Using GitLab Issues for Iterative, Asynchronous Software Design

A whitepaper for the 2023 Collegeville Workshop on Scientific Software, focussing on Software Design.

Jason M. Gates; *Sandia National Laboratories*

When much of the world transitioned to working from home in early 2020, the software development world was, in a sense, largely primed for the switch. For many of us, all we need is a laptop and an internet connection to do our development work. We can clone our repositories and access our issue tracking systems from wherever we like. This assumes, however, that we already know the work that needs to be done and how we intend to go about doing it, but what about when *designing* software? When everyone was together in the office, we would gather in a conference room and use whiteboards, post-it notes, and a good deal of discussion to determine the design. This was an aspect of our process that didn't directly translate to a virtual work environment. A few years into the all-remote setting, many teams still struggle with it.

Your team might have tried to mimic the conference room design session virtually using video conferencing and perhaps even some sort of digital whiteboard application. Chances are the experience was sub-optimal. When transitioning to hybrid- or all-remote, studies show it is best to not replicate the in-office experience remotely. Instead, the recommendation would be to adopt a handbook-first documentation paradigm, where the focus is on written, asynchronous communication. What exactly does that mean in practice? There's not necessarily a one-size-fits-all solution, but we hope to inspire you with how one team of eight developers spread across four cities and two time zones used GitLab issues as the medium to enable iterative, asynchronous software design on a fast-paced project.

GitLab to the Rescue

Many modern-day repository hosting services come with built-in features to enable issue tracking and code review that can be leveraged for design needs. We chose to use GitLab, but many of the recommendations that follow may apply to your tool of choice as well. The remainder of this section includes detailed recommendations for using GitLab as a tool to enable asynchronous design discussion. To start a new design discussion, simply create a new issue, as you would for a bug report, a feature request, etc.

Draft the Description

When filling out the issue description, *don't start designing the piece of software yet*—instead, establish a starting point for the discussion. Consider including information such as:

- The overall objective of the piece of software under design—i.e., what does it need to do?
- Necessary inputs or outputs, or pre- or postconditions.
- Constraints imposed on the design (timeline, manpower, cost, etc.).
- Customer requests that may impact the design. (Consider using the "As a *role*, I want *desire*, because *motivation*" user story form, though other options abound.)

For ease of use, we recommend creating an issue template to help you include all the relevant details when you start each new design discussion. You can also create a label to make it easy to find any such issues in the future, and use a quick action in the issue template to automatically apply it to all design issues.

Propose the Initial Design

Once the description is complete, draft an initial design. Start typing it up in the first comment on the issue. Keep in mind that it doesn't need to be perfect—this is just a first pass. If you're unsure of any areas of the design, call them out so others know where more refinement is needed. Be sure to refer back to the specifications given in the issue description to ensure the design is meeting the needs and to highlight what holes need to be filled in later. This first comment on the issue can simply be text-based, but using GitLab-flavored Markdown can give you a richer communication experience (explained in more detail below).

Invite Collaboration

Once the initial design is in place, direct the rest of your team to the issue and invite them to participate. If they have questions or comments on the specifications given in the issue description, have them comment at the bottom of the issue page. However, if they want to contribute to someone else's comment (including your initial design proposal), make sure they reply to the comment to start a thread. In this way you keep the asynchronous discussion organized and easy to follow on the issue page.

Iterate on the Design

As the discussion continues, you'll eventually get to a point where the team realizes the initial design proposal is insufficient to meet your needs. If you were to edit that first comment, you would confuse the story that all the comments on the issue are telling through time, so instead simply comment again at the bottom of the issue page to start a new thread. Rather than typing everything up again from scratch, you can copy, paste, and then modify the initial design write-up. You may find it worthwhile to use headers to distinguish these design proposal comments, making them easier to notice when scrolling through the issue conversation.

Finalize the Design

Once your design discussion has come to a close, either through consensus, or through the decision of the directly responsible individual, be sure you capture the final version of the design in one last comment at the bottom of the issue. In this way, as you scroll through the issue, you can see the initial requirements, followed by the different iterations the team went through, and concluding with the final version that should meet all the requirements. Just like `git` can preserve the history of your software development efforts, so can GitLab preserve the history of your design discussions.

GitLab-Flavored Markdown for Richer Communication

As mentioned previously, it's possible to do all the above with plain text, but your communications can be enhanced with GitLab-flavored Markdown. Markdown is a plain text markup language that is intended to be easily human-readable, while at the same time allowing for translation to richer communication media such as HTML, PDF, etc. GitLab will render Markdown to HTML in issue and merge request descriptions and comments, as well as wiki pages and any `*.md` files in your repository.

Basic Functionality

The standard Markdown functionality we have found most useful is the following:

- Headers allow you to logically organize what you're writing into sections, subsections, etc.
- Lists and tables help you to present the information on the page in an easy-to-digest manner.
- Links are important to direct people to more information. If at any time you're referring to details that are captured elsewhere, link to them so there's no confusion.
- Blockquotes give you the ability to denote text as being quoted from another source.
- Code blocks, which include syntax highlighting for a variety of languages, allow you to include snippets of code, or perhaps pseudo-code, in the design discussion for the times when you need to make the details concrete.
- You can also embed images, video, and audio to show your collaborators exact representations of concepts or ideas.
- If you need additional flexibility, you can include a subset of raw html. For example, you can use this to hide extra details in a collapsible section.

