The Legacy of Dijkstra's Paper, GoTo Considered Harmful

Thomas Chambers, Falmouth University

November 12, 2021

I. Introduction

Edsger W. Dijkstra is arguably the most influential programmer to have contributed to the field of computer science and software engineering. As a proud pioneer of programming, declaring it as a profession before the Dutch authorities even legally accepted such a claim[5], he was known for his elegant, colourful prose and strong opinions on the field [1].

In the March of 1968, in the *Communications of the ACM*, Dijkstra submitted a short paper with the iconic name, *Go To Statement Considered Harmful*. Originally called *A case against the GoTo statement*[6], this paper became one of Dijkstra's most famous pieces of writing and, as Dijkstra put it, *"a cornerstone"* of his fame[8].

Here, I will outline the state of computing that led to the paper, the controversy of Dijkstra's writing, and the influence.

II. The State of Programming and Programming Attitudes up to the time of the Paper

i. Programming Languages

In the '50s and '60s, the vast breadth of programming terminology and knowledge we have today was currently in development. In these early days, computers were programmed in primitive assembly languages or machine language. This led to a lot of the programmers attention being drawn toward working around the limitations of the time, such as a lack of index registers or primitive input-output arrangements[4]. However, what was starting to be developed were languages that utilized *compilers*. These allowed a programmer to write in a higher-level language, meaning faster writing times and less of a concern on the system hardware (this was still a considerable factor however).

Higher-level languages, such as Fortran and Algol 60 (a language Dijkstra helped develop[1]), were some of the first to use advanced compilers. Fortran at the time quickly became the *go-to* programming language for scientific computation[4].Though many years later it would be described by Dijkstra as "*hopelessly inadequate*"[7], it was the best he or anybody had at the time.

ii. GoTo Statement's and their Use

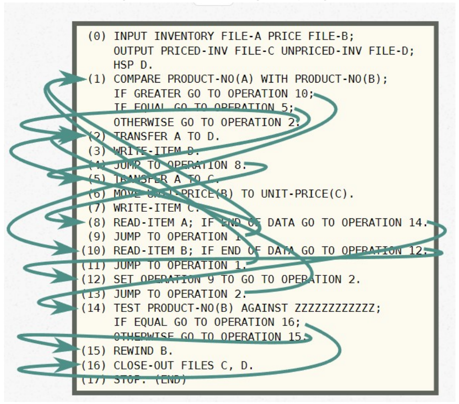
A *GoTo* statement (often called a *Jump To*)is used to take the flow of a program from one point to another, that new point being marked by a *label* and being located anywhere in the program. The GoTo statement made its way into high-level languages thanks to the fact that they evolved out of assembly, where GoTos are essential for the workings of the program. It was standard practice to write programs with a non-sequential control flow, meaning the program *flows* up and down the code as its executed. The use of GoTo was showing no signs of slowing down, with higher-level languages expanding their use.

- Fortran introduced "computed GoTo", allowing the ability to jump to many labels rather than just one.[14]
- FLOW-MATIC, which used more English like terms, differentiated between GoTo and *jump to*, with *jump to* appearing at

the beginning of statements and GoTo in conditionals.[15]

GoTos in higher level languages led to what's now known as *Spaghetti Code*. A consequence of non-sequential control flow, leading to a very messy trace of the program.

Figure 1: *Traced Jump Statements in a FLOW-MATIC program, showing "Spaghetti Code"*



```
(0) INPUT INVENTORY FILE-A PRICE FILE-B;
    OUTPUT PRICED-INV FILE-C UNPRICED-INV FILE-D;
    HSP D.
(1) COMPARE PRODUCT-NO(A) WITH PRODUCT-NO(B);
    IF GREATER GO TO OPERATION 10;
    IF EQUAL GO TO OPERATION 5;
    OTHERWISE GO TO OPERATION 2.
(2) TRANSFER A TO D.
(3) WRITE-ITEM D.
(4) JUMP TO OPERATION 8.
(5) TRANSFER A TO C.
(6) MOVE UNIT-PRICE(B) TO UNIT-PRICE(C).
(7) WRITE-ITEM C.
(8) READ-ITEM A; IF END OF DATA GO TO OPERATION 14.
(9) JUMP TO OPERATION 1.
(10) READ-ITEM B; IF END OF DATA GO TO OPERATION 12.
(11) JUMP TO OPERATION 1.
(12) SET OPERATION 9 TO GO TO OPERATION 2.
(13) JUMP TO OPERATION 2.
(14) TEST PRODUCT-NO(B) AGAINST ZZZZZZZZZZZZ;
    IF EQUAL GO TO OPERATION 16;
    OTHERWISE GO TO OPERATION 15.
(15) REWIND B.
(16) CLOSE-OUT FILES C, D.
(17) STOP. (END)
```

