A Practical Guide to Offboarding Developers

GitKraken

GitKraken

A Practical Guide to **Offboarding Developers**

Summary

Losing team members is not fun. Whether it's by choice as the developer moves on to pastures new, or through the large layoffs we're seeing right now around tech. For engineering managers handling the offboarding, the priority is ensuring the remaining team has the knowledge and context it needs to continue to be productive.

This hands-on guide walks you through how you can use code visibility and code automation to manage the offboarding process and keep productivity high with the rest of the team.

3 Losses to Your Team When a Developer Leaves

When a developer leaves, a manager's first thought will be about the here and now– the exact feature they are working on, this week's sprint, or how you're going to cover bugs or support requests.

But losing team members means much more than the immediate work impact. This person has been enmeshed in your organization, your team, and your codebase, maybe for years. They've built up knowledge not just of the product, but the codebase, the architecture, the processes, and the system. It's a definite "never send to know for whom the bell tolls" moment.

They take with them all the implicit knowledge and the context for that knowledge, as well as impacting the productivity of your team.

1. You lose all the knowledge that person had

Losing a developer with deep knowledge creates significant challenges, especially if the developer has been a key member of the team. When devs leave a company, the company loses:

- **Technical skills and knowledge:** The developer will take their technical skills and knowledge with them and the proficiency in specific languages, frameworks, libraries, and tools that the company uses.
- **Domain-specific knowledge:** The developer will have specific knowledge about the industry, customers, or users that the company serves. They may also have insights into the company's unique challenges and opportunities.
- **Codebase knowledge:** The developer will have an understanding of the company's codebase, including architecture, design patterns, and coding standards. They may know which parts of the codebase are well-maintained and which areas need improvement.
- **Process knowledge:** The developer will be familiar with the company's development processes, including project management, quality assurance, testing, and deployment. They may know which processes are effective and which ones need improvement.

- **Tribal knowledge:** The developer may have institutional knowledge that is not written down or documented, such as unwritten rules, tips and tricks, or workarounds.

Each of these will impact the company at different levels. Technical knowledge is often quickly replaceable, but domain, codebase, or process-specific knowledge isn't. It's only learned over time within your organization. If someone takes this with them when they leave, they are taking a big part of your organization with them.

2. You lose the context of decisions within your codebase

It's not just the specific knowledge that you lose with the developer. It's also the reasoning and context behind design decisions and code implementations that's lost. Context here could be:

- the **goals and objectives** of the project. Ideally these are well-documented, but there are always uncodified objectives for a project that are known just to the developers.

- the **constraints and limitations** of the tech stack. Tech stacks get chosen for myriad reasons. If a developer pushed for a certain framework then left, it can easily be lost why it was chosen, and the problems it was causing.

- the **preferences and opinions** of the team. These are never written down in official documentation, but they give you an understanding of why choices were made in the design and build processes.

- the **history of the project**. You can sometimes piece this together from commit history, but when developers leave, you lose the ability to tell the whole story of your project.

This leads to confusion and misinterpretation of the code, especially if the company has not been proactive about documenting and sharing information.

3. You lose the productivity from that developer and the team they work with

Even though you are just losing a single developer, this has ramifications for the entire team. Productivity may decline team-wide when a single developer leaves a company because:

- **Increased workload:** When a developer leaves, the remaining team members may have to take on additional work to cover the gaps. This can lead to increased stress and burnout, which can impact productivity.

- **Communication breakdowns:** The departing developer may have been a key communicator or liaison with other teams, or even customers. Without this person, communication may become more difficult, leading to misunderstandings, delays, and other issues.

- **Disruption to team dynamics:** Developers often have close working relationships with their colleagues. When one member of the team leaves, the dynamics of the team may shift, leading to a period of adjustment and reduced productivity.

- **Rebuilding processes:** If the departing developer was a key member of the team, the team may need to rebuild some of its processes, such as code review, testing, or deployment. This can be a time-consuming and disruptive process.

Losing a single developer can have ripple effects throughout the entire team. It's worth noting that the extent of the productivity decline will depend on the specifics of the situation. The impact will be more significant if the departing developer was a team lead or the only person with expertise in a critical area.

Code Visibility Keeps Knowledge and Context in the Codebase

> "GitKraken gives me more visibility into what code is responsible for what parts of the product."
>
> **Ryan**
> Senior Engineer, Stripe

Code visibility is the ideal way to deal with offboarding a developer. Code visibility allows you to document everything they know in a way that allows them to keep their knowledge and context close to the code, and nudges them to remember all that tribal and tacit knowledge and pass it on to the remaining developers.

