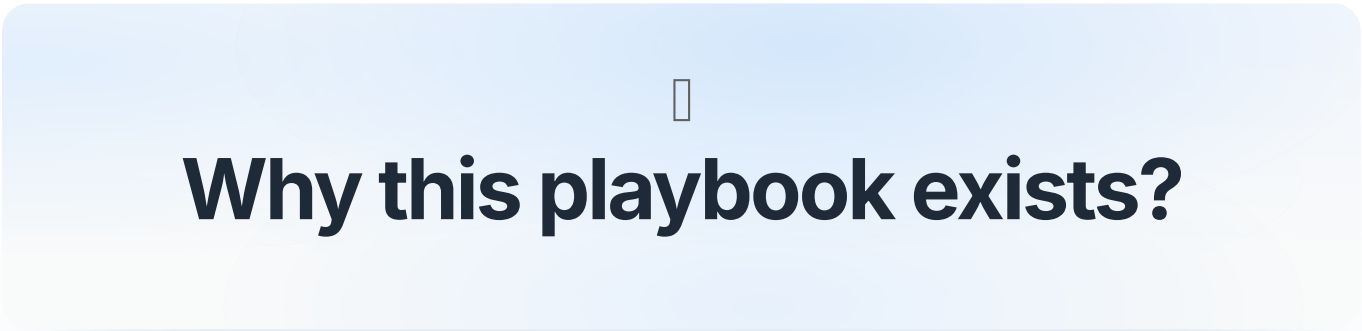
Technical Writing

1. Introduction to Technical Writing



This playbook is an introduction to technical writing. It's practical, straightforward, and just a touch more entertaining than the average style guide.

It's aimed at new technical writers but doesn't assume you've already memorized a dictionary. It should also come in handy if you're a part-time documentarian. That includes developers, business analysts, or anyone who's just been told, "Can you write this up quickly?" and knows that quick rarely means quick.

What Is Technical Writing, Anyway?

At its core level, technical writing is all about making complex information easier to understand. If this sounds vague, it's because technical writing is a vast domain and not that easy to describe in a sentence. Sometimes it's not even technical and doesn't include writing!

(That said - this playbook will focus on the written type of documentation, especially in the software world.)

Some examples of technical writing

- **The manual** that came with your washing machine
- **The Archbee documentation**
- **IKEA diagrams**
- A programming language **API reference**

- **Standard operating procedures** for servicing an airplane

Alien assembly instructions, by Ed Harrington

The day-to-day tasks of a technical writer are different from role to role, but technical writing in general has the following characteristics:

- **Clear and concise** - avoids unnecessary words and uses straightforward language.
- **Structured and organized** - uses headings, lists and sections for easy reading.
- **Objective and neutral** - focuses on facts rather than opinions or emotions.
- **Instructional** - often includes step-by-step guidance or explanations.
- **Visual elements** - may include diagrams, tables or charts to support understanding.

What *Isn't* Technical Writing?

Creative Writing

As is obvious from the name, creative writing is almost the opposite of technical writing.