GitLab Goodies

To add to the basic functionality, GitLab extends Markdown with a number of helpful features:

- In addition to standard lists, task lists allow you to check items off as you go, and GitLab will show you at the top of the issue how many of the total tasks have been completed.
- In addition to standard links, there are also a number of GitLab-specific references you can use that will automatically cross-link to other GitLab entities (e.g., issues, merge requests, teams, etc.).
- If your team culture is amenable to it, the use of emojis can both lighten the mood and help to convey intent when body language isn't available. See The Turing Way as an example of a project that does this well.
- If the work you're doing is scientific in nature, you'll likely find it helpful to include math written in LaTeX syntax.

Mermaid Diagrams

In addition to all the rich text formatting, you also have the ability to embed diagrams into comments using a Mermaid code block. Mermaid is a tool that translates plain text into rich diagrams using a Markdown-like format. The next time you need to create a flowchart, class diagram, pie or Gantt chart, etc., you can do so right in GitLab rather than relying on a separate application. That said, if you find that Mermaid doesn't provide all the capability you require, you can also use GitLab's design management feature to keep track of uploaded design artifacts in a variety of formats.

Why Should I Make the Switch?

Given all that you've read so far, you may be thinking that this seems like a good deal of change to implement. Will all the effort be worthwhile? After all, won't things be going back to normal soon, such that we can meet back up in our conference rooms again? Early indicators say "no", with some predicting that 90% of those who aren't comfortable with remote work will retire in the next decade. The landscape of the working world is shifting, whether we like it or not, and this particular transition is arguably a good one. The principle of transparent, asynchronous communication makes it such that everyone can contribute, which increases inclusivity, and promotes innovation.

As far as design discussions go, the benefits include the following:

- Goals are explicitly defined. The ability to easily refer back to them can help keep your team on task.
- You're encouraged to thoroughly explore your ideas before adding them to the conversation. This tends to generate clearer communication and higher quality contributions.
- The complete history of the design discussion is captured. No more, "Does anyone remember that one idea we erased?"
- The justification for various decisions can be referenced when making future decisions. No more, "Why on earth did we pick this? Let's just change it."

On top of this, you'll also find that discussing the design in detail ahead of time eases and accelerates the development of the software. You'll be tempted to think that all the time you spend in the design phase isn't "real work" because you haven't coded anything yet. However, when the design is thought out well in advance, the implementation flows naturally from it. You may be pressured to minimize design time in favor of getting into the code and starting to deliver minimum viable increments of value, but if you do so, you're likely to build far more technical debt than if you hadn't. You may face criticism that such a process isn't "agile" enough, but keep in mind that the process described above is highly iterative and collaborative. Once you begin iterating on the implementation, you always have the ability to go back and rethink things as new information arises.

The benefits are clear enough, but how much time is this actually going to take? After all, it's relatively easy to share ideas in person and prototype them on a whiteboard. How much longer will I spend trying to do all the same things asynchronously? Unfortunately there's no one answer to this question, as it depends on a number of factors:

- How familiar is your team with the tools involved? If you're starting from scratch, there will be some time spent getting used to them, but if you're already proficient, then the time required to, e.g., create a diagram, is largely inconsequential.
- How familiar is your team with open source development practices? If you're used to the majority of your communication happening through issues and merge requests, then adopting the same for design discussions will come naturally. If not, there's a significant cultural barrier to be overcome, which will necessarily slow down communication at first.
- How comfortable is your team with the all-remote work environment? If you're used to only synchronous communication, the shift to asynchronous can add a good deal of angst and confusion to the process. However, if you're already used to asynchronous communication, then these suggestions will naturally fit into how you already do your work.

For the small, all-remote, open source team, these suggestions will be easy to implement, and you'll quickly see a return on investment. For the large team used to collocated work and unfamiliar with the open source paradigm, it will take much longer to work these recommendations into the culture. The investment will be well-worth the effort in the long run, but be patient and have grace for one another as you all adjust to the new norm.

Discussion

Does this mean we'll never have meetings ever again? While some developers may relish the thought, the answer there is "no" as well. However, when you do need to sync up, following guidelines for remote meetings can make those times more productive and enjoyable than they were when they were in person.

The goal in transitioning to this new communication paradigm isn't to hamper individuals' unique social styles; rather, the goal is to improve both your ability to deliver top-quality software and the quality of life for you and your team, regardless of social style. In the end, our geographically distributed team was able to design and build a modular ecosystem of packages that replaced an existing infrastructure with a three-fold reduction in complexity, and we did it faster than anticipated. GitLab was instrumental to making our efforts a success—both the tool, and the recommendations from the company that makes it. We hope it can be similarly transformational for you.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2022-12393 C