp. 234–247). Springer. [https://doi.org/10.1007/978-3-540-24659-6\\_17](https://doi.org/10.1007/978-3-540-24659-6_17)
- Hellman, S. V. (2007). Humor in the Classroom: STU'S Seven Simple Steps to Success. *College Teaching*, 55(1), 37–39. <https://doi.org/10.3200/CTCH.55.1.37-39>
- Heurix, J., Zimmermann, P., Neubauer, T., & Fenz, S. (2015). A Taxonomy for Privacy Enhancing Technologies. *Computers & Security*, 53. <https://doi.org/10.1016/j.cose.2015.05.002>
- Higashinaka, R., Meguro, T., Sugiyama, H., Makino, T., & Matsuo, Y. (2015). On the Difficulty of Improving Hand-Crafted Rules in Chat-Oriented Dialogue Systems. *Proceedings of APSIPA Annual Summit and Conference 2015*, 1014–1018.
- Hobert, S. (2020). Say Hello to 'Coding Tutor'! Design and Evaluation of a Chatbot-Based Learning System Supporting Students to Learn to Program. *40th International Conference on Information Systems (ICIS 2019)*, 3, 1776–1792.
- Hoel, T., & Chen, W. (2014). Learning Analytics Interoperability - Looking for Low-Hanging Fruits. *Proceedings of the 22nd International Conference on Computers in Education*.
- Hohenwarter, M. (2002). *GeoGebra - ein Softwaresystem für dynamische Geometrie und Algebra der Ebene*. Universität Salzburg.
- Hsing, C., & Gennarelli, V. (2019). Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 672–678. <https://doi.org/10.1145/3287324.3287460>
- Hu, X., Moore, A., Coleman Eubanks, J., Aiyaz, A., & P. McMahan, R. (2020). Evaluating Interaction Cue Purpose and Timing for Learning and Retaining Virtual Reality Training. *Symposium on Spatial User Interaction*, 1–9. <https://doi.org/10.1145/3385959.3418448>
- Huang, W., Hew, K. F., & Fryer, L. K. (2022). Chatbots for Language Learning—Are They Really Useful? A Systematic Review of Chatbot-Supported Language Learning. *Journal of Computer Assisted Learning*, 38(1), 237–257. <https://doi.org/10.1111/jcal.12610>
- Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95.
- Hutto, C. J., & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*, 216–225.
- Hwang, G.-J., & Chang, C.-Y. (2021). A Review of Opportunities and Challenges of Chatbots in Education. *Interactive Learning Environments*. <https://doi.org/10.1080/10494820.2021.1952615>

## Bibliography

---

- IMS Global Learning Consortium. (2019). *Learning Tools Interoperability Core Specification (Version 1.3)* (tech. rep.). IMS Global Learning Consortium, Inc.
- Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2020). A Review of Peer Code Review in Higher Education. *ACM Transactions on Computing Education*, 20(3). <https://doi.org/10.1145/3403935>
- Ismail, M., & Ade-Ibijola, A. (2019). Lecturer's Apprentice: A Chatbot for Assisting Novice Programmers. *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. <https://doi.org/10.1109/IMITEC45504.2019.9015857>
- Jain, M., Kumar, P., Kota, R., & Patel, S. N. (2018). Evaluating and Informing the Design of Chatbots. *Proceedings of the 2018 Designing Interactive Systems Conference*, 895–906. <https://doi.org/10.1145/3196709.3196735>
- Jia, J. (2004). CSIEC (Computer Simulator in Educational Communication): A Virtual Context-Adaptive Chatting Partner for Foreign Language Learners. *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, 690–692.
- Jiang, W., & Clifton, C. (2005). Privacy-Preserving Distributed k-Anonymity. In S. Jajodia & D. Wijesekera (Eds.), *Data and Applications Security XIX* (pp. 166–177). Springer.
- Jiang, Z., Xu, F. F., Araki, J., & Neubig, G. (2020, May 3). *How Can We Know What Language Models Know?* arXiv: arXiv:1911.12543. Retrieved October 17, 2022, from <http://arxiv.org/abs/1911.12543>
- Johansson, M. (2021). Talking with a Chatbot: Simulated Understanding of Human–Chatbot Communication? In M. Johansson, S.-K. Tanskanen, & J. Chovanec (Eds.), *Analyzing Digital Discourses* (pp. 105–131). Springer. [https://doi.org/10.1007/978-3-030-84602-2\\_5](https://doi.org/10.1007/978-3-030-84602-2_5)
- Johnson, S. C. (1978). *Lint, A C Program Checker* (tech. rep.). Bell Laboratories. Murray Hill, NJ, USA.
- Joint Task Force on Computing Curricula. (2015). *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* (tech. rep.). IEEE & ACM.
- Joubert, P. H., & Rogers, S. M. (2015). Language Pitfalls: Native English Speakers. In *Strategic Scientific and Medical Writing* (pp. 25–37). Springer. [https://doi.org/10.1007/978-3-662-48316-9\\_4](https://doi.org/10.1007/978-3-662-48316-9_4)
- Jung, H., Lee, J., & Park, C. (2020). Deriving Design Principles for Educational Chatbots from Empirical Studies on Human–Chatbot Interaction. *Journal of Digital Contents Society*, 21(3), 487–493. <https://doi.org/10.9728/dcs.2020.21.3.487>
- Kanuka, H., & Szabo, M. (1999). Conducting Research on Visual Design and Learning: Pitfalls and Promises. *Canadian Journal of Learning and Technology / La revue canadienne de l'apprentissage et de la technologie*, 27(2). <https://doi.org/10.21432/T2SW37>
- Kaplow, R., Desch, S. H., Pettijohn, D. O., Rodman, M. H., & Smith, F. C. (1973). Illustrations of Conversational, Inquiry, Problem-Solving, and Questionnaire Type Interactions within the TICS System. *Proceedings of the 7th Annual Princeton Conference on Information Sciences and Systems*, 389–393.

- Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., ... Kasneci, G. (2023, January 30). *ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education*. <https://doi.org/10.35542/osf.io/5er8f>
- Kellogg, S., & Edelman, A. (2015). Massively Open Online Course for Educators (MOOC-Ed) Network Dataset: MOOC-Ed Network Dataset. *British Journal of Educational Technology*, 46(5), 977–983. <https://doi.org/10.1111/bjet.12312>
- Kensing, F., & Munk-Madsen, A. (1993). PD: Structure in the Toolbox. *Communications of the ACM*, 36(6), 78–85. <https://doi.org/10.1145/153571.163278>
- Kerly, A., Hall, P., & Bull, S. (2007). Bringing Chatbots into Education: Towards Natural Language Negotiation of Open Learner Models. *Knowledge-Based Systems*, 20(2), 177–185. <https://doi.org/10.1016/j.knosys.2006.11.014>
- Kim, A. S., & Ko, A. J. (2017). A Pedagogical Analysis of Online Coding Tutorials. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 321–326.
- Komarova, T., Nekipelov, D., Al Rafi, A., & Yakovlev, E. (2017). k-Anonymity: A Note on the Trade-Off between Data Utility and Data Security. *Applied Econometrics*, 48, 44–62.
- Kubincová, Z., & Csicsolová, I. (2018). Code Review in High School Programming. *Proceedings of the 2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*. <https://doi.org/10.1109/ITHET.2018.8424617>
- Kubincová, Z., & Homola, M. (2017). Code Review in Computer Science Courses: Take One. In H. Xie, E. Popescu, G. Hancke, & B. Fernández Manjón (Eds.), *Advances in Web-Based Learning – ICWL 2017* (pp. 125–135). Springer. [https://doi.org/10.1007/978-3-319-66733-1\\_14](https://doi.org/10.1007/978-3-319-66733-1_14)
- Kuhail, M. A., Alturki, N., Alramlawi, S., & Alhejori, K. (2023). Interacting with Educational Chatbots: A Systematic Review. *Education and Information Technologies*, 28(1), 973–1018. <https://doi.org/10.1007/s10639-022-11177-3>
- Kumar, J. A. (2021). Educational Chatbots for Project-Based Learning: Investigating Learning Outcomes for a Team-Based Design Course. *International Journal of Educational Technology in Higher Education*, 18(1). <https://doi.org/10.1186/s41239-021-00302-w>
- La Scala, J., Aad, G., Vonèche-Cardia, I., & Gillet, D. (2022). Developing Transversal Skills and Strengthening Collaborative Blended Learning Activities in Engineering Education: A Pilot Study. *2022 20th International Conference on Information Technology Based Higher Education and Training (ITHET)*, 1–8. <https://doi.org/10.1109/ITHET56107.2022.10031948>
- Laiq, M., & Dieste, O. (2020). Chatbot-Based Interview Simulator: A Feasible Approach to Train Novice Requirements Engineers. *2020 10th International Workshop on Requirements Engineering Education and Training (REET)*, 1–8. <https://doi.org/10.1109/REET51203.2020.00007>
- Laranjo, L., Dunn, A. G., Tong, H. L., Kocaballi, A. B., Chen, J., Bashir, R., Surian, D., Gallego, B., Magrabi, F., Lau, A. Y. S., & Coiera, E. (2018). Conversational Agents in Healthcare: A

- Systematic Review. *Journal of the American Medical Informatics Association*, 25(9), 1248–1258. <https://doi.org/10.1093/jamia/ocy072>
- Laugwitz, B., Held, T., & Schrepp, M. (2008). Construction and Evaluation of a User Experience Questionnaire. In A. Holzinger (Ed.), *HCI and Usability for Education and Work* (pp. 63–76). Springer. [https://doi.org/10.1007/978-3-540-89350-9\\_6](https://doi.org/10.1007/978-3-540-89350-9_6)
- Lebeuf, C., Storey, M.-A., & Zagalsky, A. (2017). How Software Developers Mitigate Collaboration Friction with Chatbots. *Proceedings of the Talking with Conversational Agents in Collaborative Action Workshop at the 20th ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW '17)*.
- Lebeuf, C., Storey, M.-A., & Zagalsky, A. (2018). Software Bots. *IEEE Software*, 35(1), 18–23. <https://doi.org/10.1109/MS.2017.4541027>
- Lederer, R. (1989). *Anguished English: An Anthology of Accidental Assaults Upon the English Language*. Dell.
- Lester, M., & Beason, L. (2018). *The McGraw-Hill Education Handbook of English Grammar and Usage* (3rd). McGraw-Hill Education.
- Li, N., Li, T., & Venkatasubramanian, S. (2007). T-Closeness: Privacy Beyond k-Anonymity and l-Diversity. *2007 IEEE 23rd International Conference on Data Engineering*, 106–115. <https://doi.org/10.1109/ICDE.2007.367856>
- Li, X., & Prasad, C. (2005). Effectively Teaching Coding Standards in Programming. *Proceedings of the 6th Conference on Information Technology Education (SIGITE '05)*, 239–244. <https://doi.org/10.1145/1095714.1095770>
- Liaw, S.-S., & Huang, H.-M. (2013). Perceived Satisfaction, Perceived Usefulness and Interactive Learning Environments as Predictors to Self-Regulation in e-Learning Environments. *Computers & Education*, 60(1), 14–24. <https://doi.org/10.1016/j.compedu.2012.07.015>
- Liu, D., Smith, M. J., & Veeramachaneni, K. (2020). Understanding User-Bot Interactions for Small-Scale Automation in Open-Source Development. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3334480.3382998>
- Loehr, D. (1996). An Integration of a Pun Generator with a Natural Language Robot. In J. H. Hulstijn & A. Nijholt (Eds.), *Proceedings of the International Workshop on Computational Humor* (pp. 161–172). University of Twente.
- Luger, E., & Sellen, A. (2016). “Like Having a Really Bad PA”: The Gulf Between User Expectation and Experience of Conversational Agents. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 5286–5297. <https://doi.org/10.1145/2858036.2858288>
- Luo, C. J., & Gonda, D. E. (2019). Code Free Bot: An Easy Way to Jumpstart Your Chatbot! *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*. <https://doi.org/10.1109/TALE48000.2019.9226016>
- Lye, S. Y., & Koh, J. H. L. (2014). Review on Teaching and Learning of Computational Thinking through Programming: What Is next for K-12? *Computers in Human Behavior*, 41, 51–61.

