Goto Statement Considered Harmful – A Paper Review

Author: Maryna Babayeva
Student-ID: F100185
maryna.babayeva@gmail.com

Course: Software Engineering
Department of Information and Computing Sciences,
Utrecht University

Lecturer: Jurriaan Hage
November 11, 2011

Abstract

In 1968 Edsger Dijkstra wrote a letter to the editor of *Communications of the ACM* (CACM) stating his opinion about using the **goto** statement in programs. This letter would serve to begin a hot-headed debate within the programming community. This topic, however, would eventually stop being the subject of discussion as computer scientists shifted their attention to the structured programming paradigm, which forbids the use of the **goto** statement. This is especially manifested in the fact that some high level programming languages don't even include the **goto** statement as part of their syntax. This literature review aims to expose the different perspectives on the use of **goto**. Although many computer scientists believe that **goto** can be useful in increasing the efficiency of a program, few are really convinced that the enhanced performance is worth the extra complexity. Here Dijkstra's letter will be reviewed and further opinions will be discussed to give a comprehensive view on the current state of this debate.

1 Introduction

In his famous letter "Go To Statement Considered Harmful" [1] Dijkstra argued that it should always be possible to predict a value of a variable based on the program's coordinate system. However, when a **goto** is used it becomes quite difficult to keep track of the program flow. A remedy to this problem, according to Dijkstra, would be to substitute the use of **goto**s with conditional and branch statements. This was justified on the basis that the latter kind of statements conserves the position of variables in the program's coordinate system. Take as example a sequential program, where the coordinate system is denoted by the the line number, adding loops would introduce a loop counter but the possibility of predicting the variable values would be preserved. Building on that, Dijkstra concluded that a proper program construct must not destroy such a coordinate system for the programmer, and thus the **goto** construct must fall out of favor. Dijkstra's letter sparked a revolution in programming methods. It gave rise to terms as "**goto**-less programming" and significantly influenced trends as "structured programming". And now, as we move on in

time, programmers and computer science students are becoming more unfamiliar with the **goto** construct, as it is neither emphasized in the teaching curricula nor is it taught as a good programming practice at work. There remains, however, computer scientists who are sceptical about the harm of a **goto** and regard it rather as a benefit to languages that offer the possibility of using it.

2 Discussion

2.1 Pro Goto

As a response to Dijkstra's letter, Frank Rubin likewise wrote to CACM almost 20 years later [2]. He argued that **goto**-less programming has cost businesses "hundreds of millions of dollars." He also provided a code fragment explicitly showing the extent to which a **goto** statement could simplify a program. Rubin thus raised the complaint that the belief that **goto** statements are harmful had become more of a religious doctrine without any proven evidence. A quote by Frank Rubin about the **goto** statement emphasizes his point of view:

> *'It is like butchers banning knives because workers sometimes cut themselves.'*

Rubin thus conjectured that programs using **goto** are less complex, with fewer lines of code and operations, than **goto**-less programs. Furthermore, he mentioned his observations that programmers needed less time to come up with a solution when allowed to use the **goto**.

The opinion of **goto** statement being evil appeared when Fortran was the most popular language. Since Fortran did not provide loop structures yet, programmers tended to produce unstructured spaghetti code. But according to Donald Knuth's article from 1974 [3] where he presents the pros and cons of the **goto**, using it can result in faster and smaller code in some cases. By providing a collection of programming examples with a **goto** statement, Knuth explains that a **goto** statement does not differ from variables or other identifiers when given a meaningful name.

Kondoh and Futatsugi wrote a similar article in 2006 [4], where they assert in contradiction to Dijkstra that the **goto** programming styles are more suitable for proving correctness in Hoare Logic than the **goto**-less ones. In their article they agree with Rubin that removing the **goto** introduced new variables to the program and made it more complicated.

2.2 Contra Goto

Figure 1: An illustration by Leo Brodie from his book about the 'Thinking Forth' programming language [5].

