10

WRITE A TEAM HANDBOOK

On remote teams, clear documentation is often the most direct way to share information at scale. Yet, many teams avoid this until the point that a lack of documentation starts hurting. Some of this stems from a misconception about agile software development. People believe that if you're working on an agile project, documentation is not necessary.

Martin Fowler says this:

> Agile methods downplay documentation from the observation that a large part of documentation effort is wasted. Documentation, however, becomes more important with offshore development since face-to-face communication is reduced.... As well as documents, you also have more need for more active collaboration tools: wikis, issue tracking tools and the like.

Across the IT industry, timely delivery matters. To support timely delivery, you need easy access to information. A team handbook catalogs the most important pieces of information related to a team.

DISTRIBUTED TEAMS NEED A SINGLE SOURCE OF TRUTH

A lack of documentation leads to communication overhead. There are too many meetings, closed email threads, and instant messaging discussions to disseminate information. This makes repeated activities, such as onboarding, inefficient. Knowledge sharing is slow, information moves in bits and bobs, and those inefficiencies cause interruptions galore.

As GitLab rightly says,

> As a team scales, the need for documentation increases in parallel with the cost of not doing it.

Writing a handbook that contains stable documentation about the team's work is a better alternative. Daunting as it may seem, a handbook-first approach has many advantages. Not only do you maintain a single source of truth for the team's knowledge; you also reduce the chaos and confusion from inefficient communication. In this chapter, I'll walk you through some ideas about what to include in such a resource and how you can create and maintain it.

BY DEFAULT, START WITH YOUR TEAM

The first question to answer is about the scope of your handbook. Who is it for? All-remote organizations such as GitLab document everything about their company in a handbook. Changing your whole company, however, will be hard even if you're influential. So, I suggest that you focus on your immediate team. Makers will experience immediate benefits from an asynchronous way of working, so it makes sense to start documenting your practices and team information with your immediate colleagues in mind.

The good news is that nothing succeeds like success itself. If you can implement this way of working successfully with your team, your handbook could become a blueprint for other teams in the organization.

Team Handbook vs. Company Handbook

In an ideal world, your team's handbook will sit on the same system as the company handbook. Several companies achieve this using systems such as Confluence, SharePoint, or Mediawiki, which allow different groups to organize their content into spaces or team sites.

However, it may not always be possible to set up a universal system such as this. For example, in professional services firms, team documentation usually rests in client-approved systems. In other kinds of firms, there may not be an org-wide system available yet. In such situations, teams must set up their handbook using a tool they agree on and can procure. It helps to also onboard adjacent teams to this tool so sharing information becomes easier.

Once you've had some success with your team's handbook, you can make a case for more org-wide tools if that makes sense for your business.

WHAT GOES INTO A HANDBOOK?

Imagine your dev team as a mini company. There are two critical parts of knowledge relevant to your work.

- The organizational *context* that lays down *how you work* together
- The *content* of *what you work on* together

You can use a similar model to structure your handbook. Figure 10.1 helps you visualize this content architecture. Let's dive into a little detail.