- Machado, J. S., Farah, J. C., Gillet, D., & Rodríguez-Triana, M. J. (2019). Towards Open Data in Digital Education Platforms. *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, 2161-377X, 209–211. <https://doi.org/10.1109/ICALT.2019.00048>
- Machanavajjhala, A., Gehrke, J., Kifer, D., & Venkitasubramaniam, M. (2006). l-Diversity: Privacy Beyond k-Anonymity. *22nd International Conference on Data Engineering (ICDE'06)*. <https://doi.org/10.1109/ICDE.2006.1>
- Mad Daud, S. H., Ibrahim Teo, N. H., & Mat Zain, N. H. (2020). E-JAVA Chatbot for Learning Programming Language: A Post-Pandemic Alternative Virtual Tutor. *International Journal of Emerging Trends in Engineering Research*, 8(7), 3290–3298. <https://doi.org/10.30534/ijeter/2020/67872020>
- Majumdar, R., Akćapınar, A., Akćapınar, G., Flanagan, B., & Ogata, H. (2019). LAViEW: Learning Analytics Dashboard Towards Evidence-based Education. *Companion Proceedings 9th International Conference on Learning Analytics & Knowledge (LAK19)*.
- Mangaroska, K., & Giannakos, M. (2019). Learning Analytics for Learning Design: A Systematic Literature Review of Analytics-Driven Design to Enhance Learning. *IEEE Transactions on Learning Technologies*, 12(4), 516–534. <https://doi.org/10.1109/TLT.2018.2868673>
- Marques, M., & Robledo, J. (2018). What Software Engineering “Best Practices” Are We Teaching Students - A Systematic Literature Review. *2018 IEEE Frontiers in Education Conference (FIE)*. <https://doi.org/10.1109/FIE.2018.8658576>
- McDonnell, M., & Baxter, D. (2019). Chatbots and Gender Stereotyping. *Interacting with Computers*, 31(2), 116–121. <https://doi.org/10.1093/iwc/iwz007>
- McLean, S. (2013). *Writing for Success*. Flat World Knowledge, Inc.
- McTear, M. (2021). *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Springer. <https://doi.org/10.1007/978-3-031-02176-3>
- Medhi Thies, I., Menon, N., Magapu, S., Subramony, M., & O'Neill, J. (2017). How Do You Want Your Chatbot? An Exploratory Wizard-of-Oz Study with Young, Urban Indians. In R. Bernhaupt, G. Dalvi, A. Joshi, D. K. Balkrishan, J. O'Neill, & M. Winckler (Eds.), *Human-computer interaction - interact 2017* (pp. 441–459). Springer.
- Minocha, S., & Thomas, P. G. (2007). Collaborative Learning in a Wiki Environment: Experiences from a Software Engineering Course. *New Review of Hypermedia and Multimedia*, 13(2), 187–209. <https://doi.org/10.1080/13614560701712667>
- Moon, Y. (2000). Intimate Exchanges: Using Computers to Elicit Self-Disclosure from Consumers. *Journal of Consumer Research*, 26(4), 323–339. <https://doi.org/10.1086/209566>
- Moore, J., & Churchward, M. (2010). *Moodle 1.9 Multimedia Extension Development: Customize and Extend Moodle by Using its Robust Plugin Systems*. Packt Publishing Ltd.
- Morgado da Costa, L., Bond, F., & He, X. (2016). Syntactic Well-Formedness Diagnosis and Error-Based Coaching in Computer Assisted Language Learning Using Machine Translation. *Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications*, 107–116.
- Mori, M. (2012). The Uncanny Valley (K. MacDorman & N. Kageki, Trans.). *IEEE Robotics & Automation Magazine*, 19(2), 98–100. <https://doi.org/10.1109/MRA.2012.2192811>

## Bibliography

---

- Morkes, J., Kernal, H. K., & Nass, C. (1999). Effects of Humor in Task-Oriented Human-Computer Interaction and Computer-Mediated Communication: A Direct Test of SRCT Theory. *Human-Computer Interaction*, 14(4), 395–435. [https://doi.org/10.1207/S15327051HCI1404\\_2](https://doi.org/10.1207/S15327051HCI1404_2)
- Munroe, R. (2015, April 17). *Code Quality*. xkcd. <https://xkcd.com/1513/>
- Nass, C., Moon, Y., & Green, N. (1997). Are Machines Gender Neutral? Gender-Stereotypic Responses to Computers with Voices. *Journal of Applied Social Psychology*, 27(10), 864–876.
- Nass, C., Steuer, J., & Tauber, E. R. (1994). Computers Are Social Actors. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Celebrating Interdependence - CHI '94*, 72–78. <https://doi.org/10.1145/191666.191703>
- Nazario, L. A., Borchers, D. D., & Lewis, W. F. (2010). *Bridges to Better Writing*. Wandsworth.
- Nelson, G. E., Ward, J. R., Desch, S. H., & Kaplow, R. (1976). Two New Strategies for Computer-Assisted Language Instruction (CALI). *Foreign Language Annals*, 9(1), 28–37.
- Nicholson, J., & Tasker, I. (2017). DataExchange: Privacy by Design for Data Sharing in Education. *2017 International Conference on the Frontiers and Advances in Data Science (FADS)*, 92–97. <https://doi.org/10.1109/FADS.2017.8253202>
- Niculescu, A. I., & Banchs, R. E. (2015–September 13). Strategies to Cope with Errors in Human-Machine Spoken Interactions: Using Chatbots as Back-Off Mechanism for Task-Oriented Dialogues. *Proceedings of the Errors by Humans and Machines in Multimedia, Multimodal and Multilingual Data Processing Workshop (ERRARE 2015)*.
- Niculescu, A. I., van Dijk, B., Nijholt, A., Li, H., & See, S. L. (2013). Making Social Robots More Attractive: The Effects of Voice Pitch, Humor and Empathy. *International Journal of Social Robotics*, 5(2), 171–191. <https://doi.org/10.1007/s12369-012-0171-x>
- Nijholt, A. (2007, November 2). Conversational Agents and the Construction of Humorous Acts. In T. Nishida (Ed.), *Wiley Series in Agent Technology* (pp. 19–47). John Wiley & Sons, Ltd. <https://doi.org/10.1002/9780470512470.ch2>
- Nijholt, A., Niculescu, A. I., Valitutti, A., & Banchs, R. E. (2017). Humor in Human-Computer Interaction: A Short Survey. *Adjunct Proceedings of the 16th IFIP TC 13 International Conference on Human Computer Interaction (INTERACT)*, 192–214.
- O'Hara, K. J., Blank, D., & Marshall, J. (2015). Computational Notebooks for AI Education. *FLAIRS 2015*, 263–268.
- Okonkwo, C. W., & Ade-Ibijola, A. (2021). Chatbots Applications in Education: A Systematic Review. *Computers and Education: Artificial Intelligence*, 2, 100033. <https://doi.org/10.1016/j.caeai.2021.100033>
- O'Neill, R., & Russell, A. M. T. (2019). Stop! Grammar time: University Students' Perceptions of the Automated Feedback Program Grammarly. *Australasian Journal of Educational Technology*, 35(1), 42–56. <https://doi.org/10.14742/ajet.3795>
- OpenAI. (2022, November 30). *Introducing ChatGPT*. Retrieved March 3, 2023, from <https://openai.com/blog/chatgpt>
- OpenAI. (2023). *OpenAI API Documentation*. OpenAI API. Retrieved March 31, 2023, from <https://platform.openai.com/docs/introduction>

