Functional Programming For All! Scaling a MOOC for Students and Professionals Alike

Heather Miller
EPFL, Switzerland

Philipp Haller
Typesafe, Switzerland

Lukas Rytz
EPFL, Switzerland

Martin Odersky
EPFL, Switzerland

ABSTRACT

Massive open online courses (MOOCs) have launched a scale shift in higher education, with several individual MOOCs now boasting tens or hundreds of thousands of participants worldwide. Our MOOC on the principles of functional programming has more than 100,000 registered students to date, and boasts one of the highest rates of completion (19.2%) for its size. In this paper, we describe our experience organizing this popular MOOC, and demonstrate how providing innovative supporting tools (IDE plugins, testing frameworks, interactive build tools, automated cloud-based graders, style checkers) and considering key human-computer interaction factors potentially contributed to this markedly high completion rate. We collect an unprecedented volume of course statistics and survey results and have made them available, along with scripts for generating interactive web-based visualizations, as an open-source project.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Experimentation, Management

Keywords

Software engineering education, programming education, automated grading, MOOC

1. INTRODUCTION

Massive open online courses (MOOCs) have the potential to revolutionize teaching and learning in both academia and industry. Courses that teach tools and technologies accepted by and relevant to industry have proven a particularly attractive option for students and professionals seeking skills that they can apply immediately in practice.

ICSE '14, May 31 – June 7, 2014, Hyderabad, India

In this paper, we describe our experience organizing a MOOC on the principles of functional programming, now in its third iteration, that has attracted more than 100,000 registered students. What sets the course apart from other MOOCs related to programming or software engineering is that it has enjoyed one of the highest completion rates (19.2%) of any MOOC worldwide[1]. Completion rates of around 7% are standard [14].

In contrast to other similar SE-related MOOCs, we have leveraged the popularity of the course to collect an unparalleled volume of course statistics and survey results. This paper describes our data set, consisting of the following components:

- A large-scale and detailed survey with over 12,000 respondents worldwide, covering their experience in the course as well as their background, software development experience, and skills;
- The results from a specialized on-campus course evaluation of our MOOC, where over 150 undergraduate students at EPFL participated in the MOOC instead of taking the first half of an on-campus offering of a longer version of the same course;
- Anonymous statistics about student engagement and performance throughout each iteration of our course so far.

The data set has been made available in form of an open-source project, providing additional scripts and tools for generating interactive web-based visualizations[2]. To the best of our knowledge, data sets on MOOC statistics and survey results of a comparable size have not been published before.

This data provides insight into (a) the interaction of participants with the automated grading system, (b) the effect of different grading policies, (c) the profile, background, and motivation of participants, and (d) the acceptance and student evaluation of a MOOC, both in isolation and as integrated as a significant component of an on-campus software-development course.

The data shows that SE-related MOOCs can be very useful not just for students, but also for professional software engineers. Moreover, the survey results provide new insights into the role of new technologies and tools for effective

[1] At the time the first iteration of the course ended in November 2012. Today, it still has one of the highest rates of completion given its size [8].

[2] `https://github.com/heathermiller/progfun-stats`

teaching and learning. We find that providing innovative course-supporting tools (IDE plugins, testing frameworks, interactive build tools, automated cloud-based graders, style checkers) and considering human-computer interaction factors have helped make distance learning effective and enjoyable. In fact, our on-campus students who followed the MOOC in place of the first half of our functional programming course overwhelmingly preferred the online format over the traditional format of the course, thanks mostly to the tighter feedback loop made possible by the tools.

The remainder of the paper is organized as follows. Section 2 covers the format and organization of our MOOC, and interaction between the MOOC and the on-campus course. In Section 3 we describe the tools used to support teaching and learning, as well as the integration with Coursera, the MOOC infrastructure. In Section 4 we describe the data set and interactive tools for experimenting with it, released as an open-source project. In Section 5 we present an evaluation and analysis of some of this data. Finally, we survey related work in Section 6 and conclude in Section 7.

2. COURSE FORMAT

The objective of our MOOC, *Functional Programming Principles in Scala* (or "progfun" for short), is to introduce functional programming principles. As such, it covers topics such as recursion, persistent/immutable data structures, higher-order functions, and pattern matching. Some of the material is based on the well-known *Structure and Interpretation of Computer Programs* MIT course [1]. Instead of Scheme, the Scala programming language [11] is used throughout the course, both in the lectures and in the homework assignments.

The first half of the course (*online*, seven weeks) is offered as a MOOC worldwide and to EPFL undergraduates alike, whereas the second half of the course (*offline*, seven weeks) comprises a continuation of the course material exclusive to EPFL students, intended for second year students in Computer Science.

