Mattermost

Best Practices for Software Development

WITH MATTERMOST

Ask any software engineer about the best way to do something and they'll likely tell you "it depends." Every project and team works differently and has specific concerns, requirements and opinions, adding complexity on top of the technical and operational complexity of software development. The challenge lies in managing the work and emergent complexity while staying abreast of industry best practices in a way that fits our teams' particular processes and environment.

First, let's review a few general best practices that can be tailored to fit a team's specific needs. Then, we'll get into some details about exactly how to implement them using Mattermost as a central hub for collaborative workflows. You'll learn how messaging, structured process execution, and task management come together in a way that unites teams and tools across an organization. The result will be less context switching, fewer manual tasks, and more freedom to ship stable software faster.

Common Best Practices

When it comes down to it, a lot of best practices are centered around the same principles. Teams need to agree on how work will happen, document that process, use it, and improve on it. That means you need to:

Communicate clearly and constructively

Have a single source of truth

Follow the agreed on process without barriers

Easily update and share resources

All of these best practices are based on open communication, visibility, and teamwork. Making tools, documentation, and processes publicly available to everyone on the team removes barriers to creativity and fosters alignment. By eliminating hurdles like context switching, siloed information, and poorly-defined processes, a team can more easily get through the day-to-day work and make room for transformation and innovation.

Your goals as a team may include following specific practices like DevOps, GitOps or Agile, or emphasize automation, transparency, and repeatability. To achieve those goals, you'll need systems that support changes to both technical practices and team culture. We recommend starting simple, especially if you have a small team, focusing on getting the process right before adding exciting new software into the mix, and following a similar process to our Seven Steps to ChatOps Guide. Look for common tasks where a 5% improvement would have exponential benefit over time, such as tasks that are easy to automate, and meetings that can be asynchronous. When you understand what your team needs, then spend the time matching those requirements to software and learning how to use it.

For some inspiration, check out this blog post on how we use the Mattermost platform to support best practices at our company.

Keep it agile and visible

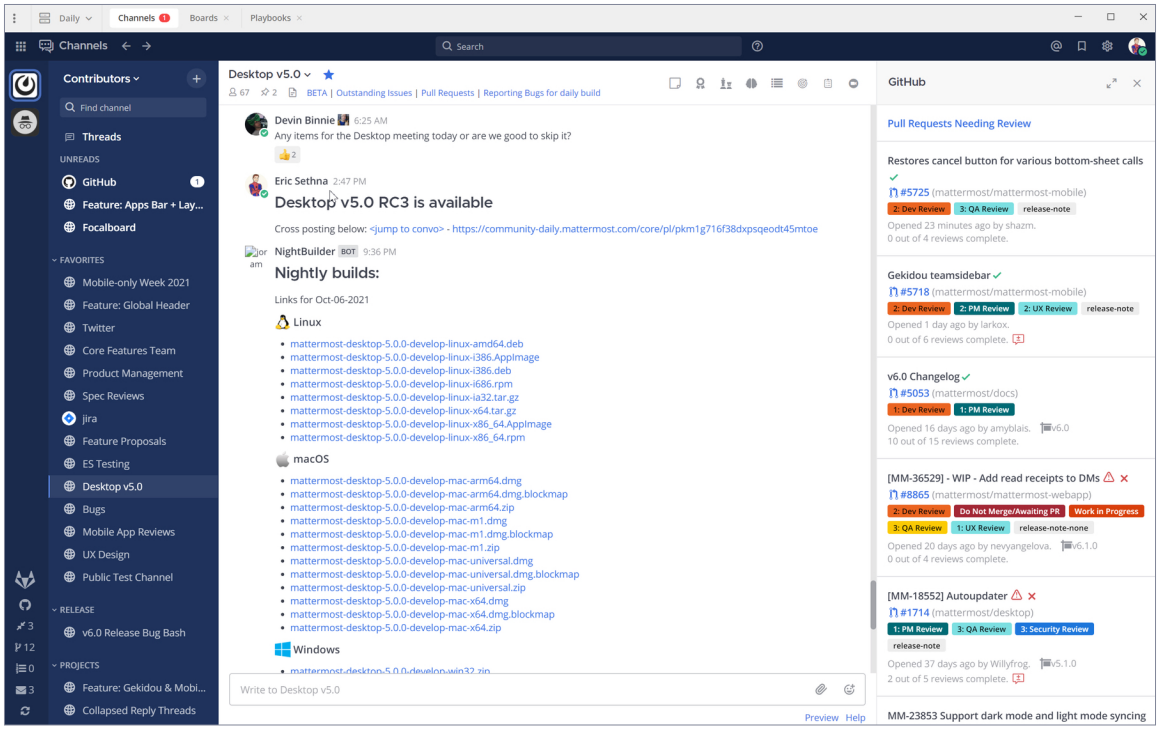
Success is built around trust and open communication. When team members can communicate comfortably and consistently they're more likely to be highly engaged. Globally dispersed teams and the rise of remote work are nearly universal challenges. By putting information in digital spaces where the entire team can access it when and how they need to, everyone can stay on the same page and also help their colleagues self-serve information. Developer knowledge can be shared to the entire organization and junior team members can quickly reference team resources or search for past work to understand the team norms.