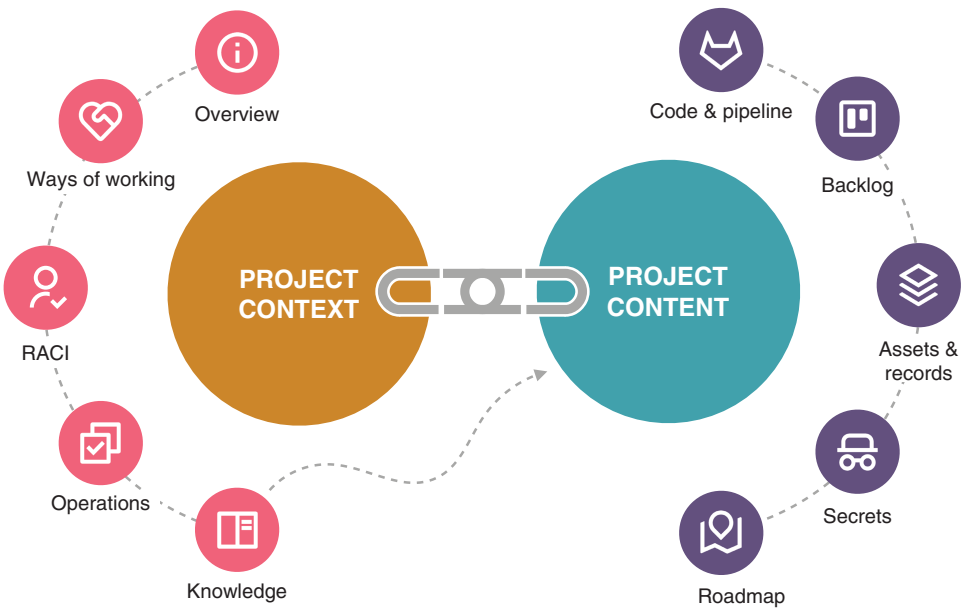

Figure 10.1 The structure of a project handbook.

Project Context

This part of your handbook derives from several topics we've addressed so far:

- **Overview.** The problem you're solving, with its industry context, stakeholder bios, and any background information that's useful for the team, goes here.
- **Ways of working.** Team values, guidelines, work agreements, and communication protocols and response times go here.
- **Responsible, accountable, consulted, informed (RACI).** As teams get larger and distributed, everyone can't possibly do everything. Clear roles and responsibilities on projects help avoid confusion. RACI charts are a succinct way to describe these roles and responsibilities. The chart clarifies roles using four labels.

- **Responsible (R):** The people who do the work
- **Accountable (A):** The person or group that all stakeholders will hold accountable for the task or deliverable
- **Consulted(C):** The people who give advice or input for the work
- **Informed (I):** The people who need to know what is happening with the work

Your RACI chart can help broadcast who does what and who they must collaborate with. I usually make it using a simple table with outputs in one column and a column each for the different roles on my team. The intersecting boxes describe how each role participates in delivering that output. Table 10.1 is an example RACI chart for a small development team. Your chart could look quite different based on your project context.

Table 10.1 An Example RACI Matrix

Outcome/ Output	Product Manager	Engineering Lead	Developers	Quality Assurance	Designer
User research	A	I	I	I	R
Product road map	RA	I	I	I	C
System requirements	RA	C	I	I	C
Site blueprint	C	I	I	I	RA
Design system	C	I	I	I	RA
Wireframes and mockups	C	I	I	I	RA
Architecture	I	RA	R	C	O
CI/CD and environments	I	RA	R	C	I
Code quality	I	A	R	I	I
Unit testing	I	A	R	I	I
Integration and automation tests	I	C	C	RA	I
UAT	C	I	I	RA	R
Team management and reporting	C	RA	I	C	I

- **Operations.** Depending on the team you are in, there will be operational information that everyone on the team should have. For example, people will need instructions on filling timecards; steps to get access to specific resources; or, for that matter, information about getting laptops serviced.
- **Knowledge.** Not only is this section useful for new members of the team, it's also important to existing team members. Everyone can't be familiar with every part of the project, and referencing the information in a structured fashion speeds up learning for the team. This section also links to your project content.

Project Content

Depending on the tools your team uses, these assets can exist across systems. Therefore, the knowledge section of your project context should provide a way to navigate across systems and provide your team with a bird's-eye view of your project's structure. The following are some of the most rudimentary components of your project content:

- **Codebases and pipelines.** Code is at the heart of your project. You should have an up-to-date catalog of your repositories, and it should be clear to your team how they can generate the application. In a Chapter 20, we'll discuss lightweight techniques to document your codebase.
- **Backlog.** The structured list of requirements your team is building should be here. It should be clear to the team how they can navigate this backlog and make sense of it.
- **Assets and records.** From design documents to prototypes to research or design sprint outputs—all these artifacts add up to the story of your product. Catalog these artifacts so your team can make sense of the part of the solution they're dealing with at any point in time. This is also the place where you should log all decision records and meeting notes.
- **Secrets.** You'll need access to SSH and API keys, passwords, and other sensitive information. It should be clear within the team how you will access these securely.
- **Roadmap.** If you've made commitments to stakeholders or the market about what functionality or capabilities you'll deliver in the immediate future, make these commitments visible to the team. Ideally, they should have provided input for this road map. Making the roadmap visible to the team allows everyone to own it.