2.1 Lectures

The course lectures are provided in the form of short online videos, each about 8–12 minutes in duration. Each week, 5–7 videos are released. Students are able to speed up or slow down the video to suit their preferred pace.

2.2 Assignments

The lectures are accompanied by interactive, not-for-credit quizzes requiring real-time participation from the students as they watch the videos, as well as weekly or biweekly assignments, all of which consist of programming exercises.

Testing and submission.

Several provided tools enable students to interact with the assignments and their submission. The interactive build tool [17] allows students to run unit tests and to directly submit their solutions to our cloud-based grading infrastructure. Submitted solutions are processed in the cloud in two steps. First, the solution is analyzed with a custom code style checker. This style checker is based on Scalastyle [4], which is similar to Checkstyle [21] for Java. Submissions that are not written in the functional style taught in the course are penalized.

In a second step, the solution is tested using a comprehensive test suite. The test suite that is used for grading is *secret*; only a small scaffolding containing a couple of sample tests is provided to the students. This ensures that their solutions are compatible with our test suite, and also creates an incentive for students to build a comprehensive test suite of their own.

Interactive feedback.

Upon each submission the student receives feedback about how the code fared on the secret test suite. This feedback contains non-obvious hints about where the submitted code could be logically incorrect, causing failing tests, and lists the reduction in points. Additionally, the result of running our style checker is included to give feedback on aspects of style that need to be improved.

Students can submit revisions of their solution as often as they like until the submission deadline without penalty. This policy was changed for the second iteration of the course where assignments could only be re-submitted up to five times (the later submission attempts would earn no credit), as we will discuss below. Only the best of the student's submissions is considered: the grade of an assignment is the grade of the best submission for that assignment.

Certificate of completion.

Students who obtain more than 60% of all possible points (or 48 points out of a total of 80 possible points) receive a certificate of completion. In addition, those who earn more than 80% of all possible points (or 64 points out of a total of 80 possible points) receive a so-called "certificate of distinction".

2.3 Additional elements of the EPFL course

The EPFL course is a true superset of the MOOC: EPFL students follow the online lectures and quizzes just like any other participant, in our case on the Coursera platform. The elements exclusive to EPFL students consisted of:

- **in-person exercise sessions** where students can work on programming assignments or review the course material and ask teaching assistants questions; and
- **written exams** which accounted for a majority of the final grade on students' academic record. These exams were essential to satisfy the requirements of their degree.
- **offline second-half of the course** where subsequent course material was presented to students as a traditional on-campus course.

2.4 Commercial tutorial sessions

The course format lends itself to additional, commercial offerings. For the third iteration of our course on functional programming principles, Typesafe, Inc., has introduced supervised tutorial sessions accompanying selected Coursera classes. In weekly, one-hour tutorial sessions, experts from Typesafe provide in-depth answers to questions about the course material and discuss solutions to homework assignments past the deadline. Tutorial groups are small (10 participants max) in order to meet individual mentoring needs as much as possible. Tutorial session slots are offered in both European and American time zones.

3. TOOLING AND GRADING SYSTEM

The only mandatory tool that students are required to use is sbt [17], a standard build tool for Scala and Java. Sbt is used to compile, run, test and submit the code for the assignments. The build tool can also generate project definitions for the Scala IDE for eclipse and IntelliJ IDEA, which gives the students effortless access to those IDEs. The developers of the Scala IDE at Typesafe introduced Scala worksheets for the start of the first iteration of the course. A worksheet is a single source file in which each line is evaluated as a separate expression and the result is presented in a separate column on the side. This feature makes exploring the language and library features even easier than a classical read-eval-print-loop, and the sessions can be saved for future reference.

Automated Grading System.

The automated grading system for each assignment is based on two components: a style checker and a test suite of unit tests. The style checker is based on Scalastyle [4] and allows identifying syntactic constructs that are discouraged in the context of the course on functional programming principles. Examples are mutable variables, return statements, the `null` value, while loops, magic numbers, overlong lines or non-standard capitalization.

The majority of the grade is obtained by running a comprehensive test suite that is private to the automated graders on each student's submission. Tests are executed by using ScalaTest [18], a unit testing framework for Scala and Java. The framework was customized so that each unit test can be assigned an individual weight and the framework computes the overall score of the successful tests. Some of the unit tests are handed out to the students as part of the assignment, which ensures that the function signatures in their code are compatible with the test suite.

A subset of the unit tests are implemented using a custom-written Java virtual machine agent [12] that instruments the bytecode to record statistics about invoked methods. This allows enforcing constraints that could not be tested with a classical unit test, for example that a method is implemented in terms of another one, or that some specific libraries are not being used.

Security.