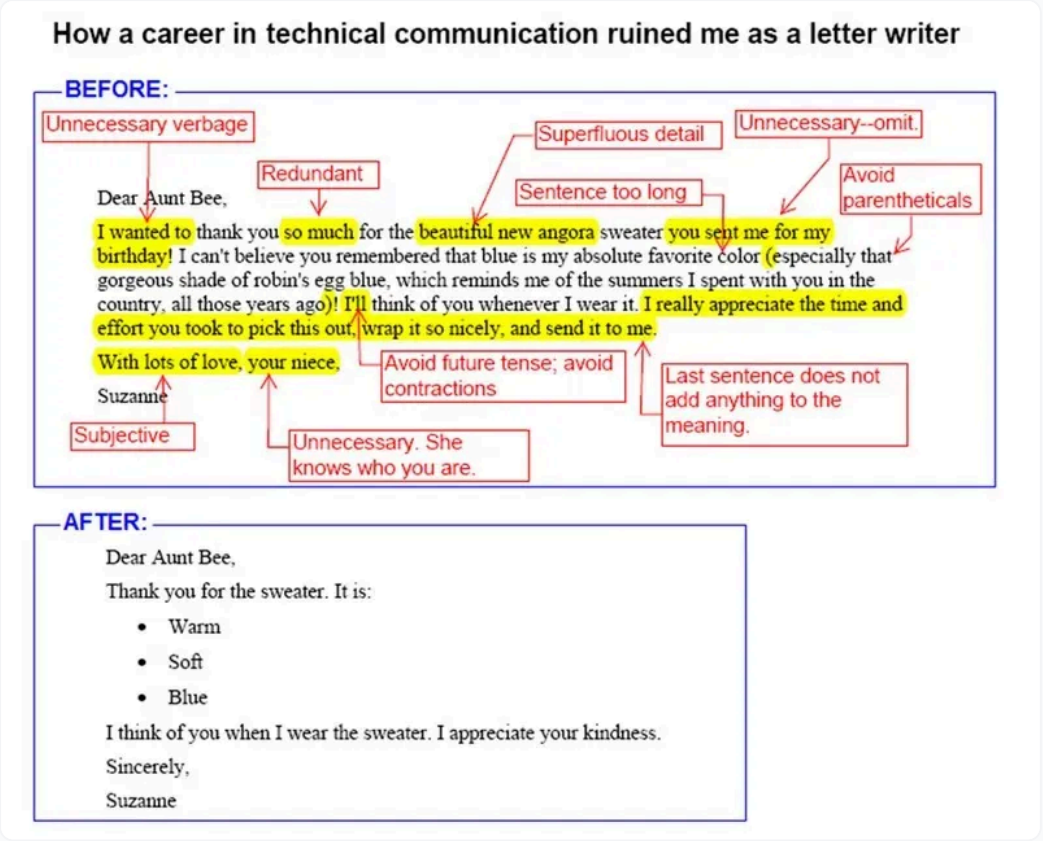
Creative writing

More artistic, aiming to entertain or evoke emotions and prioritizing style over clarity.

Technical writing

Factual and structured, aiming to instruct or explain.

Interview with Becky Chambers -- author of The Long Way to a Small, Angr...



When a technical writer tries to write a thank you note

Technical Content Writing

Technical writing and technical *content* writing are often confused one for the other, for obvious reasons. While they do have their similarities, the two terms cannot be used interchangeably.

Content writing

Aimed at a broad audience and is more focused on storytelling, SEO or marketing.

Technical writing

Targeted at the users of a product and focuses on helping them accomplish specific tasks.

That being said, experience in content writing can be useful for someone trying to transition into technical writing, since both fields involve simplifying complex information,

adherence to grammatical and stylistic standards, and deep research into technical topics.

Academic Writing

Writing is a large part of life in academia, but it follows different conventions than technical writing.

Research papers

Focus on analysis and argumentation, generally using long sentences, abstract language and dense jargon. To sound objective, academic writing uses passive voice.

Technical writing

Focuses on practical information, using clear language and simplified explanations. Since it's meant to actively instruct users, technical writing uses active voice.

Journalistic Writing

Good writing and adherence to a style guide is critical for journalism, but there are also significant differences compared to technical writing.

Journalism

Focuses on storytelling or current events, often using an engaging and dynamic style to capture readers' attention.

Technical writing

Uses neutral language to guide the users through a process, without eliciting an emotional response.