- Palan, S., & Schitter, C. (2018). Prolific.ac—A Subject Pool for Online Experiments. *Journal of Behavioral and Experimental Finance*, 17, 22–27. <https://doi.org/10.1016/j.jbef.2017.12.004>
- Pardo, A., & Siemens, G. (2014). Ethical and Privacy Principles for Learning Analytics. *British Journal of Educational Technology*, 45(3), 438–450. <https://doi.org/10.1111/bjet.12152>
- Pedrosa, D., Fontes, M. M., Araújo, T., Morais, C., Bettencourt, T., Pestana, P. D., Morgado, L., & Cravino, J. (2021). Metacognitive Challenges to Support Self-Reflection of Students in Online Software Engineering Education. *2021 4th International Conference of the Portuguese Society for Engineering Education (CISPEE)*. <https://doi.org/10.1109/CISPEE47794.2021.9507230>
- Peng, Z., & Ma, X. (2019). Exploring How Software Developers Work with Mention Bot in GitHub. *CCF Transactions on Pervasive Computing and Interaction*, 1(3), 190–203. <https://doi.org/10.1007/s42486-019-00013-2>
- Perkel, J. M. (2018). Why Jupyter Is Data Scientists' Computational Notebook of Choice. *Nature*, 563(7732), 145–147.
- Perkins, K., Adams, W., Dubson, M., Finkelstein, N., Reid, S., Wieman, C., & LeMaster, R. (2006). PhET: Interactive Simulations for Teaching and Learning Physics. *The Physics Teacher*, 44(1), 18–23. <https://doi.org/10.1119/1.2150754>
- Politz, J. G., Krishnamurthi, S., & Fisler, K. (2014). CaptainTeach: A Platform for In-Flow Peer Review of Programming Assignments. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*, 332. <https://doi.org/10.1145/2591708.2602687>
- Pottier, N. (2013). *Textit*.
- Prasser, F., Eicher, J., Spengler, H., Bild, R., & Kuhn, K. A. (2020). Flexible Data Anonymization Using ARX—Current Status and Challenges Ahead. *Software: Practice and Experience*, 50(7), 1277–1304. <https://doi.org/10.1002/spe.2812>
- Prince, M. (2004). Does Active Learning Work? A Review of the Research. *Journal of Engineering Education*, 93(3), 223–231. <https://doi.org/10.1002/j.2168-9830.2004.tb00809.x>
- Pritchard, D., & Vasiga, T. (2013). CS Circles: An in-Browser Python Course for Beginners. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 591–596.
- Project Jupyter, Blank, D., Bourgin, D., Brown, A., Bussonnier, M., Frederic, J., Granger, B., Griffiths, T., Hamrick, J., Kelley, K., Pacer, M., Page, L., Pérez, F., Ragan-Kelley, B., Suchow, J., & Willing, C. (2019). Nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook. *Journal of Open Source Education*, 2(16), 32.
- Ptaszynski, M., Dybala, P., Higuhi, S., Shi, W., Rzepka, R., & Araki, K. (2010). Towards Socialized Machines: Emotions and Sense of Humour in Conversational Agents. In Z.-U.-H. Usmani (Ed.), *Web Intelligence and Intelligent Agents* (pp. 173–205). InTech. <https://doi.org/10.5772/8384>
- Puggioni, M. P., Frontoni, E., Paolanti, M., Pierdicca, R., Malinverni, E. S., & Sasso, M. (2020). A Content Creation Tool for AR/VR Applications in Education: The ScoolAR Framework.