Executing code that was written and uploaded by arbitrary students has obvious security issues: an adversary could try to read the private test suite, modify the computed grade or sabotage the grading infrastructure. To prevent such issues we rely on the security manager infrastructure provided by the Java platform [13]. This facility allows disabling sensitive functionality with detailed granularity for specific parts of the executed code. This allows executing the testing framework with elevated privileges (it requires access to IO and reflection), but the code from the students is executed in a restricted environment within each test case. We may also add that we are not aware of any attempts to compromise our grading infrastructure.

Infrastructure for Grading.

The submission scripts uploads the source code to the servers of Coursera, the MOOC provider used for this class, using their custom REST HTTP API. This API does not only allow uploading solutions, but it also provides access for grading scripts to download the submitted assignments and upload a grade and feedback for the student.

Given the large number of participants and the submission model that gives the students an arbitrary (or even unlimited) number of attempts at each assignment, the use of a scalable cloud computing service was a natural choice to implement the grading infrastructure. The goal is to provide the students feedback for each submission within 15–20 minutes. For this course, the grading infrastructure is implemented using Amazon Web Services (AWS). This service can be configured to automatically start up or terminate virtual machines based on custom workload measures. The machines are configured such that they automatically start downloading and grading assignments from the work queue provided by Coursera once they are booted.

The source code of our grading and submission infrastructure has been developed for one specific MOOC and cannot be reused directly for other online courses. Most of the implementation is written as a plugin for the sbt build tool, which can be used for Scala and Java projects. The same applies to the customized test framework. Finally, the scripts interact with the HTTP API of Coursera and would need to be adapted for other providers. However, the main concepts of our infrastructure apply to programming courses in general and many of the ideas we implemented can be re-used.

4. SURVEY AND DATA SET

4.1 Survey

After the completion of each iteration of our MOOC we sent a survey to all registered participants. For the Fall 2012 iteration of the course, we received responses from 7,492 out of about 50,000 registered students. For the Spring 2013 iteration of the course, we received responses from 4,595 out of about 37,000 registered students. Thus, we collected results from a total of 12,087 respondents.

The survey consisted of questions included but not limited to the following:

- **What's your age group?** (Possible choices: 10-17, 18-24, 25-34, 35-44, 45-54, 55-64, 65+)
- **What country do you live in?**
- **What's your highest degree?** (Possible choices: No High School (or equivalent), Some High School (or equivalent), High School (or equivalent), Some College (or equivalent), Bachelor's Degree (or equivalent), Master's Degree (or equivalent), Doctorate Degree (or equivalent), Other)
- **Did you finish the course?**
- **How difficult did you find the homework assignments?** (Possible choices: 1 - Too Easy, 2, 3, 4, 5 - Too Hard)
- **Where do you plan to apply what you've learned in this course?** (Possible choices: Personal projects, Individual project at work, Team project at work, University projects, No application plans, Attended for general interest)

- **What experience do you have with other programming languages or paradigms?** (asked once each for Java, C/C++/Objective-C, Python/Ruby/Perl/other scripting language, C#/.NET, JavaScript, Haskell/OCaml/ML/F#, Lisp/Scheme/Clojure, possible choices: No experience / not seen it at all, I've seen and understand some code, I have some experience writing code, I'm fluent, I'm an expert)

```
object WorthItBarGraph
    extends SimpleBarGraphFactory {
  import CourseraData.worthIt

  val name = "worth-it.html" // file name
  val label = "Percentage"   // label on y axis

  override val width = 250
  override val height = 250
  override val maxy = 70

  def data: List[(String, Int)] = {
    val counts = getFreqs(worthIt)
      .sortBy(_._1)
      .map { case (name, value) =>
        (name.toString, (value.toDouble /
          worthIt.length * 100).round.toInt)
      }

    val correctedLabels: List[(String, Int)] =
      List(("1 Disagree", counts(0)._2)) ++
        counts.drop(1).take(3) ++
        List(("5 Agree",counts(4)._2))

    correctedLabels
  }
}
```

Figure 1: Generating a bar graph which represents how "worth it" the course was for students.

All survey results have been released as part of the PROGFUN-STATS open-source project [9]. The survey data is available in simple plain text formats, such as tab-separated values. Thus, it is easy to process and analyze the data in any general-purpose programming language. In addition to the raw data set, PROGFUN-STATS provides a library extension for the Scala programming language [11] which makes it easy to generate interactive, web-based visualizations[3].

For example, Figure 1 shows how to create a simple bar chart that represents how "worth it" the course was for students, on a scale from 1 (disagree) to 5 (agree).

4.2 EPFL course evaluation

Upon the conclusion of each semester, EPFL undergraduates complete an evaluation for each of their courses, typically as an assessment of course quality for the Computer Science department.

After spending half of a semester learning functional programming principles with a MOOC, and another half of a semester in a traditional classroom setting, the progfun undergraduates were asked to complete a specialized course evaluation designed to compare the effectiveness of the two approaches.