Transferrable Skills

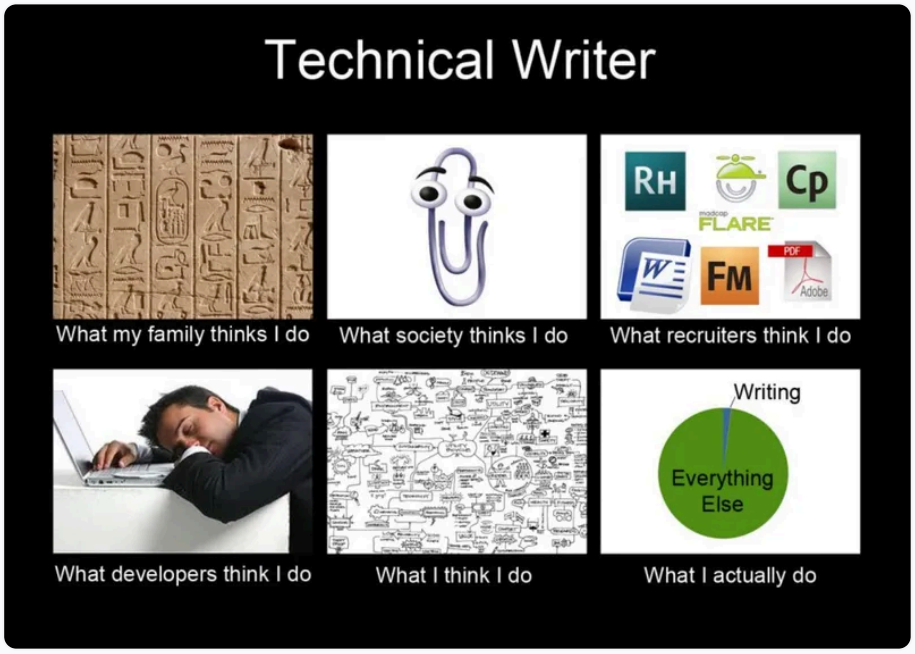
The previous section focused on the ways other types of writing are *not* technical writing. However, this does not mean that journalists or content writers don't have a lot of experience that can be leveraged into technical writing!

The **Essential Skills for Technical Writing** section focuses on the skills that a successful technical writer needs.

1.1. Essential Skills for Technical Writing

Creating good documentation
isn't just for
professional technical writers.

Anyone who needs to clearly explain information can focus on improving these specific skills.



A vintage meme that is still applicable

Writing and Editing

Most documentation is consumed in written form, so writing and editing skills are at the top of the list.

- **Ability to translate complex technical concepts into simple, understandable language**
  This is the core of the job and the main value a technical writer adds to a team.

- **Ability to write clearly, concisely, and accurately**
  "Omit needless words" is a technical writing mantra and helps users accomplish their goals without interruptions.

- **Strong command of grammar and punctuation**
  Proper grammar ensures that the explanations are clear and cannot be misinterpreted.

- **Editing skills**
  Self-editing skills are particularly important because technical editors are rare, and sometimes there's not even time for a peer review.

- **Adherence to style guides and branding guidelines**
  A consistent writing style and appearance ensure that the documentation looks professional.

Research and Information Gathering

One thing that most technical writers agree on is that, despite the name of the job, the bulk of the work is actually the research.

- **Independent research**
  Whether it's understanding a software application or becoming familiar with a particular industry, technical writers must be excellent at learning new things. Curiosity is an essential trait.

- **Ability to interview subject matter experts (SMEs)**
  Very often, the details of a new feature are only available in someone's head. It's critical to understand the information gaps and ask the right questions to fill them.

- **Audience analysis**
  You can't write documentation if you don't know who you are writing it *for.*

Collaboration and Teamwork

Technical writing tends to attract introverts due to its apparent focus on solo work, but this is not true. Writers never work in a vacuum and very often *depend* on help from others.

- **Working with people across different departments**
  Writers need communication skills and tact when requesting information from engineers or business analysts. Documentation is, unfortunately, often seen as an afterthought, so writers need to be flexible and adapt to SMEs' availability and communication preferences.

- **Collaborating with other technical writers**
  Writers who are part of a team may be involved in training coworkers, peer reviewing their work, or helping out on their tasks.

Documentation and Publishing Tools

It's not a coincidence that this is at the end of the list. Tools are important, but they should be there to support the writing, not the other way around.

