Thinking about the Go Proposal Process

Go Proposals, Part 1

Russ Cox

August 5, 2019

research.swtch.com/proposals-intro

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 first in a series of posts about what works well and, more importantly, what we might want to change.

This post is focused on where we are now and how we got here. Later posts will focus on specific areas where we might improve. At our annual contributor summit held at Gophercon last week, about thirty or so contributors from outside Google attended and helped us on the Go team think through many of these areas. Their suggestions figure prominently in the posts to follow, as, I hope, will yours.

(If you are curious what the contributor summit is, here's Sam Whited's recap of our first such summit in 2017. This year's was pretty similar in spirit, although with different discussion topics. This year there were about 60 people in the room, roughly evenly split between members of the Go team at Google and contributors from outside Google.)

Proposal Process

When we started Go, we launched with instructions on day one detailing how to send us code changes, which are of course the core of any open source project. Around five years ago we noticed that despite the many code contributions, most change proposals were made by the Go team at Google, even when motivated and driven by feedback from the broader Go user community. One reason, we realized, was that the process for proposing changes was nearly completely undocumented. To try to address this, we introduced a formal change proposal process in 2015, now documented at golang.org/s/proposal. For more background, see Andrew Gerrand's 2015 GopherCon talk, starting at 27m17s (only a few minutes). In that talk, Andrew said, "It's important to note that this process is an experiment. We're still kind of discussing exactly how that process should work."

I talked at GopherCon this year about the experiment, simplify, ship cycle we use for just about everything. Like with other parts of Go, we've learned from our experiments using the proposal process and made adjustments in the past, and of course we intend to keep doing that.

The current proposal process is documented as four steps:

1. The author creates a brief issue clearly describing the proposal. (No need for a design document just yet.)
2. Discussion on the GitHub issue aims to triage the proposal into one of three buckets: accept; decline; or ask for a detailed design doc addressing an identified list of concerns.
3. If the previous step ended at accept/decline, we're done. Otherwise, the author writes a design doc, which is discussed on the GitHub issue.
4. Once comments and design doc revisions wind down, a final discussion aims to reach a final accept or decline decision.

See the full document for details.

A Random Sample

As I write this, there are 1633 GitHub issues labeled Proposal. Of the 1187 that have been closed, 170 were accepted, about 14%.

To get a sense of the submissions, discussions, and outcomes, here are twenty selected at random (by a Perl script).

- #11502 A security response policy for Go (39 comments, accepted)
- #14991 add a builtin splice function for easier slice handling (7 comments)
- #16844 freeze net/rpc (20 comments, accepted)
- #17672 remote runtime (9 comments)
- #18303 flag.failf should return an error with Cause() error (15 comments)
- #18662 Struct field tags for directional Marshal/Unmarshal (6 comments)
- #21360 add a build tag "test" (14 comments)
- #21592 add an in-memory writer/seeker to io package (9 comments)
- #22247 Add sync.Map.Len() method (5 comments)
- #22918 go/doc: consts/vars should be grouped with types by their computed type (31 comments)
- #23331 encoding/json: export the offset method of the Decoder (7 comments)
- #23789 add uuid generator to stdlib (13 comments)
- #24410 Add some way to explore packages and structs inside. (6 comments)
- #25273 add token to syntax to reduce error boilerplate (2 comments)
- #25518 x/vgo: allow aliases in go.mod (10 comments)
- #25670 don't include cgo with net on Unix by default (7 comments)
- #26803 mime/multipart: add (*Reader).NextRawPart to avoid quoted-printable decoding (9 comments, accepted)
- #26822 flag: clean up error message (20 comments, accepted)
- #30886 cmd/go: allow replacing a subdirectory within a package (4 comments)
- #31041 package and file organisation (7 comments)

One is significant. Most are small. A few are not well-defined. Most had fewer than ten comments. This is typical.

Process Evolution

For many people, the proposal process represents not those small changes and suggestions in the random sample but instead larger proposed changes, like type aliases (2016), monotonic time (2017), Go modules (2018), new number literals (2019), and the abandoned "try" proposal (2019).

These large changes are of course important, and in each of these we've learned a bit more about what works and what doesn't for making successful changes.

The discussion of the original aliases proposal helped us understand the importance of motivating changes, like my codebase refactoring talk and article motivated type aliases. During that experience, someone introduced me to Rust's "no new rationale" rule, which we have tried to follow when making difficult decisions since then. I reflected more about motivation for changes, using both aliases and monotonic time as examples, in my GopherCon 2017 talk kicking off Go 2.