Of course, you can think of other branches to this structure, and each level of this structure will have its own substructure as well. Much like someone must pave the

road before others travel on it, some people on the team must structure version 1 of the handbook so others can benefit from it. This list is a starting point for that first version.

TOOLS YOU CAN USE

On tech teams, tools are a topic of endless discussion. I always suggest using the tools that your team already has access to. Even if there's a more efficient tool out there, you'll face trouble if your company doesn't already use it.

- Access control will be problematic. It's hard to keep information secure, and you'll end up taking on too much risk of inadvertent intellectual property leakage.
- Onboarding people will be difficult because your teammates may not know about this new tool.
- Off-boarding people is also tough. If you don't remove people when they leave the team, you'll risk unauthorized access to content, code, and credentials.
- You'll have to maintain tools yourself and manage subscriptions and licenses at the team level, which is unnecessary overhead.

If your company doesn't provide you with a basic set of documentation tools, start with a work-around and make a case to get the right tools in place. The due diligence and sign-up may take time, but once the new tools are in place, you'll be able to translate the benefits outside the team as well. You don't really need fancy software. GitLab, Confluence, SharePoint, Notion, Almanac, Mediawiki—these are all good starting points.

RESOURCE

Simplify Documentation with Modern Tools

Modern tools can help automate some of your documentation efforts.

- Qatalog can help set up a structure for your team's workspace using a simple prompt.
- Scribe can help create effective, visual how-to guides in a short time.
- Glean can enhance the discoverability of your content with its AI-powered search.

START SMALL, OWN COLLECTIVELY, BE ITERATIVE

If you look at the various elements of project context and content, you'll realize that putting together the first version of your handbook is about a week's effort. You don't have to be perfect. Just get it out there. Expect everyone in the team to update relevant sections of the handbook over time. I encourage my colleagues to build a habit of updating the handbook whenever they face a question and the answer isn't in there. Designate someone in the team as the handbook manager and, if it works for you, rotate the role. This will help build collective ownership, and people's trust in the system will improve as a consequence.

Let me also offer some advice on version control. Most documentation tools offer some level of versioning. Figure 10.2 illustrates versioning in Confluence. You can also "watch" pages for which you want the system to notify you, whenever there's a change. These features allow you to be a light touch with editorial control for your handbook.

- Make it easy for everyone to edit all pages. You can limit editing access for some pages only if the content is sensitive and if the cost of misinformation is high. Such content should be the exception, not the norm.
- Leave comments open on pages where you've restricted editing. This way, the team can still suggest changes to the page.
- Encourage people to summarize their changes using an edit summary. Most version-controlled documentation platforms offer this functionality. If the platform doesn't allow such a summary, they can report their change as a comment on the page.

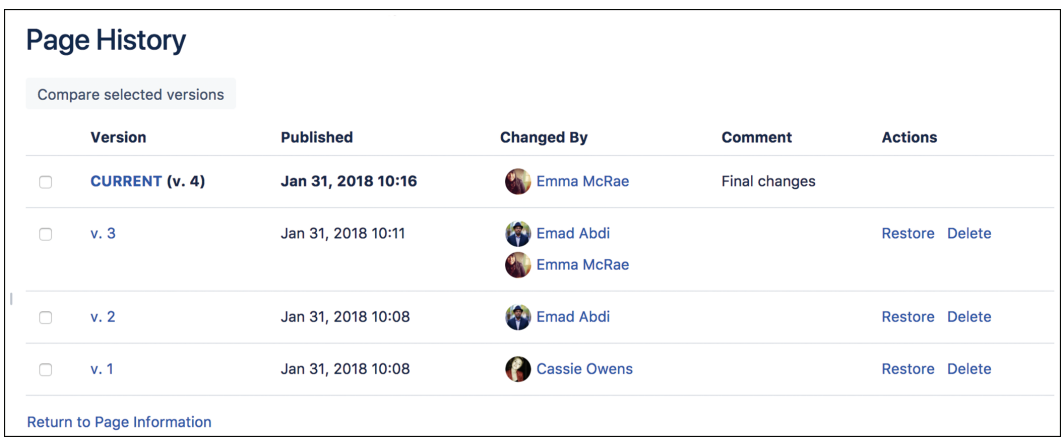

Figure 10.2 Version control on *Confluence*.