- In L. T. De Paolis & P. Bourdot (Eds.), *Augmented Reality, Virtual Reality, and Computer Graphics* (pp. 205–219). Springer. [https://doi.org/10.1007/978-3-030-58468-9\\_16](https://doi.org/10.1007/978-3-030-58468-9_16)
- Quiroga Pérez, J., Daradoumis, T., & Marquès Puig, J. M. (2020). Rediscovering the Use of Chatbots in Education: A Systematic Literature Review. *Computer Applications in Engineering Education*, 28(6), 1549–1565. <https://doi.org/10.1002/cae.22326>
- Radenski, A. (2006). “Python First”: A Lab-Based Digital Introduction to Computer Science. *SIGCSE Bulletin*, 38(3), 197–201.
- Reinmann, G. (2020). Outline of a Holistic Design-Based Research Model for Higher Education. *Educational Design Research*, 4(2). <https://doi.org/10.15460/eder.4.2.1554>
- Reynolds, L., & McDonell, K. (2021, February 15). *Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm*. arXiv: arXiv:2102.07350. Retrieved March 9, 2023, from <http://arxiv.org/abs/2102.07350>
- Ribeiro Guimarães, J. P. (2016). *Serious Game for Learning Code Inspection Skills* (Master's Thesis). Universidade do Porto. Porto, Portugal.
- Richter, S. (2012). Learning Tasks. In N. M. Seel (Ed.), *Encyclopedia of the Sciences of Learning* (pp. 1975–1979). Springer. [https://doi.org/10.1007/978-1-4419-1428-6\\_342](https://doi.org/10.1007/978-1-4419-1428-6_342)
- Ritz, E., & Grueneke, T. (2022). Learn Smarter, Not Harder – Exploring the Development of Learning Analytics Use Cases to Create Tailor-Made Online Learning Experiences. *Proceedings of the 55th Hawaii International Conference on System Sciences*, 921–930. <https://doi.org/10.24251/HICSS.2022.114>
- Robertson, T., & Simonsen, J. (2012). Challenges and Opportunities in Contemporary Participatory Design. *Design Issues*, 28(3), 3–9. [https://doi.org/10.1162/DESI\\_a\\_00157](https://doi.org/10.1162/DESI_a_00157)
- Rodríguez-Triana, M. J., Martínez-Monés, A., & Villagrà-Sobrino, S. (2016). Learning Analytics in Small-Scale Teacher-Led Innovations: Ethical and Data Privacy Issues. *Journal of Learning Analytics*, 3(1). <https://doi.org/10.18608/jla.2016.31.4>
- Ruch, W. (2008). Psychology of Humor. In V. Raskin (Ed.), *The Primer of Humor Research* (pp. 17–100). Mouton de Gruyter.
- Ruipérez-Valiente, J. A., Halawa, S., Slama, R., & Reich, J. (2020). Using Multi-Platform Learning Analytics to Compare Regional and Global MOOC Learning in the Arab World. *Computers & Education*, 146. <https://doi.org/10.1016/j.compedu.2019.103776>
- Rule, A., Tabard, A., & Hollan, J. D. (2018). Exploration and Explanation in Computational Notebooks. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12.
- Schanke, S., Burtch, G., & Ray, G. (2021). Estimating the Impact of ‘Humanizing’ Customer Service Chatbots. *Information Systems Research*, 32(3), 736–751. <https://doi.org/10.1287/isre.2021.1015>
- Schrepp, M., Hinderks, A., & Thomaschewski, J. (2017). Design and Evaluation of a Short Version of the User Experience Questionnaire (UEQ-S). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(6), 103–108. <https://doi.org/10.9781/ijimai.2017.09.001>
- Sclater, N. (2016). Developing a Code of Practice for Learning Analytics. *Journal of Learning Analytics*, 3(1). <https://doi.org/10.18608/jla.2016.31.3>



- Scott, C., & Jaramillo Cherrez, N. (2021, November 19). Supporting Language Learning With OERs and Open-Authoring Tools: In A. El Shaban & R. Abobaker (Eds.), *Advances in Educational Technologies and Instructional Design* (pp. 186–198). IGI Global. <https://doi.org/10.4018/978-1-7998-8267-1.ch010>
- Shum, H.-y., He, X.-d., & Li, D. (2018). From Eliza to XiaoIce: Challenges and Opportunities with Social Chatbots. *Frontiers of Information Technology & Electronic Engineering*, 19(1), 10–26. <https://doi.org/10.1631/FITEE.1700826>
- Shuster, K., Xu, J., Komeili, M., Ju, D., Smith, E. M., Roller, S., Ung, M., Chen, M., Arora, K., Lane, J., Behrooz, M., Ngan, W., Poff, S., Goyal, N., Szlam, A., Boureau, Y.-L., Kambadur, M., & Weston, J. (2022, August 10). *BlenderBot 3: A Deployed Conversational Agent That Continually Learns to Responsibly Engage*. arXiv: 2208.03188 [cs]. Retrieved March 3, 2023, from <http://arxiv.org/abs/2208.03188>
- Siemens, G., Gasevic, D., Haythornthwaite, C., Dawson, S., Buckingham Shum, S., Ferguson, R., Duval, E., Verbert, K., & Baker, R. S. J. d. (2011). *Open Learning Analytics: An Integrated & Modularized Platform* (tech. rep.). Society for Learning Analytics Research.
- Sjöström, J., Aghaee, N., Dahlin, M., & Ågerfalk, P. J. (2018). Designing Chatbots for Higher Education Practice. *Proceedings of the 2018 AIS SIGED International Conference on Information Systems Education and Research*.
- Slade, S., & Prinsloo, P. (2013). Learning Analytics: Ethical Issues and Dilemmas. *American Behavioral Scientist*, 57(10), 1510–1529. <https://doi.org/10.1177/0002764213479366>
- Smutny, P., & Schreiberova, P. (2020). Chatbots for Learning: A Review of Educational Chatbots for the Facebook Messenger. *Computers & Education*, 151, 103862. <https://doi.org/10.1016/j.compedu.2020.103862>
- Son, T., Xiao, T., Wang, D., Kula, R. G., Ishio, T., & Matsumoto, K. (2021, August 18). *More Than React: Investigating The Role of EmojiReaction in GitHub Pull Requests*. arXiv: 2108.08094. Retrieved January 21, 2022, from <http://arxiv.org/abs/2108.08094>
- Song, D., Oh, E. Y., & Rice, M. (2017). Interacting with a Conversational Agent System for Educational Purposes in Online Courses. *2017 10th International Conference on Human System Interactions (HSI)*, 78–82. <https://doi.org/10.1109/HSI.2017.8005002>
- Song, X., Goldstein, S. C., & Sakr, M. (2020). Using Peer Code Review as an Educational Tool. *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 173–179.
- Stalhane, T., Kutay, C., Al-Kilidar, H., & Jeffery, R. (2004). Teaching the Process of Code Review. *Proceedings of the 2004 Australian Software Engineering Conference (ASWEC '04)*, 271–278.
- Starr, C. R. (2018). “I’m Not a Science Nerd!”: STEM Stereotypes, Identity, and Motivation among Undergraduate Women. *Psychology of Women Quarterly*, 42(4), 489–503.
- Steiner, C. M., Kickmeier-Rust, M. D., & Albert, D. (2016). LEA in Private: A Privacy and Data Protection Framework for a Learning Analytics Toolbox. *Journal of Learning Analytics*, 3(1). <https://doi.org/10.18608/jla.2016.31.5>