It also takes the stress out of the process for the developer and the engineering managers:

- **For the developer**, it allows your offboarding engineer to add their knowledge and context directly to the code. It helps them organize what they know so nothing is forgotten. It means that their offboarding process is simplified as they aren't updating numerous documentation or having dozens of calls with other developers. Instead they are creating walkthroughs of the codebase from their perspective.

- **For the managers**, it means you can set out a clear process for offboarding. Everything is coordinated in a single place, close to the codebase. You and the remaining team members can clearly see what has been added by the departing developer and what still needs to be added. It lets you keep knowledge/context permanently in a single place that you can share with anyone in your organization.

For the company, it means you can offboard quicker and lessen the impact on your team. The team can work on more important things. It also helps when replacing the departing team member, as an offboarding tour can become an onboarding tour for new developers.

Sometimes constant offboarding is a factor of the business model. At DistributeAid, volunteers work on the codebase, and cycle in and out constantly. Volunteers leaving DistributeAid need to make sure their knowledge and context are passed on to the next cohort. As Taylor, co-founder of Distribute Aid told us:

> "GitKraken allows Distribute Aid to implement relay-style code handoffs. Everyone is up to date, and we can maintain our momentum without burning out our volunteers."
>
> **Taylor**
> Co-Founder, Distribute Aid

Where Code Visibility Can Help

There are three main ways you can use GitKraken and code visibility to offboard engineers.

1. Embed knowledge in code maps

Code maps allow you to visualize your codebase. Your developers can see how files, functions, and modules interact. Code maps act as the base for tours, but can be used by an offboarding developer to describe the codebase at a higher level.

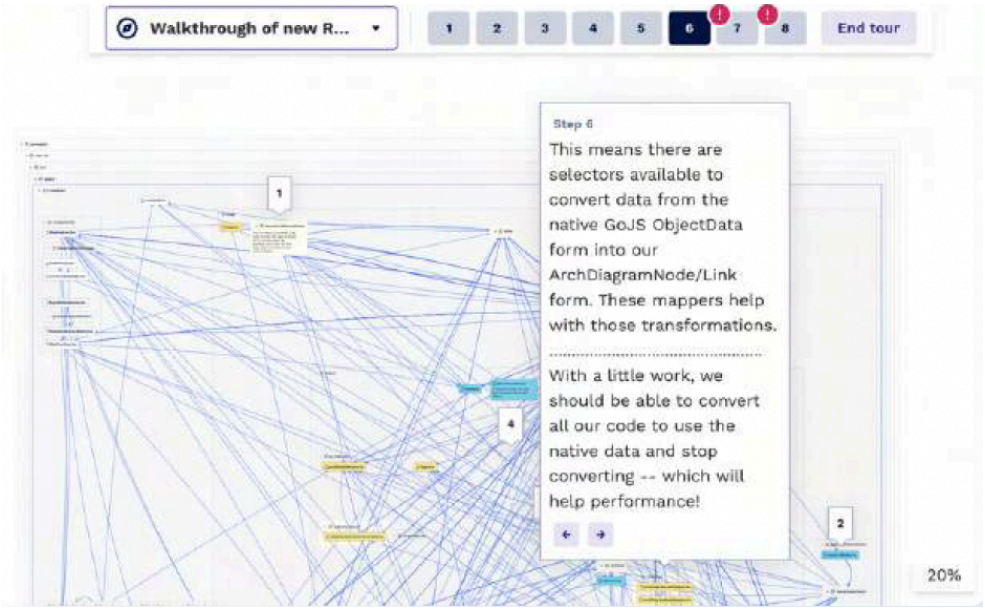
Let's say the developer has been working on technical debt in this codebase and needs to pass on their knowledge to the wider team or the next developer. They can color code each file or directory to show what needs to be done:

In this case, the departing developer is showing some of the tasks that still need to be completed: listing tests, breaking up files, TypeScript conversions, and logging. The Tech Lead can then assign each of these tasks to different developers on the team, or prep these jobs for an incoming developer. It also sets up the conversations between the departing developer and the team. For instance, the offboarding developer might have good insight or processes for converting JS to TS that it would be great to share with the team.

2. Adding context through tours

When more time is available, getting a departing developer to add codebase tours allows them to add context to the knowledge.