Asynchronous communication across these teams is made possible by robust chat platforms with features like threads, markdown formatting, automatic code syntax highlighting, and other dynamic content. All these features add up to a workspace where communication is nuanced, clear, and always with context - available not only in the immediate course of the conversation but in the backscroll for future reference.

A collaboration tool such as Mattermost Channels can be used to communicate fast feedback during the lifecycle of a project so that changes can be incorporated as soon as possible. Small chunks of work, reviewed frequently, can lead to a much more solid final product and less rework in the long term. Team channels include descriptions and pinned items where a team can store frequently referenced resources and links. Open and async communications in channels help establish team norms and culture when senior members lead by example where everyone can see.

Use version control and project tracking

Version control systems are a critical software development best practice. Git is the de facto standard for tracking and managing code changes especially for a distributed team. The key here is that changes are tracked in small increments, and changes can be rolled back to a previous version if something goes wrong. Many developers can be working safely and simultaneously, and anyone can look into the project history to clearly view what happened.



As a developer working with a project stored on say, GitHub, you may get a lot of notifications from GitHub for comments on issues, code review requests, pull requests, and more. Whether those notifications appear in a web browser, via email, or somewhere else, it's one more place you need to check for messages. Mattermost offers a direct integration with GitHub (and GitLab) so that your notifications appear directly in the Mattermost App, alongside your conversations in Channels. You can also interact with your GitHub account using slash commands for the integrated bot to subscribe a channel to a GitHub feed and more, so that everyone in the channel stays up to date.

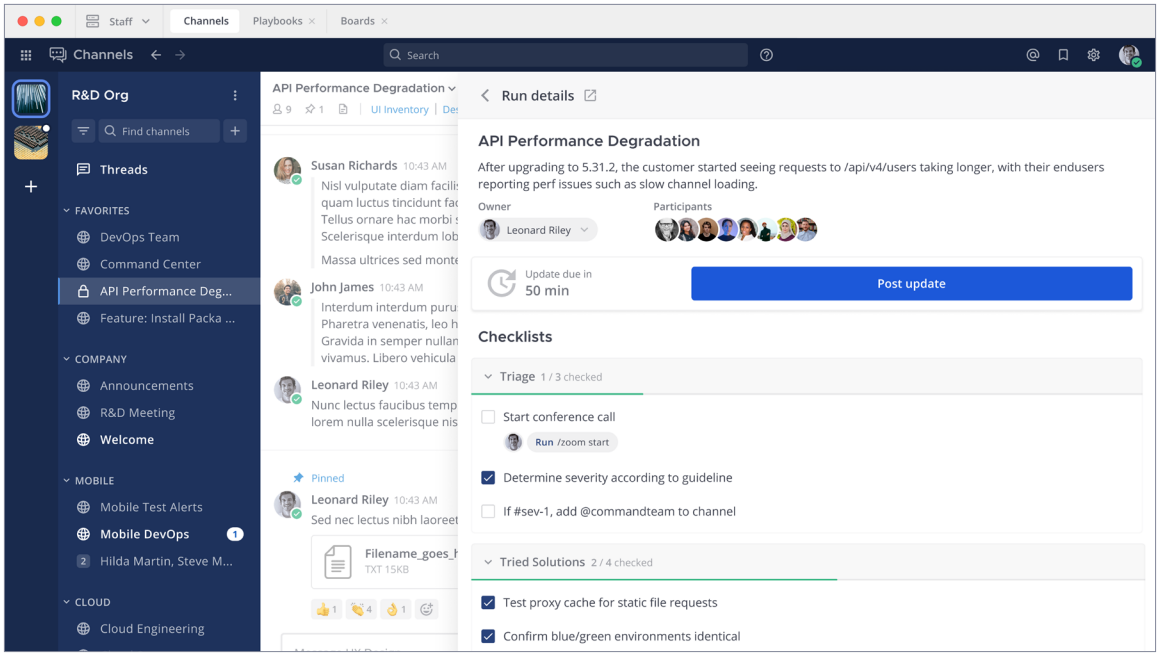
Mattermost also has an integration with Jira, which works similarly. Try running `/jira help` in a conversation with yourself on Mattermost. Without leaving the team conversation in a channel, you can quickly spin up a new issue stub to track a request or bug. Any message can also be used to create or add to a GitHub issue or Jira ticket by clicking the message actions menu, then `create` or `attach to`. Team members can shut off notifications from everything but Mattermost and stay focused while still receiving those critical review requests that keep work moving forward.

