Editor: **Tim Menzies**
North Carolina State University
tim@menzies.us

Reconsidering Whether GOTO Is Harmful

Meiyappan Nagappan

From the Editor

It's high time to recheck the truisms of software engineering because many of them are poorly established and not so wise. For example, consider GOTO (the ultimate four-letter word in software engineering):

"In 1978, Berkeley scientists did a study on the effects of GOTO on rats. Twenty rats in a control group were fed a normal diet, while twenty other rats were forced to program in Apple BASIC. Their health was to be monitored for a month, but after two weeks all of the rats in the programming group had melted into a dense goo that tasted a little like quiche.

"But some rats learned to execute GOTO END_OF_MAZE, and finished in record time, while the nested block rats took too long to climb into and out of nested boxes to get to the end.

"Seriously, though, don't use GOTO statements. Like any four letter words, they are not appropriate for polite company." (wiki.c2.com/?GotoConsideredHarmful).

But as Mei Nagappan and colleagues report here, perhaps it's high time to readmit GOTO into polite society.

Do you have any other surprising result or industrial experience? Something that challenges decades of conventional thinking in software engineering? If so, please email a one-paragraph synopsis to tim@menzies.us (use the subject line "REDIRECTIONS: Idea: [your idea]"). If that looks interesting, I'll ask you to submit a 1,000- to 2,400-word article (where each graph, table, or figure is worth 250 words) for review to *IEEE Software*. *—Tim Menzies*

THIS MARCH WILL be 50 years since the publication of "Go To Statement Considered Harmful" by Edsger Dijkstra in *Communications of the ACM*.[1] This is one of his more popular papers among practitioners[2] and researchers (more than 1,700 citations in Google Scholar). This in turn has had a considerable impact on what educators teach their students. Anecdotally, several introductory courses in CS programs teach students to not use GOTOs.

Dijkstra's Argument against GOTO

Dijkstra presented his well-reasoned thoughts on GOTO statements and the problems they can cause. The article was motivated by his desire to make programs verifiable. He stated that one of the main goals of a software developer is to know the state of the executing program (and any developer would confirm this statement). He then observed that in an executing program, at any given time instant, the use of GOTO can make it difficult to determine the exact set of statements that have executed so far. He stated, "The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program." This is because multiple GOTO statements can jump to the same label that is sequentially before the GOTO statement in a piece of code.

For example, in a procedural programming language such as C, GOTO statements could be in lines 4, 6, and 12 of a method, and the label that the GOTO statements jump to could be in line 2. (Such a use of GOTO simulates a loop, but of variable length.) So, when the program crashes and we know we're at line 2, we don't know how much of the program has actually executed (it could be up to line 2, 4, 6, or 12).

An Empirical Look at GOTO

Although several blogs, forums, and Q&A websites have discussed the merit and issues of Dijkstra's papers, none have presented any empirical evidence about using GOTO. So, a team of researchers from across the world came together to examine the use of GOTO empirically.[3] We looked at 11,627 C projects from the popular social-coding website GitHub.

We asked three questions:

- How prevalent was the use of GOTO?
- How did the developers use GOTO?
- Did the developers modify or remove any GOTO statements?

To determine the answers, we analyzed the projects both quantitatively and qualitatively.

How Prevalent Was GOTO?

Of the 11,627 projects, 3,093 used at least one GOTO statement. Almost 11.5 percent of all the files in the projects had a GOTO statement. However, approximately 14 percent of the files had only one GOTO statement. Almost half of the projects that had a GOTO statement had one in at least 20 percent of the files. So, we concluded that the amount of GOTO use was nontrivial.

How Did Developers Use GOTO?

To understand how the developers used GOTO, we looked at a statistically valid sample (95 percent confidence with a 5 percent error margin) of files with GOTO statements. We examined it at two levels of granularity: the file level and the function level.

Almost 85 percent of the files were system drivers or networking files. Only about 1.5 percent of the GOTO statements were in machine-generated code. So, we had more evidence that the developers used GOTO statements, but primarily for system drivers and networking code.

At the function level, we found that the programmers used GOTO statements primarily for

- error and exception handling,
- cleaning up during memory deallocation,
- exiting a nested loop,
- creating a loop, and
- performing random jumps in spaghetti-code fashion.

The developers used GOTO statements predominantly for error and exception handling (more than 80 percent), followed by cleanup (more than 40 percent). This is because languages such as C have no explicit mechanism for error and exception handling or cleanup, such as the try-catch or finalize mechanism, respectively. Such use of GOTO was not one of the ways Dijkstra thought GOTO statements could be misused. Less intuitive uses such as creating and exiting a loop were rarer (about 8 to 10 percent).

Finally, the developers rarely (only about 6 percent of the time) used GOTO statements in the way Dijkstra feared they could be misused (in spaghetti code). So, we could safely say that the developers were careful about how they used GOTO statements so that they would know precisely what code had executed.

We also found that 54 percent of the functions had labels with only one GOTO statement jumping to them. And, more than 90 percent of the functions had GOTO statements that jumped forward to labels that occurred later in code. Thus, there was no option of creating loops. These results reinforced our conclusion that the developers used GOTO statements carefully.

Did the Developers Modify or Remove Any GOTO Statements?

We examined this question because we felt that if developers often changed or modified GOTO statements, the statements probably needed more maintenance and were potentially buggier. Because we couldn't closely examine all the projects, we examined five projects that had a clean commit history with clear commit comments we could use to determine when and why a GOTO statement was removed or modified.

The results surprised us. Given the extent to which the developers had

ABOUT THE AUTHOR

MEIYAPPAN NAGAPPAN is an assistant professor in the University of Waterloo's David R. Cheriton School of Computer Science. He's the editor of the *IEEE Software* blog. Contact him at mei.nagappan@uwaterloo.ca.

used GOTO statements, the number of removed or modified GOTO statements was extremely small. Only 0 to 2.5 percent of the GOTO statements were removed, and only 0 to 3.1 percent of the GOTO statements were modified. Furthermore, only two of the five projects had a GOTO statement removed or modified as part of a bug fix commit. These results reinforced our determination that most GOTO statements weren't used in a harmful way.

My colleagues and I agreed with Dijkstra's observation that GOTO statements can be used in a harmful way. However, our empirical evidence showed that the developers were aware of these dangers (we hypothesized that this was mostly because of the impact of Dijkstra's paper) and used GOTO statements carefully.

So, educators must be careful to present not only the harm in using GOTO but also the safe ways to use it, so that students have all the tools they need to create good software. Our study also taught us that we need to closely and experimentally examine other such conventional wisdom in the world of programming. SW

References

1. E. Dijkstra, "Go To Statement Considered Harmful," *Comm. ACM*, vol. 11, no. 3, 1968, pp. 147–148.
2. MaD70, "GOTO Still Considered Harmful?"; goo.gl/jjG8qI.
3. M. Nagappan et al., "An Empirical Study of GOTO in C Code from GitHub Repositories," *Proc. 10th Joint Meeting Foundations of Software Eng.* (ESEC/FSE 2015), 2015, pp. 404–414.