## Bibliography

---

- Sweeney, L. (2002). K-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05), 557–570. <https://doi.org/10.1142/S0218488502001648>
- Tallyn, E., Fried, H., Gianni, R., Isard, A., & Speed, C. (2018). The Ethnobot: Gathering Ethnographies in the Age of IoT. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–13. <https://doi.org/10.1145/3173574.3174178>
- Tammeleht, A. (2022). Design Principles for Developing Online Ethics Resources - the Outcome of Holistic DBR Process. *Educational Design Research*, 6(1). <https://doi.org/10.15460/eder.6.1.1713>
- Tang, M. (2011, August 22). *Caesar: A Social Code Review Tool for Programming Education* (Master's Thesis). Massachusetts Institute of Technology. Cambridge, MA, USA.
- Tavares, R., Vieira, R. M., & Pedro, L. (2020). A Participatory Framework Proposal for Guiding Researchers through an Educational Mobile App Development. *Research in Learning Technology*, 28. <https://doi.org/10.25304/rlt.v28.2370>
- Taylor, J., & Mazlack, L. (2005). Toward Computational Recognition of Humorous Intent. In B. G. Bara, L. Barsalou, & M. Bucciarelli (Eds.), *Proceedings of the 27th Annual Conference of the Cognitive Science Society* (pp. 2166–2171). Lawrence Erlbaum Associates, Inc.
- Tegos, S., Demetriadis, S., Papadopoulos, P. M., & Weinberger, A. (2016). Conversational Agents for Academically Productive Talk: A Comparison of Directed and Undirected Agent Interventions. *International Journal of Computer-Supported Collaborative Learning*, 11(4), 417–440. <https://doi.org/10.1007/s11412-016-9246-2>
- Telang, P. R., Kalia, A. K., Vukovic, M., Pandita, R., & Singh, M. P. (2018). A Conceptual Framework for Engineering Chatbots. *IEEE Internet Computing*, 22(6), 54–59. <https://doi.org/10.1109/MIC.2018.2877827>
- Thorson, J. A., & Powell, F. (1993). Sense of Humor and Dimensions of Personality. *Journal of Clinical Psychology*, 49(6), 799–809.
- Tómasdóttir, K. F., Aniche, M., & van Deursen, A. (2020). The Adoption of JavaScript Linters in Practice: A Case Study on ESLint. *IEEE Transactions on Software Engineering*, 46(8), 863–891. <https://doi.org/10.1109/TSE.2018.2871058>
- Tomorrow Corporation. (2015). *Human Resource Machine*.
- Towhidnejad, M., & Salimi, A. (1996). Incorporating a Disciplined Software Development Process in to Introductory Computer Science Programming Courses: Initial Results. *Technology-Based Re-Engineering Engineering Education Proceedings of Frontiers in Education FIE'96 26th Annual Conference*, 2, 497–500. <https://doi.org/10.1109/FIE.1996.572893>
- Trivedi, J. (2019). Examining the Customer Experience of Using Banking Chatbots and Its Impact on Brand Love: The Moderating Role of Perceived Risk. *Journal of Internet Commerce*, 18(1), 91–111. <https://doi.org/10.1080/15332861.2019.1567188>
- Trytten, D. A. (2005). A Design for Team Peer Code Review. *ACM SIGCSE Bulletin*, 37(1), 455–459. <https://doi.org/10.1145/1047124.1047492>
- Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59(236), 433–460. <https://doi.org/10.1093/mind/LIX.236.433>

- Unicode. (2021). *Unicode CLDR Project* (Version 40.0).
- Urquhart, C. (2012). *Grounded Theory for Qualitative Research: A Practical Guide*. Sage.
- Valdiviezo, A. D., & Crawford, M. (2020). Fostering Soft-Skills Development through Learning Experience Platforms (LXPs). In *Handbook of Teaching with Technology in Management, Leadership, and Business* (pp. 312–321). Edward Elgar Publishing. <https://doi.org/10.4337/9781789901658.00040>
- Van Goidsenhoven, S., Bogdanova, D., Deeva, G., vanden Broucke, S., De Weerd, J., & Snoeck, M. (2020). Predicting Student Success in a Blended Learning Environment. *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, 17–25. <https://doi.org/10.1145/3375462.3375494>
- van Rossum, G., Warsaw, B., & Coghlan, N. (2001). *Style Guide for Python Code* (PEP No. 8).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, December 5). *Attention Is All You Need*. arXiv: 1706.03762 [cs]. Retrieved February 24, 2022, from <http://arxiv.org/abs/1706.03762>
- Vermeulen, J., Luyten, K., van den Hoven, E., & Coninx, K. (2013). Crossing the Bridge over Norman's Gulf of Execution: Revealing Feedforward's True Identity. *Proceedings of the 2013 CHI Conference on Human Factors in Computing Systems*.
- Villegas-Ch, W., Arias-Navarrete, A., & Palacios-Pacheco, X. (2020). Proposal of an Architecture for the Integration of a Chatbot with Artificial Intelligence in a Smart Campus for the Improvement of Learning. *Sustainability*, 12(4). <https://doi.org/10.3390/su12041500>
- Wagner, I., & Eckhoff, D. (2018). Technical Privacy Metrics: A Systematic Survey. *ACM Computing Surveys*, 51(3). <https://doi.org/10.1145/3168389>
- Wallace, R. (2002). *Pandorabots*.
- Wallace, R. S. (2009). The Anatomy of A.L.I.C.E. In R. Epstein, G. Roberts, & G. Beber (Eds.), *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer* (pp. 181–210). Springer. [https://doi.org/10.1007/978-1-4020-6710-5\\_13](https://doi.org/10.1007/978-1-4020-6710-5_13)
- Wang, A. Y., Mittal, A., Brooks, C., & Oney, S. (2019). How Data Scientists Use Computational Notebooks for Real-Time Collaboration. *Proceedings of the ACM on Human-Computer Interaction*, 3, 1–30. <https://doi.org/10.1145/3359141>
- Wang, F., & Hannafin, M. J. (2005). Design-Based Research and Technology-Enhanced Learning Environments. *Educational Technology Research and Development*, 53(4), 5–23. <https://doi.org/10.1007/BF02504682>
- Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of Programming Language Learning Based on Peer Code Review Model: Implementation and Experience Report. *Computers & Education*, 59(2), 412–422. <https://doi.org/10.1016/j.compedu.2012.01.007>
- Wanzer, M. B., & Frymier, A. B. (1999). The Relationship Between Student Perceptions of Instructor Humor and Students' Reports of Learning. *Communication Education*, 48(1), 48–62. <https://doi.org/10.1080/03634529909379152>