[3]There is also a website showcasing more data and visualizations than shown in this paper, `http://lampwww.epfl.ch/~hmiller/progfun`

5. EVALUATION

In this section, we analyze and evaluate a selection of the collected data. First, we'll examine the role of the MOOC's tooling infrastructure (*e.g.,* IDEs, or the automated grader) in making distance learning more effective. Next, we'll examine the background and motivation of MOOC participants which helps explain whether MOOCs can be useful to professional software engineers. Finally, as a third step, we evaluate the acceptance of our MOOC in a university setting.

5.1 Tools for effective distance learning

In this section we evaluate the use of two important tools: (a) the automated grading system and (b) the Scala IDE for Eclipse plugin with the Worksheet IDE, which was released specifically for use in the first iteration of the course.

5.1.1 Automated grading system

Before evaluating the grading system, we first clarify the respective grading policies used. As mentioned in Section 2.1 the grading policy changed between the Fall 2012 and the Spring 2013 iterations of the course. In the Fall 2012 iteration, students could re-submit solutions as often as they wanted without penalty until the submission deadline. In the Spring 2013 iteration, students were allowed to re-submit only up to five times; the sixth and later submissions would not give any credit. In both iterations, the (valid) submission with the highest grade determined the final grade for that assignment.

Usage of the submission system.

Figure 2 shows the correlation between the achieved score for one representative assignment (an implementation of Huffman coding using pattern matching) and the number of submissions required of each student to achieve that score. For both the Fall 2012 and the Spring 2013 iterations of the course, the largest concentration of students fell at or close to the perfect score (10). In fact, both plots in Figure 2 show that for a large number of students, only five submissions or less were required to achieve their final/best score, justifying the change made in Spring 2013.

For the Fall 2012 plot, for example, each bright yellow dot along the y-axis and concentrated around a score of 10 represents approximately 1,200-1,700 individual user submissions. Above 5 submissions however, the density of individual user submissions drastically drops to approximately 250 or fewer individual user submissions. This suggests that to achieve a higher score (even when submissions are unlimited), a higher number of submissions was *not usually necessary.*

Nevertheless, during the Fall 2012 iteration, there were still thousands of students with 10 or even 15 submissions. The results for the Spring 2013 iteration are quite different: most people achieved their final score after only two or four submissions. This difference stems from the adoption of the different grading policy used in the second iteration of the course: no credit for the sixth and later submissions. This is clearly reflected in the changed submission statistics.

However, interestingly, there are still a significant number of students who also submitted six or more times (thus earning no additional credit); this is a clear indication that the automated grading and feedback was also used for learning without improving one's score.

This change in the grading policy had an interesting ef-

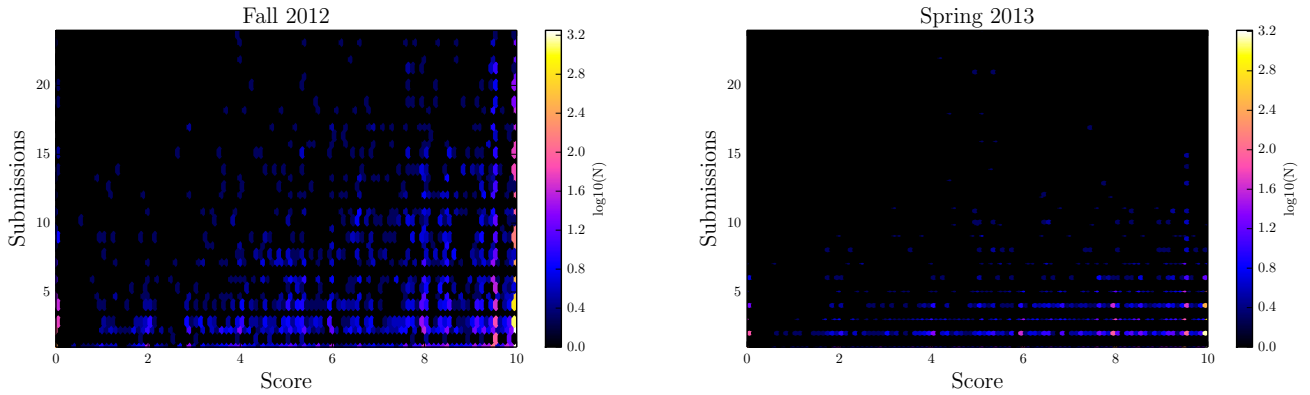

Figure 2: Heat maps correlating scores for the "Huffman" assignment and the number of required submissions.