As we have seen, the most common claim against the use of **goto**s is that code containing them tends to deteriorate into low-quality code. Dijkstra even ventured to say that the quality of a program is inversely proportional to the number of **goto**s used. Another claim is that it makes programs' behavior hard to predict as was illustrated in a humorous way by Leo Brodie in Figure 1.

In their article [6], Ryder *et al.* approach the debate about the use of **goto**s by interviewing programming language designers. Jean Ichbiah, the principal designer of Ada,

states in this interview that when developing a program the key emphasis lies in making it readable. According to him, readability is the most important issue, as it is the driving factor behind both the reliability and the future maintainability of a program.

Another side of the argument comes from Wulf, who presents a series of arguments against the use of **goto**s [7]. Wulf states that **goto**s obscure the logical structure of a program and make it difficult to understand and modify it. He mentions that although not all uses of the **goto** are to be considered harmful, the "good" uses apply only to a small number of specific cases and may be handled by other constructs in favor of readability.

Another aspect mentioned by Wulf is compiler optimization. Flow analysis of programs including **goto** statements are hard. Thus, since flow analysis is necessary for global optimization, it must be taken into consideration as well.

3 Conclusion

Edsger Dijkstra's opinion on the use of **goto** statements contributed to the way of how programs are designed today. Now it seems to be common understanding that a **goto** is a confusing language construct and most programmers lack the knowledge of its proper use. Extensive use of the **goto** statement might produce unreadable and complicated code, which in addition to that, is hard to maintain and modify. In today's applications, readability of a program plays a major role in software development, especially in large projects with thousands lines of code and big teams of programmers. I can thus conclude in favor of structural programming as it satisfies the above mentioned properties. Today's highest quality codes are based heavily on structural programming paradigms without making use of the **goto** statement. And that clearly rests the case in favor of **goto**-less programming.

4 Acknowledgements

I would like to thank Sebastian Waszak for proofreading.

References

[1] Dijkstra, Edsger W.: Letters to the editor: Go To Statement Considered Harmful. In: *Commun. ACM* 11 (1968), Nr. 3, 147–148. http://doi.acm.org/10.1145/362929.362947. – ISSN 0001–0782

[2] Rubin, Frank: Letters to the editor: "Go To Considered Harmful" Considered Harmful. In: *Communications of the ACM* 30 (1987), March, Nr. 3, 195–196. http://www.ecn.purdue.edu/ParaMount/papers/rubin87goto.pdf

[3] Knuth, D.: Structured programming with go to statements. In: *Computing Surveys* Vol. 6 (1974), December, Nr. Nr. 4. http://portal.acm.org/citation.cfm?id=1241535

[4] Kondoh, Hidetaka ; Futatsugi, Kokichi: To use or not to use the goto statement: Programming styles viewed from Hoare Logic. In: *Science of Computer Programming* 60 (2006), Nr. 1, 82 - 116. http://www.sciencedirect.com/science/article/B6V17-4H0S16M-1/2/be4caefa0dcbe939bf7a087e6f987bef. – ISSN 0167–6423

[5] Brodie, Leo: *Thinking Forth.* Creative Commons, 2004 http://netcologne.dl.sourceforge.net/project/thinking-forth/reprint/rel-1.0/thinking-forth.pdf. – ISBN 0–9764587–0–5

[6] Ryder, Barbara G. ; Soffa, Mary L. ; Burnett, Margaret: The impact of software engineering research on modern progamming languages. In: *ACM Trans. Softw. Eng. Methodol.* 14 (2005), Nr. 4, 431–477. http://portal.acm.

org/citation.cfm?id=1101818. – ISSN 1049–331X

[7] Wulf, William A.: A case against the GOTO. In: *ACM '72: Proceedings of the ACM annual conference* (1972), 791–797. http://portal.acm.org/citation.cfm?id=805861