## Bibliography

---

- Weizenbaum, J. (1966). ELIZA—A Computer Program For the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM*, 9(1), 36–45. <https://doi.org/10.1145/365153.365168>
- Wessel, M., de Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P., & Gerosa, M. A. (2018). The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW). <https://doi.org/10.1145/3274451>
- Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2020a). Effects of Adopting Code Review Bots on Pull Requests to OSS Projects. *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. <https://doi.org/10.1109/ICSME46990.2020.00011>
- Wessel, M., Serebrenik, A., Wiese, I., Steinmacher, I., & Gerosa, M. A. (2020b). What to Expect from Code Review Bots on GitHub?: A Survey with OSS Maintainers. *Proceedings of the 34th Brazilian Symposium on Software Engineering*, 457–462. <https://doi.org/10.1145/3422392.3422459>
- West, R., & Horvitz, E. (2019). Reverse-Engineering Satire, or “Paper on Computational Humor Accepted Despite Making Serious Advances”. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 7265–7272. <https://doi.org/10.1609/aaai.v33i01.33017265>
- Wieggers, K. E. (2002). *Peer Reviews in Software: A Practical Guide*. Addison-Wesley.
- Wiley, D. A. (2002). Connecting Learning Objects to Instructional Design Theory: A Definition, a Metaphor, and a Taxonomy. In *The Instructional Use of Learning Objects* (pp. 3–23).
- Winkler, R., Hobert, S., Salovaara, A., Söllner, M., & Leimeister, J. M. (2020). Sara, the Lecturer: Improving Learning in Online Education with a Scaffolding-Based Conversational Agent. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3313831.3376781>
- Winkler, R., & Soellner, M. (2018). Unleashing the Potential of Chatbots in Education: A State-Of-The-Art Analysis. *Academy of Management Proceedings*, 2018(1), 15903. <https://doi.org/10.5465/AMBPP.2018.15903abstract>
- Wollny, S., Schneider, J., Di Mitri, D., Weidlich, J., Rittberger, M., & Drachsler, H. (2021). Are We There Yet? - A Systematic Literature Review on Chatbots in Education. *Frontiers in Artificial Intelligence*, 4, 654924. <https://doi.org/10.3389/frai.2021.654924>
- Wu, S.-R. (2008). Humor and Empathy: Developing Students’ Empathy through Teaching Robots to Tell English Jokes. *Proceedings of the 2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, 213–214. <https://doi.org/10.1109/DIGITEL.2008.27>
- Wyrich, M., Ghit, R., Haller, T., & Müller, C. (2021). Bots Don’t Mind Waiting, Do They? Comparing the Interaction With Automatically and Manually Created Pull Requests. *Proceedings of the 2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, 6–10.
- Xu, G., Qi, C., Yu, H., Xu, S., Zhao, C., & Yuan, J. (2019). Detecting Sensitive Information of Unstructured Text Using Convolutional Neural Network. *2019 International Conference*

- on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 474–479. <https://doi.org/10.1109/CyberC.2019.00087>
- Yadav, A., Stephenson, C., & Hong, H. (2017). Computational Thinking for Teacher Education. *Communications of the ACM*, 60(4).
- Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing Computational Thinking in Education Courses. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 465–470.
- Yang, S., & Evans, C. (2019). Opportunities and Challenges in Using AI Chatbots in Higher Education. *Proceedings of the 2019 3rd International Conference on Education and E-Learning*, 79–83. <https://doi.org/10.1145/3371647.3371659>
- Zagalsky, A., Feliciano, J., Storey, M.-A., Zhao, Y., & Wang, W. (2015). The Emergence of GitHub as a Collaborative Platform for Education. *CSCW 2015*, 1906–1917. <https://doi.org/10.1145/2675133.2675284>
- Zaharias, P., & Poylymenakou, A. (2009). Developing a Usability Evaluation Method for e-Learning Applications: Beyond Functional Usability. *International Journal of Human-Computer Interaction*, 25(1), 75–98. <https://doi.org/10.1080/10447310802546716>
- Zakas, N. C. (2013). *ESLint*.
- Zastre, M. (2019). Jupyter Notebook in CS1: An Experience Report. *Proceedings of the Western Canadian Conference on Computing Education*, 1–6.
- Zhao, R., Romero, O. J., & Rudnicky, A. (2018). SOGO: A Social Intelligent Negotiation Dialogue System. *Proceedings of the 18th International Conference on Intelligent Virtual Agents*, 239–246. <https://doi.org/10.1145/3267851.3267880>
- Zhao, T. Z., Wallace, E., Feng, S., Klein, D., & Singh, S. (2021, June 10). *Calibrate Before Use: Improving Few-Shot Performance of Language Models*. arXiv: arXiv:2102.09690. Retrieved March 9, 2023, from <http://arxiv.org/abs/2102.09690>
- Zong, Z., Wang, Y., & Schunn, C. D. (2021). Why Students Want to Provide Feedback to their Peers: Drivers of Feedback Quantity and Variation by Type of Course. *Journal of Psychology in Africa*, 31(4), 336–343.

# Juan Carlos FARAH

[juancarlos.farah@epfl.ch](mailto:juancarlos.farah@epfl.ch) | [Google Scholar](#)

**Current Research Interests:** *Conversational Agents, Natural Language Processing, Human Computer Interaction, Attribution of Consciousness, Network Neuroscience, Second-Person Neuroscience, Information Theory*

## Selected Education

**École Polytechnique Fédérale de Lausanne (EPFL), 2023** Lausanne, CH  
PhD in Robotics, Control, and Intelligent Systems  
Thesis: *A Conceptual Framework for Integrating Conversational Agents in Digital Education*

**Imperial College London, Class of 2015** London, UK  
MSc in Computing Science | Distinction

**Stanford University, Summer Session 2013** Stanford CA, US  
Certificate in Computer Science | GPA: 3.6 (4.0 scale)

**Harvard University, Class of 2008** Cambridge MA, US  
AB in Economics, Language Citation in French | GPA: 3.5 (4.0 scale)

**Colegio Franklin Delano Roosevelt, Class of 2004** Lima, PE  
IB Diploma, Valedictorian, Yale Book Award | GPA: 4.3 (4.0 scale)

## Summer / Winter Schools

- Laboratoire d'Informatique de Grenoble, [Advanced Language Processing School](#), January 2023, Virtual.
- Canadian Institute for Advanced Research, [Neuroscience of Consciousness Winter School](#), December 2022, Cancún, MX.
- Lemanic Neuroscience Doctoral School, [Brain Dynamics on the Connectome Summer School](#), October 2021, Virtual.