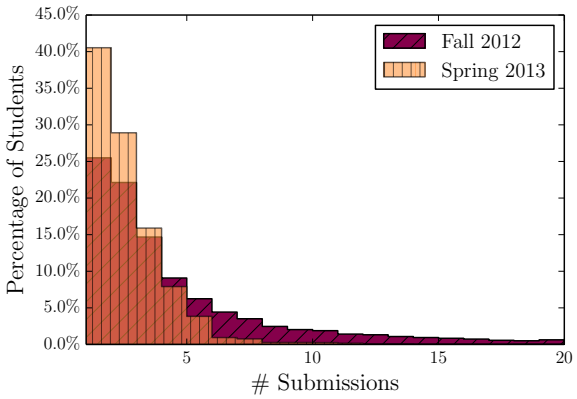

Figure 3: The number of submissions required to achieve a perfect score in the "Huffman" assignment.

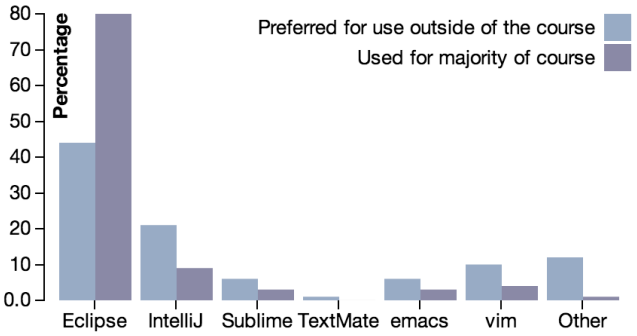

Figure 4: Results for the survey questions, what is your preferred editor, for use outside of this course?, and what editor did you end up using for the majority of the course?

fect on the quality of initial submissions. Figure 3 shows the change in the number of submissions of students who achieved a perfect score on a particular assignment. The change in submission behavior between the Fall 2012 and Spring 2013 offerings is quite noticeable. In the Fall 2012 iteration about 25% of those perfect-scorers needed only one submission, whereas in the Spring 2013 iteration this percentage grew to about 40%. For perfect-scorers who needed two submissions, the difference between the two iterations is not as large, but still significant (an increase of about 30%). This suggests that limitations on submissions should be an important consideration for designers of other MOOCs.

5.1.2 *The Scala IDE and Worksheet plugin*

Figure 4 shows survey respondents' code editor preferences both (a) preferred for use outside of the course, as well as (b) the code editor used for a majority of the course. The results show that 80% of all students prefer to use the Scala IDE for Eclipse for the course. For almost half of those students, Eclipse is not their preferred IDE for projects outside the course. One feature that was only available in the Scala IDE for Eclipse during the first course iteration is the Scala worksheet, introduced in Section 3. The students are encouraged to use the worksheet for testing their code, and the feature is also used by the lecturer in the videos. There are other reasons that explain the popularity of the Scala IDE: it is straightforward to download and install, the assignments can be easily imported into the IDE, and it is the recommended code editor for the course.

5.1.3 *Student Performance*

Figure 5 shows how students performed in both Fall 2012 and Spring 2013 offerings of the course. Students had access to an immediate feedback loop, giving them insight into how their code fared our automated test suite, including hints about which of our tests failed and cost them points, or which aspects of style needed to be improved (also a score deduction). This format resulted in a markedly different trend in how students seemed to navigate course material, how they seemed to learn, and certainly how they scored compared to a traditional course. Of particular interest is the strong tendency of students to continue improving their submission. Of all possible scores, the highest concentration of any one score (excluding scores of 0/0) for both iterations of the course was the perfect score of 80/80.

5.2 Not just for students

In this section, we evaluate the background and motivation of MOOC participants, in an effort to show that MOOCs can be useful to professional software engineers.

Figure 6 shows a summary of the educational background of participants of the MOOC. Surprisingly, nearly half of all respondents has completed a Master's degree. Thus, the question arises: how relevant are the topics of the MOOC to

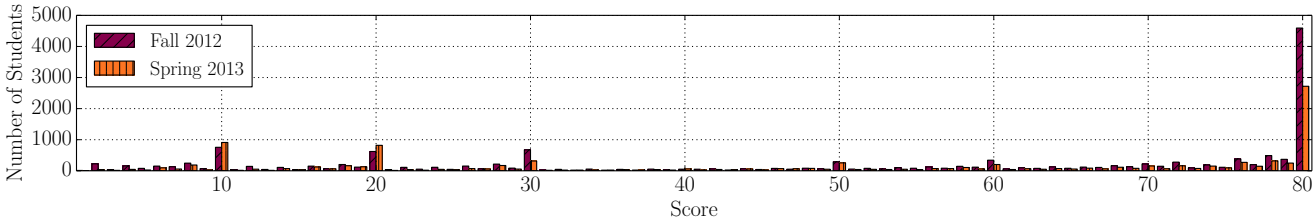

Figure 5: Final scores after the entire course (both iterations).