Let's say the developer leaving is the team expert on Redux. In this tour they create a number of different steps showing how Redux is organized and works:

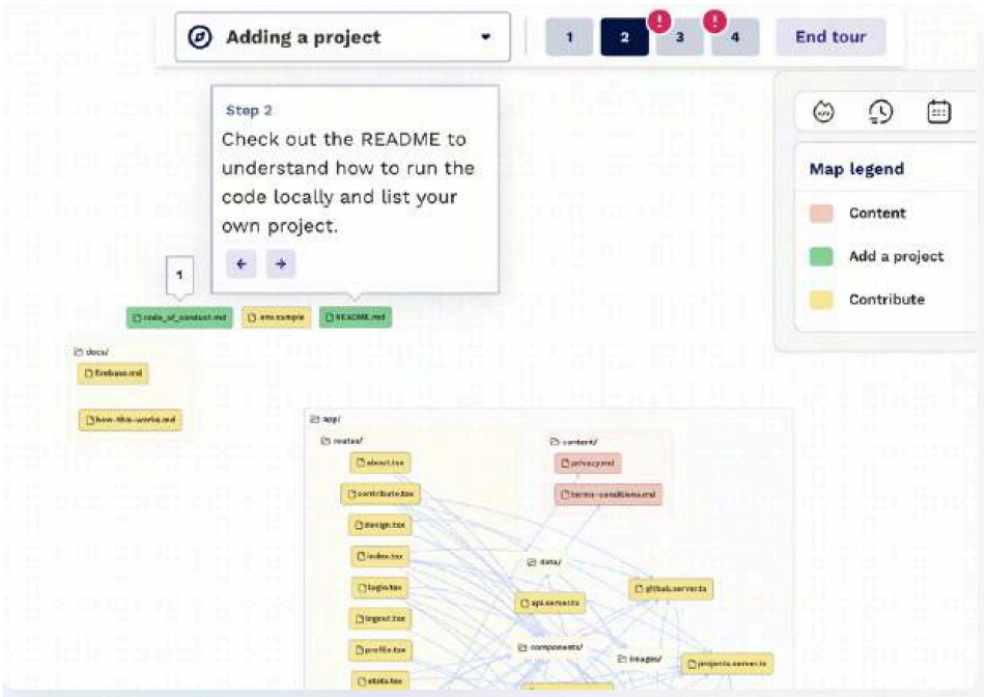


This type of tour can be used in two ways

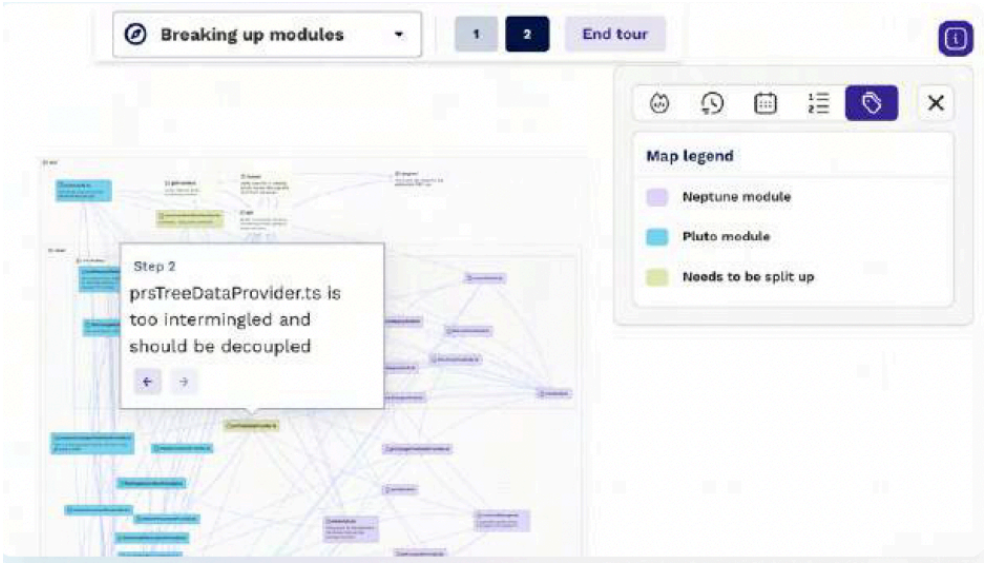
- Synchronously, where the developer gives a team-wide talk as they walk through the steps and answer any questions the team has

- Asynchronously, as this continues to be available to the team and any other new team members after the developer has left.

In this case, the tour is for the developers' peers, other developers. But you could also have tours for junior developers, showing the basics of the codebase or the best way to get started:



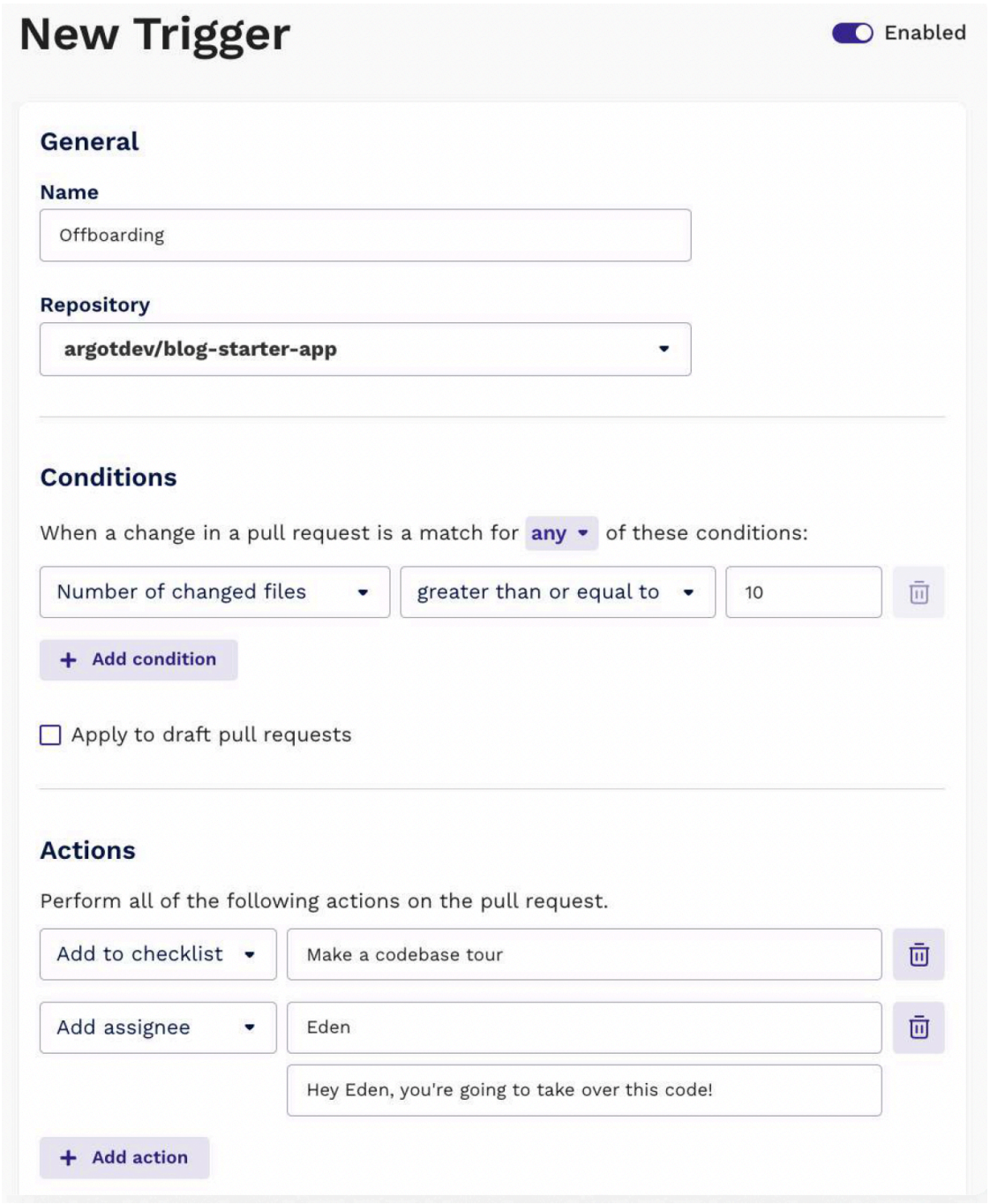
And you can have tours for managers, potentially showing current bottlenecks in the codebase or where refactoring can help:



3. Use code automations to add messages and checklists during the offboarding process

Offboarding is a process. If either the leaving developer of the lead managing the process misses something, it can have an impact on the whole rest of the team and productivity for a long time. You can manage the whole process through code automations to continue to center the offboarding around the codebase.

Using code automations, you trigger comments, checklists, and specific reviewers when changes are made to the codebase. As a developer in leaving and wrapping up their work, you'll want to make sure everything is assigned to current team members for the future, and that these people have the knowledge and context to continue the work:



So we set the Conditions as “Number of changed files” to “greater than or equal to” 10. So if the new developer is changing more than 10 files within their PR, this automation will be used. Then we set the Actions as “Add to checklist” an entry to Make a Codebase Tour and Add assignee and tag the new developer.

A Better Offboarding Process

Leaving is tough on everyone. It's even tougher when there aren't good processes in place to handle it.

When processes are broken, the developer leaving can feel rushed and pulled in a hundred directions as they fill in documentation everywhere and get on calls to explain all their code and decisions. And they'll walk out the door feeling they still haven't helped their former team enough.

When processes are broken, managers are left scrambling to get as much out of the departing developers head as possible before their time is up, which leaves everyone with a bitter feeling.

But done right, with the knowledge and context of the developer kept close to the codebase using maps, tours, and automations, and the process can run smoothly. Both developer and managers can get what they need out of the process: the new developer can go to their next position free of any burden, and the remaining team can still use their knowledge to build great products.