(https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/)

iii. Dijkstra's Motivation for the paper

Dijkstra noticed that more conceptual thought was needed in the design of the languages and the way programmers wrote them. He was a programmer with a heavy mathematical background, concerned with the *correctness* of a program. He wanted to be able to show how we could analyze a program and the way it was written to verify that it would run *correctly*, and to be able to apply a mathematical proof to the implementation[8].

III. The Contents of the Paper, "Go To Considered Harmful"

Dijkstra opens with his concerns about the state of programmers' quality. Making the argument that he is "convinced that the go to statement should be abolished from all "higher level" programming languages". *Higher-level* being in quotations as the term was still in development, so much so that Dijkstra goes on to explain the meaning[6].

Dijkstra remarks that the "true matter" of a programmer's job is not the construction of the program, but the dynamic process when it executes. Being "aware of our limitations" Dijkstra argues, we should focus on static relations. Dijkstra believed that programmers are "geared" towards understanding.[6].

He explains the importance of understanding relations between the program and its source. With the concept of the program's *independent coordinates*. These are generated by the dynamic evolution of the process. The argued importance of these coordinates is that "we can interpret the value of a variable only with respect to the progress of the process". Dijkstra explains likes so; Take a simple program written to count the amount of n people entering a room. As someone enters, n is incremented by one. A program written with a sequential control flow, at the point between when a person has entered the room, but, n is about to be incremented, you **know** the value of n is equal to the amount of people in the room *minus* one. Dijkstra explains that how in a program with a non-sequential control flow any coordinate system you use to find those *n equals the amount of people in the room minus one* moments are "**extremely** complicated" and "utterly unhelpful"[6].

In the last part of the paper, Dijkstra explains the heart of his argument; That the "*primitive*" GoTo is "too much an invitation to make a mess of one's program"[6]. Use should satisfy the requirement of the program's coordinate system to the extent it can be meaningfully described in a helpful, and maintainable way. Throughout the paper, Dijkstra has clearly conveyed that he believes the use of GoTo's, in all but machine code, ruin the benefits of structured control flow.

IV. Sparked Controversy and Debate

Dijkstra's paper ignited the debate around GoTo. While Dijkstra wasn't the first warn of GoTo's use [21], his name is now at the start of any conversation about them. Debates on GoTo's following the release of the paper would even open with a reference to Dijkstra, such as the "GOTO controversy" debate at the ACM[13], and forum debates in *Communications of the ACM* [2]. Many papers would be written in response, perhaps most famously, the paper titled *"GoTO Considered Harmful" Considered Harmful* by Frank Rubin[3].

The influence of Dijkstra's paper is clearly on show by Rubin here, where they called Dijkstra's paper "The most-noted item ever published in the *Communication* [of the ACM]". Rubin claims the paper has become "fixed in the mind of every programming manager". Rubin's argument is that while GoTo's are often used to write "some awful programs", to get rid of GoTo statements completely would be a significant lost to the programming.

The reasoning is shown better in the famous response to Dijkstra by Donald E. Knuth, in their 1974 paper, *Structured Programming with go to Statements*[10]. Again the influence can be shown, with Knuth naming Dijkstra as the cause for the "revolution". He also notes conversations with Dijkstra. He writes that Dijkstra was conceding that there are needed and good uses of GoTo. In fact, in a conversation Knuth had with Dijkstra, Dijkstra wrote to Knuth

> *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*

[10] This helps to show Dijkstra's deeper thinking and concern about programming practices, past the use of GoTo. In his later paper, *Notes on structured programming*[8], he didn't even mention GoTo's, but instead focused on structured programming and abstraction.

Knuth finds a middle ground between the arguments and makes the point that "The real goal is to formulate our programs in such a way that they are easily understood" and that we "shouldn't merely remove go to statements because it's the fashionable thing to do".

Knuth and Dijkstra, albeit not as far as Knuth, agree that it's the "conceptual formulation" of the code and "good program structure"[10] that really matter.