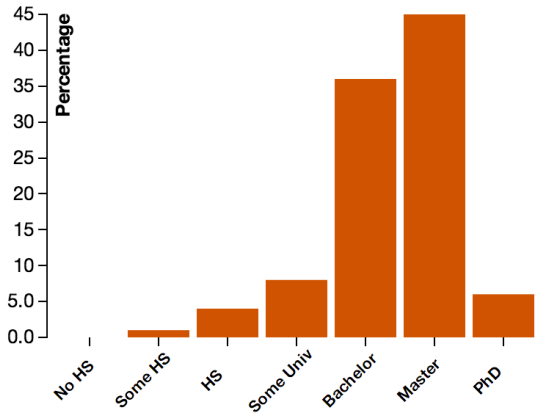

Figure 6: Educational background of MOOC participants

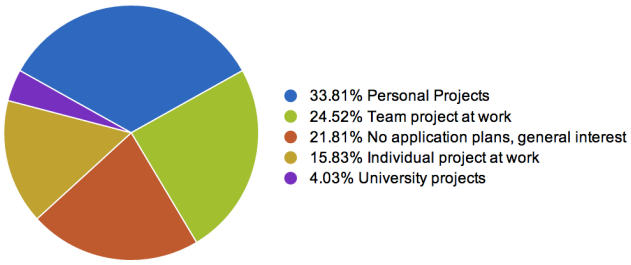

Figure 7: Results for the survey question: where do you plan to apply what you've learned in this course?

participants' professional work? Survey results indicate that about 40% of respondents plan to apply the learned knowledge at work (see Figure 7). Thus, participants' professional work appears to be a strong motivating factor. We suspect that the fact that the Scala programming language is not yet taught in many universities could also contribute to the popularity of the MOOC to (recent) university graduates.

Interestingly, professional participants felt like the course was well worth their time even more so than across all participants. Figure 8 shows the results of the question, "do you feel the course was worth the time you invested in it?" for two groups of individuals; "all respondents" as well as those respondents who indicated they were interested applying knowledge gained from the course towards a (group or individual) project at work. For the Fall 2012 offering, for example, the results show that 71% of professionals (i.e., 2,148/3,203 professional respondents) felt the course was well worth their time, as opposed to 68% (5,077/7,492) for all respondents. Thus, progfun was quite "effective" for practicing software engineers, indicating that some MOOCs can be an effective training tool for professional software engineers.

Another interesting observation that can be garnered from Figure 8 is that the difference in how "worth it" progfun was to professionals vs all respondents is large enough to be statistically significant, but still not disparate enough to indicate that potential newcomers (potentially 57%, or some 4,289/7,492 respondents) struggled or were hindered by the Scala tooling and build infrastructure. On the contrary, newcomers and Scala professionals alike followed suit in their positive overall experience with the course.

Furthermore, a certificate of completion was issued. Unlike languages established in industry for many years, there is no standard Scala certification for developers; the completion certificate could be regarded by many as the closest substitute.

5.3 Acceptance and student evaluation

Figure 9 shows the result of our survey among students at EPFL who took both the MOOC as well as the subsequent second part of a regular on-campus course.

The results show that the online part was very well received: in Figure 9(a) about 80% of all respondents think the online part was very good or excellent. In Figure 9(b), about 57% of respondents agree or strongly agree that they would like to have the opportunity to take more online courses.

In fact, 69% of students say they would prefer to have all 14 weeks of *a semester-long course like principles of functional programming* be online; only about 18% think that the configuration which was actually used, namely 7 weeks online, 7 weeks on-campus, was ideal. The results shown in Figure 9(e) show that EPFL students fully accepted the novel online part, and in fact preferred it compared to the traditional on-campus part.

Figure 9(c-d) also show that students seem to prefer the online help forums over EPFL's traditional on-campus help during weekly exercise sessions, with nearly 50% of all students rating the online forums as excellent or very good, as opposed to a mere 30% of students rating the traditional in-person exercise sessions as excellent or very good.

6. RELATED WORK

The well-known MOOC on software engineering organized by Fox and Patterson [5] has had around 50,000 registered students in its first iteration, and is thus comparable in size to our course; however, no survey of a comparable size was conducted among participants of the MOOC. Moreover, our selection of questions provides new insights, related to the interplay of MOOCs with professional software engineering,