With project tracking, the important thing is that work and the team's time is counted, credited, and visible to everyone. It's another way we can eliminate silos and keep the entire organization on the same page.

Get the bird's-eye view of what you can do with various integrations and open source magic for Mattermost.

Use documented, repeatable, and shared processes

Mattermost CEO Ian Tien recently wrote about how DevOps lost its way -- and how to bring it back, talking about the need for visibility and centralized, simplified interfaces. For example, DevOps best practices often rely on checklists. A team starting from scratch might run through a process, write down what they did, run it again, update the documentation, then continue to use that same process and iterate. That team will need to provide updates and visibility into the process when it's underway and afterwards in reports, and the checklist needs to be accessible and easy to update as the team learns and improves over time. Unfortunately, teams often rely on text documents, spreadsheets, wikis, and other tools that are both siloed and difficult to maintain.



Mattermost Playbooks are templated, collaborative checklists with built-in automation and integration with third-party tools that a team can follow over and over. Every step and required action of a process is documented and prescribed by the playbook for consistency and repeatability. A `run` of a Playbook includes a checklist with task assignments, dependencies, notifications, and integrated automations, as well as an associated Mattermost channel. Once the run has finished, Playbooks offers integrated retrospectives, reports and a timeline of everything that happened.

Playbooks support transparency within and between teams because stakeholders can observe the channel conversation side-by-side with the checklist while the run is in progress, or review the timeline and retrospective afterwards. They also support short feedback cycles, continuous improvement, and visibility of work, all components of DevOps best practices. For example, our fictional company "Acme Co" uses Playbooks for incident response.

Possibly the most important feature of a good checklist is that it enables and supports any team member using it. Imagine your on-call engineer receiving a page at 3 a.m. local time and needing to perform all the necessary steps to head off an outage, alert others, start troubleshooting, and make sure everything they do is tracked. That's a lot to ask of someone who was asleep five minutes ago. A Playbook is there to enable that engineer for success, as well as providing a clear timeline and picture of what happened so far when the next member of the response team joins.

In summary, Playbooks make life easier for everyone from release managers to on-call SREs by documenting and enforcing the use of an agreed-on process:

1. Use it for incident response
2. Use it for software releases
3. Use it for testing and QA procedures
4. Finally, use it to inform retrospectives and postmortems, which are important because that's where teammates can feel safe in sharing their opinions and experiences. Everyone is invited to contribute in an open format that encourages honest feedback and constructive suggestions for improvement. Those suggestions can then be worked back into the playbook to continuously improve the process and the team. Kaizen!

Integrate CI/CD pipelines

Continuous integration and continuous delivery (CI/CD) are considered best practices because releasing small pieces of work regularly reduces firefighting and increases value for the end user. Automations like build processes greatly reduce the risk of introducing errors during deployment, support security best practices, and help the entire team stay on track throughout the build and release process. However, working with additional systems adds to the volume of tools a team needs to manage and monitor and leads to increased context switching and decreased productive time.

Here are three ways to integrate your CI/CD pipelines so that you don't need to leave Mattermost to trigger, monitor, and interact with workflows:

1. CircleCI - Connect your builds directly into Mattermost, then use integrated commands to monitor pipelines and trigger workflows. Receive notifications from CircleCI in Mattermost channels, and interact with the plugin using `/circleci` slash commands.

2. Jenkins - Similar to the CircleCI plugin, the Jenkins plugin allows you to subscribe to build notifications in Mattermost Channels, start jobs, get logs, restart servers, and lots more. You may also want the Jenkins post-build action plugin to send webhook notifications.

3. GitLab CI/CD - Include notifications about your GitLab pipelines using the Mattermost GitLab plugin to subscribe. This plugin includes a host of GitLab integration features for two-way communication with Mattermost. Run `/gitlab help` in Mattermost for more details.

4. Find more CI/CD plugins for Mattermost in the Marketplace

Implement Your Best Practices With Mattermost

Following best practices doesn't need to be complicated. The keys are to enable as many people as possible and encourage broad participation to increase the visibility of the excellent work your team is capable of. To do so, remember to:

Communicate clearly and constructively

Have a single source of truth

Follow the agreed on process without barriers

Update and share resources

To learn more about how Mattermost works as a central operational hub for technical teams, try Mattermost today and reach out on the community server to continue the conversation!

Mattermost.com

🎉 9