V. State of Modern Programming Today and Dijkstra's Influence

Go To Considered Harmful sparked a worldwide debate, where Dijkstra's point eventually came out on top. Still, new debate is being opened, studies are being conducted and papers written [19][17][11]. Most modern programming books will advise against - or simply not accept - the use of GoTos[16][9], with such things to say as "When should a GoTo be used? The answer is not at all"[18].

Its clear that the development of languages was heavily influenced by Dijkstra's paper, with a survey of 1000 college professors for the greatest papers in computing included *Considered Harmful*[12] suggesting the influence is being passed on to new students.

The effect on language can not be understated, in an interview[20], Dr.Jean Ichbiah, principal designer of Ada, referenced Considered Harmful, describing how it was "laying the foundation of how to understand a program". Ichbiah goes on to describe its influence in designing Ada.

VI. Conclusion

Most languages today, apart from some such as Java and Python, have some use of GoTo statements. However, they're a completely different *GoTo* to the one Dijkstra was describing; These GoTo's implementation is very strict, never allowing you to jump in or out of functions or being best abstracted as *return* or *break* statements[18]. It's clear that the core ideas behind Dijkstra's writings on GoTo have been one of the largest influences on the development of computing.

References

[1] Krzysztof Apt. The man who carried computer science on his shoulders. *Inference: International Review of Science*, 5(3), Sep 2020.

[2] Robert L. Ashenhurst. Acm forum. *Commun. ACM*, 30(8):658–662, August 1987.

[3] Robert L. Ashenhurst. Acm forum. *Commun. ACM*, 30(3):195–196, March 1987.

[4] John Backus. The history of fortran i, ii, and iii. *History of programming languages*, page 25–74, 1979.

[5] P.p. Chen. From goto-less to structured programming: the legacy of edger w. dijkstra. *IEEE Software*, 19(5):21–21, 2002.

[6] Edsger W. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.

[7] Edsger W. Dijkstra. How do we tell truths that might hurt? *Selected Writings on Computing: A personal Perspective*, page 129–131, 1982.

[8] Edsger W. Dijkstra. Ewd1308: What led to "notes on structured programming". circulated privately, June 2001.

[9] K. N. King. *C Programming: A Modern Approach*, pages 113–114. W.W.Norton Company, Inc, 500 Fith Avenue, New York, NY 10110, second edition, 2008.

[10] Donald E. Knuth. Structured programming with <i>go to</i> statements. *ACM Comput. Surv.*, 6(4):261–301, December 1974.

[11] Hidetaka Kondoh and Kokichi Futatsugi. To use or not to use the goto statement: Programming styles viewed from hoare logic. *Science of Computer Programming*, 60(1):82–116, 2006.

[12] Phillip Laplante. Great papers in computer science, 1996.

[13] Burt M. Leavenworth. Control structures in programming languages - part ii (panel session). *Proceedings of the ACM annual conference on - ACM 72*, Aug 1972.

[14] Andru Luvisi. The history, controversy, and evolution of the goto statement, 2008. Sonoma State University Handout.

[15] Mark C. Marino. *Critical code studies*, page 139. The MIT Press, 2020.

[16] Robert C. Martin. *Clean Code: A handbook of Agile Software Craftsmanship*, page 49. Prentice Hall PTR, Upper Saddle River, first edition, 2009.

[17] Meiyappan Nagappan, Romain Robbes, Yasutaka Kamei, Éric Tanter, Shane McIntosh, Audris Mockus, and Ahmed E. Hassan. An empirical study of goto in c code from github repositories. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 404–414, New York, NY, USA, 2015. Association for Computing Machinery.

[18] Al Kelly Ira Pohl. *A Book On C*, pages 178–179. Addison-Wesley, 201 W. 103rd Street, forth edition, 1998.

[19] Eric S. Roberts. Loop exits and structured programming: Reopening the debate. In *Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '95, page 268–272, New York, NY, USA, 1995. Association for Computing Machinery.

[20] Barbara G. Ryder, Mary Lou Soffa, and Margaret Burnett. The impact of software engineering research on modern programming languages. *ACM Trans. Softw. Eng. Methodol.*, 14(4):431–477, October 2005.

[21] Niklaus Wirth and C. A. R. Hoare. A contribution to the development of algol. *Communications of the ACM*, 9(6):413–432, 1966.