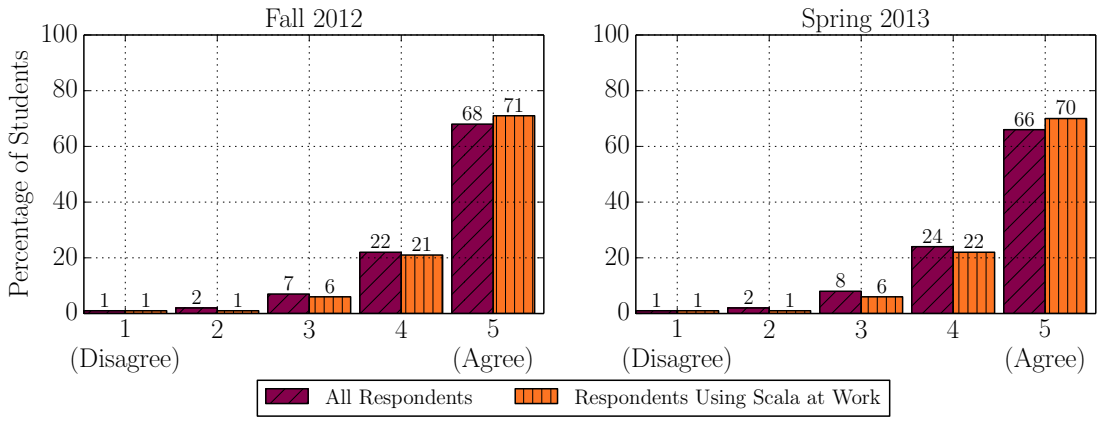

Figure 8: Correlated results from two groups of individuals for the survey question: do you feel the course was worth the time you invested in it? Shown are "all respondents", and respondents that indicated they were interested applying knowledge gained from the course towards a (group or individual) project at work.

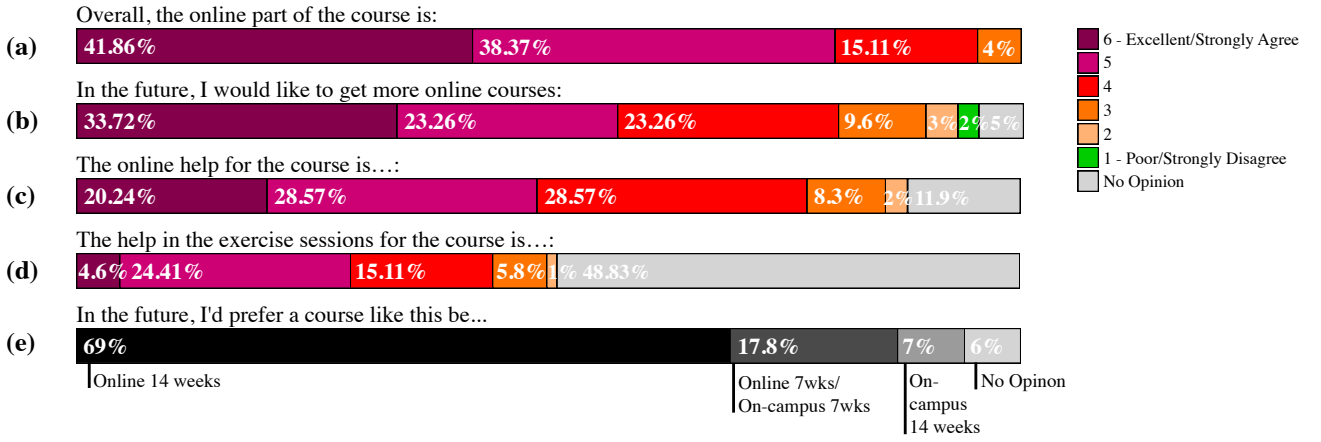

Figure 9: MOOC-related responses from EPFL's specialized course evaluation, given to on-campus EPFL students following the conclusion of the online MOOC half of the course, as well as the offline traditional half of the course.

in particular.

Vihavainen et al. report on a MOOC ("Helsinki MOOC") on introductory computer science with an emphasis on programming [19]. Compared to the Helsinki MOOC, our course had a number of registered students two orders of magnitude larger. Moreover, the Helsinki MOOC targets introductory programming, whereas our course targets advanced programming principles. As a result, our course was very popular especially among advanced developers who already have a Bachelor's or Master's degree. The Helsinki MOOC does not treat university students and MOOC participants equal with respect to the material used for exercises: the students in their university course are beta testers of the exercise material; thus, only after this beta test and necessary adjustments is the material released to non-local MOOC participants. Other organizational differences exist. For example, they give formal credits for "apprentices" who are unpaid "advisors" among fellow students with limited responsibilities. Their Extreme Apprenticeship (XA) learning methodology required a staff of around 20 persons associated with the course, with different roles and responsibilities. It is unclear whether the XA methodology would scale to a course of the scale of progfun (50,000 registered students versus 500 registered students).