## Selected Experience

**École Polytechnique Fédérale de Lausanne (EPFL) – Interaction Systems Group** Lausanne, CH  
[react.epfl.ch](http://react.epfl.ch)

*Software Engineer, Researcher* | April 2017 – Present

Technologies: Python, Node.js, MongoDB, React.js, Electron.js, React Native, ES6+, Redis, HTML, CSS, Docker, Git, AWS

- Develop front and backend applications for a digital education ecosystem supporting online learning activities.
- Design and maintain a development framework and toolkit to facilitate the creation of digital resources for education.
- Conduct, publish, and present research on human-machine interaction, data privacy, and security.

**College of Engineering and Architecture of Fribourg – Institute of Smart and Secure Systems** Fribourg, CH  
[heia-fr.ch/en/applied-research/institutes/isis/](http://heia-fr.ch/en/applied-research/institutes/isis/)

*Lecturer, Researcher* | September 2019 – February 2020

Technologies: Node.js, React.js, Python, MATLAB

- Conduct research in human-computer interaction and digital education with a focus on chatbots.
- Lecture final-year bachelor students on human-machine interaction, web design, and software development.
- Design and lead experiments both in the lab and in the classroom.

**Graasp Association – Non-Profit Organization Promoting Digital Education** Lausanne, CH  
[graasp.org](http://graasp.org)

*Co-Founder, VP Research* | October 2019 – Present

Technologies: Node.js, MongoDB, React.js, Electron.js, React Native, ES6+, Serverless, HTML, CSS, Docker, Git, AWS

- Participate in the management of a non-profit organization promoting innovation in digital education.
- Lead software engineering, developer operations, quality assurance, developer advocacy, and training.
- Support with grant and funding applications.

**MongoDB – Leading NoSQL Database** London, UK  
[mongodb.com](http://mongodb.com)

*Consultant, Teaching Assistant* | September 2013 – June 2018

Technologies: MongoDB, Python, JavaScript

- Worked with in-house and community teams to teach, promote, and translate courses offered by [MongoDB University](#).
- Moderated discussions, gathered feedback, and answered student queries through the courses' forums.
- Assisted in the proofing, debugging, and improvement of lecture, quiz, homework, and exam materials.

**Turismo Civa – Bus Company****Lima, PE**[civa.com.pe](http://civa.com.pe)*Consultant, Software Engineer* | July 2015 – July 2019

Technologies: JavaScript, React.js, Backbone.js, ES6+, Java, Spring MVC, MySQL, HTML, CSS, Git, AWS

- Designed, developed, and deployed web-based sales and operations software for one of Peru's largest bus companies.
- Spearheaded DevOps, established agile development processes and migrated legacy in-house systems to AWS.
- Co-managed a team of five, ensuring all stakeholders were aligned with progress and changes in scope.

**Boxagon – Social Commerce Platform****London, UK**[crunchbase.com/organization/boxagon](https://crunchbase.com/organization/boxagon)*Co-Founder, Software Engineer* | August 2012 – September 2015

Technologies: MongoDB, jQuery, JavaScript, Java, Spring MVC, HTML, CSS, Git, CentOS

- Led frontend and product development for a social commerce platform that helped users create and share product bundles.
- Contributed to backend application development, system administration, quality assurance, and UI/UX design.
- Carried out the daily database administration, performance monitoring, and data science tasks.

**Mimanzana – Digital Marketing Agency****Lima, PE**[mimanzana.com](http://mimanzana.com)*Member of the Board, Interim Product Lead* | January 2013 – December 2015

Technologies: MongoDB, jQuery, JavaScript, HTML, CSS, Git, CentOS

- Devised and executed commercial growth strategy leading to focus on digital marketing and social media management.
- Organized the implementation of best software engineering practices for the in-house web development team.
- Established partnerships, processes, and workflows to allow outsourcing of non-core business activities.

**Vostu – Latin American Online Social Network****Cambridge MA, US**[crunchbase.com/organization/vostu](https://crunchbase.com/organization/vostu)*Co-Founder, Marketing Director* | November 2006 – November 2007

Technologies: Java, MySQL, HTML, CSS

- Worked with a team of six to build and launch the first Latin American online social network (later a gaming platform).
- Collaborated with the programming team in the design, translation, and testing of the application.
- Managed branding, user acquisition, viral marketing, and growth hacking efforts in South America.

**Selected Publications**

- [A Blueprint for Integrating Conversational Agents in Education](#), CUI 2022. (Honorable Mention Best Short Paper)
- [Supporting Developers in Creating Web Apps for Education](#), HEAd 2022. (Best Student Paper)
- [Impersonating Chatbots in a Code Review Exercise](#), EDUCON 2022. (Best Student Paper Co-Award)
- [Integrating Code Reviews into Online Lessons to Support Software Engineering Education](#), ICL 2022.
- [An Exploratory Study of Reactions to Bot Comments on GitHub](#), BotSE 2022.
- [Conveying the Perception of Humor in Human-Chatbot Interaction](#), HAI 2021.
- [Bringing Computational Thinking to non-STEM Undergraduates](#), ECTEL 2020.
- [A Blueprint for a Blockchain-Based Architecture](#), ICAIT 2018.
- [Implementation of Attentional Bistability in a Computational Model of the Dragonfly Visual System](#), CCN 2017.
- [A Teacher Survey on Educational Data Management Practices: Tracking and Storage of Activity Traces](#), ECTEL 2017.

**Selected Talks**

- [Future Learning Initiative Colloquium](#), *Conversational Agents in Digital Education*, May 2023, Zurich, CH.
- Center for Learning Sciences, [Active Learning: From Concepts to Blended Implementation](#), May 2019, Lausanne, CH.
- EPFL, [Trust, Privacy, and the Blockchain](#), Yearly Lecture First Presented in May 2018, Lausanne, CH.
- Next-Lab Summer School, [Learning Analytics & Privacy](#), July 2017, Marathon, GR.
- MongoDB User Group, [Kickstarting Your Mongo Education with MongoDB University](#), June 2014, London, UK.

**Skills and Interests****Technologies:** MongoDB, ES6+, Node.js, React.js, Redux, Python, Java, HTML, CSS, Git, Docker, MATLAB, AWS, Linux.**Languages:** Bilingual in Spanish & English, fluent in French, proficient in Portuguese & Italian, basic in Arabic & Greek.**Interests:** Guitar, Maps, Cosmology, Coffee, Running, Football, Tennis, Golf.