Ability to learn and adapt to new tools

It's impossible to know all the tools on the market, but a good writer has to be able to learn them.

Ability to analyze tools suitability

As experts on technical documentation, writers may be asked to research and propose tools, especially when working in a startup environment.

Design and visuals

Sometimes it's not only about the words - designing diagrams, charts and illustrations are useful skills to have.

1.2. Types of Documentation

Some writers focus on user guides for customers, while others handle internal documents. Each type has a different audience and purpose, so a writer's work depends on their role and industry.

External Documentation

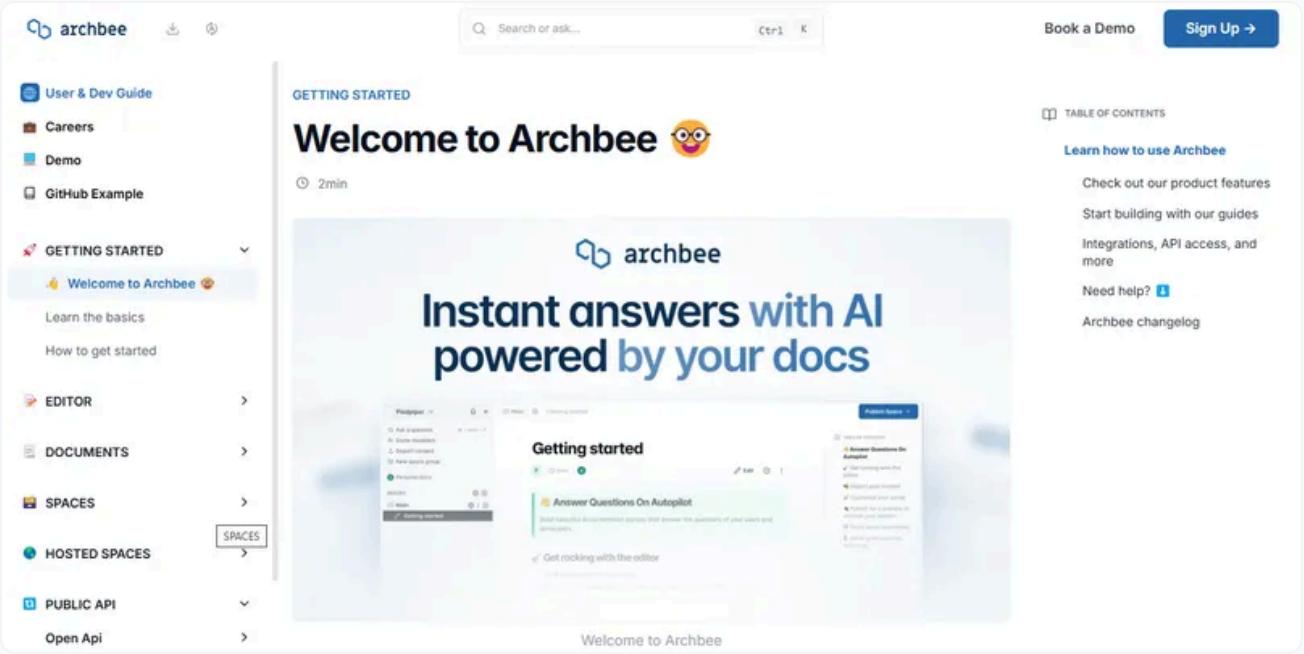
Like the name says, **external** (or **client-facing**) **documentation** is aimed at customers of a product or the general public.

The tone is generally more formal than in internal documentation, and the content focuses on usability and practical applications rather than implementation details.