Although there were parts of the Go modules proposal that did not go well, the general approach of spending significant time discussing the ideas before starting the formal proposal process did seem to help: at the time that we accepted the modules proposal, the GitHub conversation and reactions were overwhelmingly in favor.

One thing we learned from aliases and then from modules was the importance of having an implementation people can try and also the importance of having the changes ready to be committed at the start of a development cycle. We adopted this idea explicitly for Go 2 language changes, and it was successful for the smaller Go 1.13 changes like the new number literal syntax.

For the recent "try" proposal, we followed the evolving process, including making changes available for people to use early in a cycle, and the discussion and reactions were much more heated than we expected. We abandoned the proposal only a week before the summit, and one thing was eager to discuss with contributors was what had been different about "try" and how to continue to improve the process to make future changes smoother. (If the discussion about "try" was difficult, what will happen when we discuss generics?)

Improvement Areas

The discussions we had over six hours or so at the contributor summit surfaced at least six different areas where we might improve the proposal process specifically and the Go project's community engagement more generally. I plan to write a post about each of these themes, but I'll summarize them briefly here too. I'd be happy to hear suggestions for any other important areas that I've missed.

Clarity & Transparency. Adding clarity and transparency about how we make changes to Go—making that process easier to follow and to participate in—was the original motivation for creating the proposal process. There's more we could do, including publishing a record of proposal decisions. (Most of the proposal review group's time is spent not on decisions but on adding people to issues, pinging requests for more information, and so on.) *Update*: See the "Clarity & Transparency" post for more thoughts.

Scaling the Process. The proposal process is meant to lightweight enough to apply to very small changes, such as the recently accepted proposal to add a `SubexpIndex` method to `regexp.Regexp`. As the proposed change gets bigger, it may make sense to introduce additional process. For example, we were careful at Gophercon 2018 to publish our thoughts about error handling and generics as "design drafts" not proposals. For a large enough change, perhaps publishing iterating on design drafts should formally be the first step of the process. *Update*: See the "Sizing Changes" post for more thoughts.

Scaling Discussions. GitHub's issue tracker is not particularly effective at large discussions. For large changes, we may want to investigate alternatives, and we certainly want to make sure to have more discussion long before it is time to make any decisions. For example, publishing multiple design drafts, giving talks, and publishing articles are all ways to engage helpful discussion before reaching the point in the proposal process where decisions are being made.

Prototypes & Experiments. For most non-trivial changes it is helpful to understand them by trying them out before making a decision. We do this as a

matter of course for small changes: we always have at least a few months between when a change is made and the corresponding release, during which we can reconsider, adjust, or remove it. We arrange to land language changes on day 1 of a development cycle to maximize that window. But for large changes we probably need a way to make prototypes available separately, to give even more time, a bit like the vgo prototype for Go modules.

Community Representation. Andrew said in 2015 that he hoped the proposal process would "make the process more accessible to anybody who really wants to get involved in the design of Go." We definitely get many more proposals from outside the Go team now than we did in 2015, so in that sense it has succeeded. On the other hand, we believe there are over a million Go programmers, but only 2300 different GitHub accounts have commented on proposal issues, or a quarter of one percent of users. If this were a random sample of our users, that might be fine, but we know the participation is skewed to English-speaking GitHub users who can take the time to keep up with the Go issue tracker. To make the best possible decisions we need to gather input from more sources, from a broader cross-section of the population of the Go community, by which I mean all Go users. (On a related note, anyone who describes "the Go community" as having a clear opinion about anything must have in mind a much narrower definition of that group: a million or more people can't be painted with a single brush.)

Community Coordination. We have had mixed results attempting to engage the broader Go community in the work of developing Go. The clearest success is the technical development of the Go source code itself. Today, I count exactly 2,000 email addresses in the Go CONTRIBUTORS file, and only 310 from google.com or golang.org. The next biggest success is probably the proposal process itself: I estimate that the Go team accounts for about 15% of proposals overall and about 30% of accepted proposals. We also created a few working groups, most notably the package management committee in 2016 and the developer experience and community outreach working groups in 2017. Each one had aspects that worked well and aspects that didn't. More recently, the golang-tools group started in 2018 is coming up on its first birthday and seems to be operating well. We should try to learn from the successful and unsuccessful aspects of all these groups and try to create new, successful, sustainable groups.

Next

I plan to post about a new theme every day or two, starting with the ones in the previous section, until I run out of interesting thoughts.

Please remember as you read these posts that the goal here is thinking, brainstorming, looking for good ideas. There are almost certainly bad ideas in these posts too. Don't assume that everything I mention will happen, especially not in the exact form described. 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 the posts, 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.