Move Fast And Retain Corporate Governance with Pull Requests

2019-02-24 17:47:23

DevOps and change control mix like oil and water.

Product and development teams want to experiment with and release ideas as quickly as their customers request them, and do so with tight, but unstructured, collaboration across organizations. On the other hand, corporate governance wants auditability, transparent risk mitigation and justification in every step of the way.

Consequently, both of these sides often don’t get along with each other well, hindering development speed in the progress. But it doesn’t have to be this way.

# In Defense of Pull Requests

[Pull requests](https://www.atlassian.com/git/tutorials/making-a-pull-request) are often used for merging source code from a development branch to a more authoritative branch on the path to production. However, pull requests can also be an excellent tool for upholding corporate compliance standard and risk tolerance.

## Pull Requests Enable Collaboration

{{< post\_image name=“collaboration” alt=“Collaboration is easy with PRs.” >}}

It is no coincidence that major open source projects such as [Kubernetes](https://github.com/kubernetes/kubernetes) and the [Linux kernel](https://github.com/torvalds/linux) perform massive code reviews entirely through pull requests.

Pull requests make it easy for project collaborators and organizations to review, approve and comment on code changes. When project maintainers add organizers to a project, those organizers (along with any watchers interested in the progress of that project) will be asked (by email, Slack or many other communication mediums) to participate in a code review. How that code review happens is completely up to the team.

I’ve seen high-powered teams do group review sessions where people from product, engineering, platform/sysadmin and other relevant business areas gather in a room and review code on a big screen. I’ve found sessions like these to be very helpful at ensuring the readability of the code, as you can’t review code that you don’t understand, and asking everyone in the session to become a Java developer (for example) is a big ask. Once the group reaches consensius that the code to be released is “good,” then the project maintainers merge the changes up into the authoritative branch, and the code is shipped on its way to customers.

Open-source projects like the ones mentioned above do completely asynchronous reviews where organizers and collaborators review the code as they are able. While this adds some latency and redundancy to the process, it gives everyone that is interested in the project the chance to look at changes without feeling like they have to make a decision right then. This system also works really nicely for distributed teams, since asking everyone to gather at one time can be tough on people across different time zones. In this model, the code goes to production (or an upper environment) after two “thumbs-up” emojis or “looks good to me” (LGTM) comments.

*“I’m in Risk Management, Carlos; why should I care,”* you’re probably asking.

{{< post\_image name=“gathering\_approvals” alt=“Oh no, its approval’s gathering time.” >}}

Gathering approvals to execute changes is usually painful. Many ITSM platforms request approvals via email (though [ServiceNow can integrate with Slack](https://contino.slack.com/servicenow)), and spamming people to have them comb through their already-overloaded inboxes is a chore. As well, lists of approvers to be added onto changes are usually added automatically depending on their severity level. Routine, low impact work can usually go through without approvals, but emergency changes, per their name, often require director level approval or above. Adding “busywork” onto the plates of already-very-busy people can be tiresome.

***Pull requests simplify this by shifting the approval process left towards the codebase***. Having approvals reside in the same place as where code gets made makes it easy to keep track of where everything is. This can also be useful come audit-time, as managers can tell auditors to check their Bitbucket repositories instead of having to sort through ServiceNow, Bitbucket, Word/Excel and who knows where else signoffs live.

## Pull requests add safety.

{{< post\_image name=“safety” alt=“Pull requests add safety.” >}}

The other large value proposition for pull requests is their ability to invoke further automated quality checks. These are often done through [Webhooks](https://sendgrid.com/blog/whats-webhook/), which allow systems to send events to other, disparate systems over the web through good-ol’ HTTP.

Many projects use Webhooks for running tests against the pull request as part of continuous integration. For instance, the HashiCorp [Terraform](https://github.com/hashicorp/terraform/) project integrates their pull requests into various SaaS-hosted style checkers, vulnerability scanners and their own testing pipeline. This way, change reviewers don’t have to waste time on pull requests that will probably break something and maintainers have greater confidence in merging code that will probably work.

Submitters of a change are often given a questionnaire to fill out that, at its end, will yield an impact score. What if you could have these risk and impact scores calculated automatically, with near 100% correctness, for every change without asking your developers to submit anything? What if you had processes that could scan a source code branch for known threats, vulnerabilitiies and risks and attach the report directly to the change without any additional clicks? ***Pull request webhooks enable this functionality and more!***

# An example pull request workflow

Asking organizations to dump ServiceNow in favor of pull requests is like asking to dump Oracle Financials — it ain’t gonna happen. But we can have both!

{{< post\_image name=“workflow” alt=“A path to automated change control” >}}

The image above is an example of how automated change control *could* work for an organization. The workflow is generic, since every organization does change control a little bit differently. It also isn’t immutable; feel free to mix and mash it as you’d like! Also, this process is probably too heavy-handed for “internal” pull requests from one development branch to another development branch, so some thinking on the kinds of pull requests that this applies to will be useful.

1. When ready, developers create pull requests as they usually would. Bigger organizations might find tagging pull requests with their relevant business units and projects helpful here.
2. Magic begins once the PR is opened.
3. The source code collaboration platform will handle adding and notifying approvers, though you might want to extend this based on the risk and impact scores.
4. A webhook runs that runs unit, integration and acceptance tests against the pull request branch.
5. Another webhook runs that runs security and compliance tests against the code. Depending on the platform, this can take a while, so you might want to consider running a shorter subset of security scans for PRs against upper environments and more exhaustive scans against PRs destined for production.
6. Another webhook runs for risk and impact calculation. These can hook into your ITSM suite of choice and use a combination of a file containing pre-written questionnaire answers and tags applied to the pull request. This can also be a custom service that runs outside of your ITSM and synchronizes results back into it.
7. A background webhook runs throughout the entire pull request process that synchronizes data from the pull request with the change filed within the ITSM system.

If *any* of these hooks fail, you should see a red “X” somewhere on the page to warn reviewers that they are looking at dangerous code. Stricter teams might want to auto-reject the pull request depending on the failure encountered. Note that most pull request systems *will preserve* the pull request so that developers can simply write their fixes and update the PR in kind.

Once all approvals have been obtained, the developer or a project maintainer can merge the code into the authoritative branch.

# Change control doesn’t have to be awful

Understaffed and overworked changed control boards are often blamed for hampering innovation and experimentation. Reflexively, change control boards often blame engineering for not understanding the importance of “getting the process right” and how disastrous side-stepping it can be. DevOps isn’t just about bridging development and operations together; it’s about creating bridges between engineering and business, too, and this divide is one example of that.

Pull requests can help build this bridge. Development teams already understand the value that pull requests bring, but their ability to help organizations keep risk and compliance in check is not as well-understood. By taking advantage of webhooks and repository ownership, engineering teams and corporate governance can all “shift left” and start talking to each other through common sets of tools. Pull requests help corporate governance be a champion for moving fast, safely.