Go Proposal Process: Large Changes

Go Proposals, Part 3

Russ Cox

August 15, 2019

research.swtch.com/proposals-large

[*I've been thinking a lot recently about the Go proposal process, which is the way we propose, discuss, and decide changes to Go itself. Like nearly everything about Go, the proposal process is an experiment, so it makes sense to reflect on what we've learned and try to improve it. This post is the third in a series of posts about what works well and, more importantly, what we might want to change.*]

The proposal process we have today recognizes two kinds of proposal: trivial (no design doc) and non-trivial (design doc). An initial proposal issue can be just a few lines describing the idea, which is then discussed on the GitHub issue. Most proposals exit the process after this discussion. Typical reasons for declining a proposal at this stage include some combination of:

- The proposal is a duplicate of a previous proposal.
- The proposal is not specific enough to evaluate. For example, issue 20142 suggests to do something—but not something specific—about accidental `nil` dereferences.
- The proposal is not backwards compatible. For example, issue 33454 proposed first to change the type signature of `log.(*Logger).SetOutput` and then was revised to change only the semantics instead. Both would be backwards-incompatible changes.
- The proposal can already be implemented as a package outside the standard library. For example, the problem solved by issue 33454—logging to multiple writers—is easily solved with ‘io.MultiWriter.
- The proposal does not address a common enough problem to merit a change or addition to the language or standard library. For example, issue 26262 suggested a new `%*T` print verb that would have little applicability and is easily implemented as a one-line function if needed.
- The proposal violates a core design principle or goal of the package. For example, issue 33449 suggested adding indirect template calls to `text/template`, but that would invalidate the safety analysis in `html/template`.

The typical reason for accepting a proposal at this stage is that the details are simple and straightforward enough that they can be stated crisply and clearly without a design doc, and there is general agreement about moving forward. The specifics can range from significant to trivial, from issue 19069 (extend release support timeline) to issue 18086 (add `json.Valid`) to issue 20023 (document that `os.NewFile` returns an error when passed a negative file descriptor).

Other proposals are intricate enough or have subtle enough implications to merit writing a design document, which we save in the proposal design repo. For example, here are the design documents for testing subtests alias declarations (abandoned), type aliases, and mid-stack inlining.

The brief issue tracker discussion contemplated in the proposal process works well for small changes, even those with subtle implications requiring a design document. But it breaks down for what we might think of as large changes. Larger changes don't necessarily have longer design docs, but they tend to have

larger issue tracker discussions, because more people are affected by the result. Recent examples include the discussion of the Go 1.13 error value changes (404 comments as of August 1) and of course the try builtin (798 comments).

Large Checklist

One idea raised was to leave the process for small changes unaffected but add more process for large changes. That in turn requires identifying large changes. One suggestion was to create a rubric based on a checklist of how much of the Go ecosystem a change affects. For example:

- Is the change user-visible at all?
- Does it require any changes to any documentation?
- Are its effects confined to a single package?
- Does it require changes to the language spec?
- Does it require users to change existing scripts or workflows?
- Does it require updating introductory materials?
- And so on.

If the score is high, the proposal could require more process or at least more care. And if the proposal is accepted, the checklist answers would help plan the rollout.

Large Process

One piece of added process for large changes could be a pre-proposal stage, when the design is simply a draft being explored, not a change ready to be proposed and decided on. We consciously avoided the word "proposal" last year when we published the Go 2 draft designs last summer, to make clear that those designs were still in progress and being proposed as-is. That clear framing served everyone well, and, in retrospect, "try" was a large enough change that the new design we published last month should have been a second draft design, not a proposal with an implementation timeline. (Contrast the urgency many felt during the "try" discussion with the relative calm around last week's second draft design for generics.)

Another piece of added process for large changes could be a pre-proposal experimentation period of some kind (more on that in a future post).

And when a large change does migrate out of draft design to be formally proposed for acceptance, another piece of added process could be a different kind of discussion, or more discussions (more on that in a future post too).

I filed issue 33670 to track the idea of identifying large changes and adding more process for them.

Next

Again, this is the third post in a series of posts thinking and brainstorming about the Go proposal process. Everything about these posts is very rough. The point of posting this series—thinking out loud instead of thinking quietly—is so that anyone who is interested can join the thinking.

I encourage feedback, whether in the form of comments on these posts, comments on the newly filed issues, mail to rsc@golang.org, or your own blog posts (please leave links in the comments). Thanks for taking the time to read these and think with me.

The next post is about how proposal discussions might be scaled to handle large changes with many impacted users giving feedback.