- Page owners can watch their pages if they want to stay on top of all changes. Comparing versions will help them easily spot the change.
- If you notice a change that you must revert, use the versioning system to do so. Be sure to share feedback with the contributor so they know why you reverted their change.

Tools such as GitLab enable everyone to make changes using a system of merge requests. However, for an internal team handbook, where trust levels are high and the risk of vandalism is close to zero, I suggest taking the more permissive approach that I described.

GO DEEP WITH YOUR DOCUMENTATION TRIGGERS

Your handbook will need constant updates if it is to be useful. This means keeping old documents up-to-date and adding new ones when necessary.

On teams where a writing culture doesn't exist, you may need to remind yourself of triggers when documentation can help. I use the acronym DEEP to remind my colleagues about the opportunity to write or to create a referenceable artifact. Here's what the acronym stands for:

- **Decisions.** Teams make many decisions over their lifetime. Each time you decide something, document it in a way that someone who wasn't involved in the decision can understand its rationale.
- **Events.** Meetings, workshops, town halls—they're all events. By documenting these events, you persist the outcomes and knowledge from these interactions.
- **Explanations.** Every project has a body of knowledge, be it about the domain it operates in or how part of the system works. We often repeat these explanations verbally to each other. A written reference is easier to share and to improve as a team.
- **Proposals.** In the lifetime of a project, the team and its stakeholders will share many ideas and plans. Whether we implement these or not, it helps to document the thought process and the details of such proposals. Not only does it foster decision hygiene, it also helps build the collective memory of the team.

Each time there's an opportunity to document something that corresponds to the DEEP acronym, we remind each other to write things up. On one of my teams,

we printed the DEEP acronym on coffee mugs to remind colleagues about these documentation triggers.

BEST PRACTICE

Learn to Respond with a Link

If you're writing things up regularly and referencing them in your team's handbook, you can avoid redundant conversations by encouraging people to read. Just share the link and let them self-service. This will also improve adoption of your handbook and your asynchronous knowledge resources. This isn't rude. It's efficient.

Here's an example of how I'd respond with a link:

> **Co-worker:** Hey Sumeet, can you talk me through the requirements for the notifications service? I wanted to figure out what endpoints we need to build out to support the new functionality.
>
> **Me:** Hey, mate! I already wrote up the requirements as part of the proposal ([Insert link here]). Check it out. You'll find explanatory wireframes in there too. If something is not clear, just drop a comment and I'll address it pronto.

If you can't respond with a link, then it's a trigger to write things up. Help your colleague out, create the document, and then get their feedback to check whether it answers their original question. That way, you can respond with a link in the future.

AIM FOR A SHARED REALITY

For a distributed team, a single source of truth, such as a handbook, helps you build a shared reality. Without it, your team will be like the blindfolded people and the elephant that you see in Figure 10.3. Everyone will build their own mental model of the project, but the team will be none the wiser. Instead, create the handbook, figure out a way for people to contribute to its upkeep, and watch the blindfolds disappear. In this chapter, we've discussed a few ways to begin this iterative journey.

Figure 10.3 Without a single source of truth, everyone has their own truths. (Source: mentalmind/Shutterstock)

CHAPTER SUMMARY

A team handbook helps make information about work explicit to existing and new team members. This is a critical asset for async-first work.

- Focus on writing things up for your team first. Don't worry about organizational knowledge at the start. If you're successful, your handbook can be a blueprint for the company.
- Divide your handbook information by project context and project content. *Context* lays down *how you work* together. *Content* describes *what you work on* together.
- Avoid being too fancy and use the tools your company already provides.
- Expect everyone to improve and update the handbook and designate editors on a rotational basis. Start small and iterate on the content and design.
- Use the DEEP acronym to remind each other about the triggers for documentation. Printing it on team merchandise is an effective way to make the triggers visible for remote colleagues.

Easily retrievable team knowledge will reduce communication churn on the team. You'll have fewer meetings and fewer interruptions from email and instant messaging. Speaking of instant messaging, let's talk about that next.