Looking back into the literature a bit further, there also exists a body of work on automated grading of programming, beginning as early as the late 1960's, much of which is nicely summarized in a survey by Douce et al. [3]. These approaches are somewhat similar in that they are predominantly test-based. That is, even the very first automated testing systems introduced by Hext et al. [6] sought to compare some stored testing data with data from a running application. Some of these early systems, such as Assyst [7], even developed methodologies for testing for qualities such as efficiency and modularity, though naturally, none of these systems are able to be run concurrently in the cloud. Similar, however, is RoboProf [2], a web-based system for testing code submissions and providing feedback, though not suited

to scale as necessary for use with MOOCs. Meanwhile, more recent work [16] differs in that its focus is not test-based, but rather, based on program synthesis âĂŞ it's additionally not suited to our use-case because it provides too much assistance.

Our pedagogical approach is reminiscent of "autonomous learning" [15], focusing on creating rich learning environments rather than on passive transmission of information. Moreover, our MOOC format shares the goal of combining the virtues of campus-based education and distance education with other approaches to interactive distance education [20]. However, our on-campus course offering is still markedly different from offerings of open and distance learning (ODL) institutions (*e.g.,* [10]): it was an explicit non-goal to scale our on-campus course offering to a large audience of distance learners; instead, even though the on-campus course format shared learning resources with the MOOC course format, the two course formats served different purposes and were consequently kept distinct. Changes to the EPFL course had the sole aim of improving the learning experience without any compromises. As our evaluations show, EPFL students preferred the newly introduced resources for asynchronous learning.

7. CONCLUSION

We have presented a detailed experience report which covers the organization, format, and infrastructure, of our popular MOOC, *Functional Programming Principles in Scala*. We've also presented and analyzed a selection of the data we collected throughout each offering of the course, providing insight from several different perspectives; characteristics and motivations of the tens of thousands of students worldwide who have participated in the course, the experiences of the on-campus EPFL students, as well as objective anonymized data about the behavior and performance of students as seen from the perspective of the instructor.

In our evaluation of the data, we showed that providing innovative course-supporting tools (IDE plugins, testing frameworks, interactive build tools, automated cloud-based graders, style checkers) and focusing on human-computer interaction issues, such as tight feedback loops, not only facilitates distance learning, but improves the learning experience even for on-campus students. Surprisingly, from experience with our own undergraduate students, we found that our MOOC format was even preferred over traditional on-campus course format.

Finally, the results of our evaluation show that SE-related MOOCs can be very useful not just for students, but also for practicing professional software engineers.

8. REFERENCES

[1] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Mass., 1985.

[2] C. Daly. Roboprof and an introductory computer programming course. In *ITiCSE*, pages 155–158. ACM, 1999.

[3] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: a review. *Journal on Educational Resources in Computing (JERIC)*, 5(3), Sept. 2005.

[4] M. Farwell. Scalastyle project. `http://www.scalastyle.org/`.

[5] A. Fox and D. A. Patterson. Crossing the software education chasm. *Commun. ACM*, 55(5):44–49, 2012.

[6] J. B. Hext and J. W. Winings. An automatic grading scheme for simple programming exercises. *Commun. ACM*, 12(5):272–275, 1969.

[7] D. Jackson and M. Usher. Grading student programs using ASSYST. In *SIGCSE*, pages 335–339. ACM, 1997.

[8] K. Jordan. MOOC completion rates: The data. `http://www.katyjordan.com/MOOCproject.html`.

[9] LAMP/EPFL. progfun-stats project. `https://github.com/heathermiller/progfun-stats/`.

[10] R. Mills and A. Tait. *The convergence of distance and conventional education: Patterns of flexibility for the individual learner*. Routledge, 2002.

[11] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. Artima Press, Mountain View, CA, 2007.

[12] K. O'Hair. How VM agents work. `https://blogs.oracle.com/kto/entry/using_vm_agents`.

[13] Oracle. Java tutorials: The security manager. `http://docs.oracle.com/javase/tutorial/essential/environment/security.html`.

[14] C. Parr. Not staying the course. `http://www.insidehighered.com/news/2013/05/10/new-study-low-mooc-completion-rates`.

[15] O. Peters. Digital learning environments: New possibilities and opportunities. *The International Review of Research in Open and Distance Learning*, 1(1), 2000.

[16] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *PLDI*, pages 15–26. ACM, 2013.

[17] Typesafe. sbt. `http://www.scala-sbt.org/`.

[18] B. Venners. ScalaTest. `http://www.scalatest.org/`.

[19] A. Vihavainen, M. Luukkainen, and J. Kurhila. Multi-faceted support for MOOC in programming. In *SIGITE*, pages 171–176. ACM, 2012.

[20] G. L. Waddoups and S. L. Howell. Bringing online learning to campus: The hybridization of teaching and learning at brigham young university. *The International Review of Research in Open and Distance Learning*, 2(2), 2002.

[21] M. S. Ware and C. J. Fox. Securing Java code: heuristics and an evaluation of static analysis tools. In *Proc. Static Analysis Workshop (SAW)*, Jun 2008.