What is Software Engineering?

Go & Versioning, Part 9

Russ Cox
May 30, 2018

research.swtch.com/vgo-eng

Nearly all of Go's distinctive design decisions were aimed at making software engineering simpler and easier. We've said this often. The canonical reference is Rob Pike's 2012 article, "Go at Google: Language Design in the Service of Software Engineering." But what is software engineering?

> *Software engineering is what happens to programming when you add time and other programmers.*

Programming means getting a program working. You have a problem to solve, you write some Go code, you run it, you get your answer, you're done. That's programming, and that's difficult enough by itself. But what if that code has to keep working, day after day? What if five other programmers need to work on the code too? Then you start to think about version control systems, to track how the code changes over time and to coordinate with the other programmers. You add unit tests, to make sure bugs you fix are not reintroduced over time, not by you six months from now, and not by that new team member who's unfamiliar with the code. You think about modularity and design patterns, to divide the program into parts that team members can work on mostly independently. You use tools to help you find bugs earlier. You look for ways to make programs as clear as possible, so that bugs are less likely. You make sure that small changes can be tested quickly, even in large programs. You're doing all of this because your programming has turned into software engineering.

(This definition and explanation of software engineering is my riff on an original theme by my Google colleague Titus Winters, whose preferred phrasing is "software engineering is programming integrated over time." It's worth seven minutes of your time to see his presentation of this idea at CppCon 2017, from 8:17 to 15:00 in the video.)

As I said earlier, nearly all of Go's distinctive design decisions have been motivated by concerns about software engineering, by trying to accommodate time and other programmers into the daily practice of programming.

For example, most people think that we format Go code with `gofmt` to make code look nicer or to end debates among team members about program layout. But the most important reason for `gofmt` is that if an algorithm defines how Go source code is formatted, then programs, like `goimports` or `gorename` or `go fix`, can edit the source code more easily, without introducing spurious formatting changes when writing the code back. This helps you maintain code over time.

As another example, Go import paths are URLs. If code said `import "uuid"`, you'd have to ask which `uuid` package. Searching for `uuid` on godoc.org turns up dozens of packages. If instead the code says `import "github.com/pborman/uuid"`, now it's clear which package we mean. Using URLs avoids ambiguity and also reuses an existing mechanism for giving out names, making it simpler and easier to coordinate with other programmers.

Continuing the example, Go import paths are written in Go source files, not in a separate build configuration file. This makes Go source files self-contained, which makes it easier to understand, modify, and copy them. These decisions, and more, were all made with the goal of simplifying software engineering.

In later posts I will talk specifically about why versions are important for

software engineering and how software engineering concerns motivate the design changes from dep to vgo.