How 'Go to statement considered harmful' influenced computing.

COMP110 - Research Journal

1706966

November 17, 2017

1 Introduction

This paper will be looking at the historical context and influences made by the paper 'Go to statement considered harmful' [1]. It is important to note that it was originally penned in 1968, and the author of this research journal has minimal knowledge of how computing was during that time, or even in the present time period, at the start of this paper. However with the use of other research articles, a clear insight will be shown into how [1] has changed the computing world at its inception and the years following leading into the modern day.

2 Summary of chosen paper

The instant insight that is be taken from [1] is that Dijkstra's opinion on the use of go to statements in code base directly relates to the skill of the programmer. Such that if more go to statements are used in a piece of code, the lower the quality of the programmer writing it. This clearly shows Dijkstra's disapproving thoughts of the go to statement

and he goes on to mention possible better ways to code your programs.
Ways mentioned are conditional clauses (if statements) and repetition clauses (while statements). Though it is mentioned that the later is unnecessary 'because we can express repetition with the aid of recursive procedures' [1, p.147] the topic is not touched on more and just continues with repetition clauses. These clauses mentioned are widely used throughout programming currently. However from how it is stated in [1] it comes across that, although these were available for programmers to use at the time, they may not have been as commonly used then as they are today.
Dijkstra goes on to loop back around to talk about the go to statement and how 'unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress' [1, p.147]. I believe this extract implies that if the go to statement is used without proper consideration it can lead to difficulty when trying to interpret how the program is running, and where it is in the process at any giving time. This is confirmed in a more recent paper 'if a program is halted, go tos make it difficult to find a place in the programming that corresponds to the halted process' [2, p.37].
Towards the very end of the article Dijkstra mentions that in [3] the go to statement has been proved to be unnecessary using a jump-less flow diagram. He goes on to say how this is not recommend as 'the resulting flow diagram cannot be expected to be more transparent than the original one' [1, p.148]. This points back to the article title that the go to statement should be considered harmful for programs, but not entirely unusable as they can lend themselves to improving program readability. Another paper takes this view as well and argues 'for the elimination of go to's in certain cases, and for their introduction in others.' [4, p.262] using Dijkstra's paper as heavy influence.

3 Early Influence

Many papers have gone on to cite [1] over the years, using it to confirm and give a basis for some of their claims. The use of this paper alone give credence to it's impact on how code is written. The sections on influence will look at how extracts have been used in other papers, and merge similar concepts, giving a better look at the specific parts that have moulded the way code is used today.

The book Object-oriented software construction says 'The field of software development methodology is not new. Its origins may be traced to Dijkstras famous Go To Statement Considered Harmful' [5, p.664]. This example shows that Dijkstra's article was pioneering in the ways of showing how to and not to write code, a catalyst for the extensive research and structure guides in the modern day. Also said 'The next chapter in the story is what many people regard as the first' [4, p.265] backing up the idea that Dijkstra led the way in starting the never ending discussion on structure.

Knuth quotes Dijkstra from a personal communication as saying 'Please don't fall into the trap of believing that I am terribly dogmatical about [the go to statement]. I have the uncomfortable feeling that others are making a religion out of it, as if the conceptual problems of programming could be solved by a single trick, by a simple form of coding discipline!' [4, p.265]. It's interesting to read this as it shows how strongly Dijkstra's paper had influenced people at the time. Some taking the concept further than had been intended, and even forming such strong views that its being refereed to as religious.

After a while programming languages that didn't include go to statements started appearing, one of the most used being called Bliss [6], using instead statements it called escape. This apparently wasn't enough and they went on to add another statement called leave, taking the program to a different place after a certain argument [7]. It appears that completely removing go to statements, although highly advocated by some, is a tricky process not always lending itself to the most read able and time efficient code.

On a side note an interesting snippet of information found was that ‘In order to speed publication, the editor decided to publish Dijkstra’s article as a letter’ [4, p.265]. This probably circumvented rigorous peer reviewing and begs the question why the editor pushed through the article?

4 Modern Day Influence

Even in recent years Dijkstra’s paper is still called upon through out academic work, one such example is to show base structured programming constructs in [8] where they use the examples of conditional and repetition clauses
Wendy Hui Kyong Chun references Dijkstra’s work over multiple papers, in two of them the same part is used to convey similar messages. ’shorten the conceptual gap between static program and dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.’ [1, p.137] is used in both [2] and [9] to pull out the idea that ’go tos make it difficult for the source program to act as a legible source.’ [9, p.303].
This matter gets expanded on leading into the idea that code is not purely about the end product but that the source itself is what truly matters. The argument goes on to say that debugging code is not the only reason for this but also critical analyses. The author of this journal believes it could be taken deeper than this, with the readability of code being of utmost importance in the age of open source software and group collaboration, it becomes more and more imperative that programmers strive for this trait in their work.
A paper [10] focusing on teaching programming to beginners uses Dijkstra’s paper to argue that because go to statements have been marked as harmful ’why should the students learn a concept which is known to have downsides.’ [10, p.94]. Although it is understandable where they are coming from, it should be noted that it wasn’t concluded

that go to statements are not useful, just harmful when over used or used where another clause would have been better. Go to statements should be taught but done correctly and not as the main base of a language.

5 End Summary

All aspects of this paper (early and modern) go to show how truly influential Dijkstra's work was. Pushing forward new ideas which inadvertently led into a war between go to defenders, the anti go to crusaders and then those who really understood the power in balance. Though the real take away was the discussions and research pushed on by these causes, leading to a much better grasp for all on best practice concerning code structure.

References

[1] E. W. Dijkstra, "Go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147–148, 1968.

[2] W. H. K. Chun, "On software, or the persistence of visual knowledge," *grey room*, no. 18, pp. 26–51, 2005.

[3] C. Böhm and G. Jacopini, "Flow diagrams, turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366–371, 1966.

[4] D. E. Knuth, "Structured programming with go to statements," *ACM Computing Surveys (CSUR)*, vol. 6, no. 4, pp. 261–301, 1974.

[5] B. Meyer, *Object-oriented software construction.* Prentice hall New York, 1988, vol. 2.

[6] W. A. Wulf, D. Russell, and A. N. Habermann, “Bliss: a language for systems programming,” *Communications of the ACM*, vol. 14, no. 12, pp. 780–790, 1971.

[7] W. A. Wulf, “A case against the goto,” in *Proceedings of the ACM annual conference-Volume 2*. ACM, 1972, pp. 791–797.

[8] M. Neil, N. Fenton, and L. Nielson, “Building large-scale bayesian networks,” *The Knowledge Engineering Review*, vol. 15, no. 3, pp. 257–284, 2000.

[9] W. H. K. Chun, “On” sourcery,” or code as fetish,” *Configurations*, vol. 16, no. 3, pp. 299–324, 2008.

[10] F. Huch, “Learning programming with erlang,” in *Proceedings of the 2007 SIGPLAN workshop on ERLANG Workshop*. ACM, 2007, pp. 93–99.