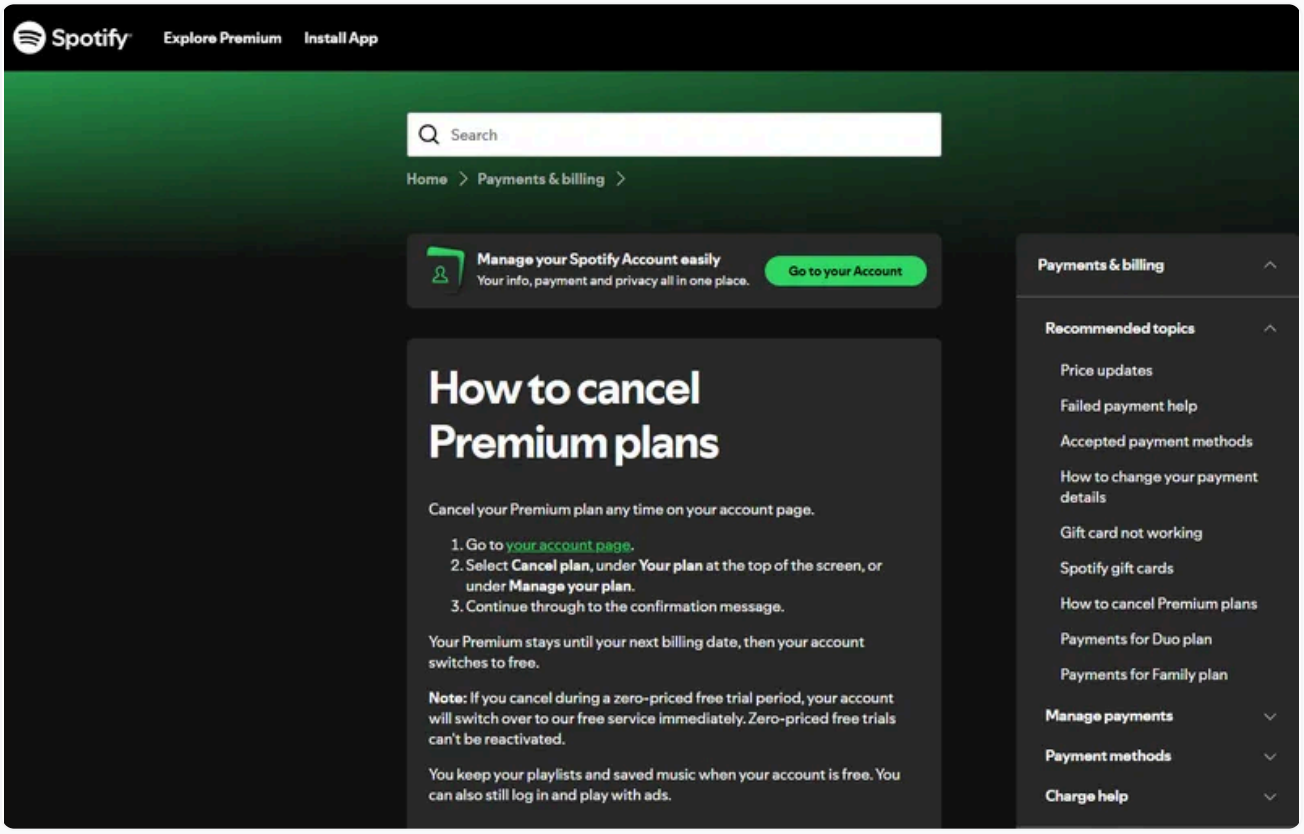
As the target audience is outside the company, this kind of documentation does not disclose sensitive or proprietary details and must comply with various legal and regulatory requirements to protect the company and the users.

Some examples of public knowledge portals:

- **Archbee documentation** (of course) - covers using and managing Archbee and using its public API
- **Stripe Payments API documentation** - explains how to integrate Stripe's Payment API into your products
- **Epson printer documentation** - user guides and data sheets for an Epson printer (in PDF format - a documentation portal is not necessarily HTML-only!)
- **Mailchimp FAQ** - frequently asked questions about Mailchimp
- **Spotify Knowledgebase** - support articles for Spotify users



Welcome to Archbee



Spotify Knowledgebase

The following list includes some examples of external documentation:

- **Installation and configuration guides** - instructions for setting up a product, software, or hardware, and making it available for end-users.

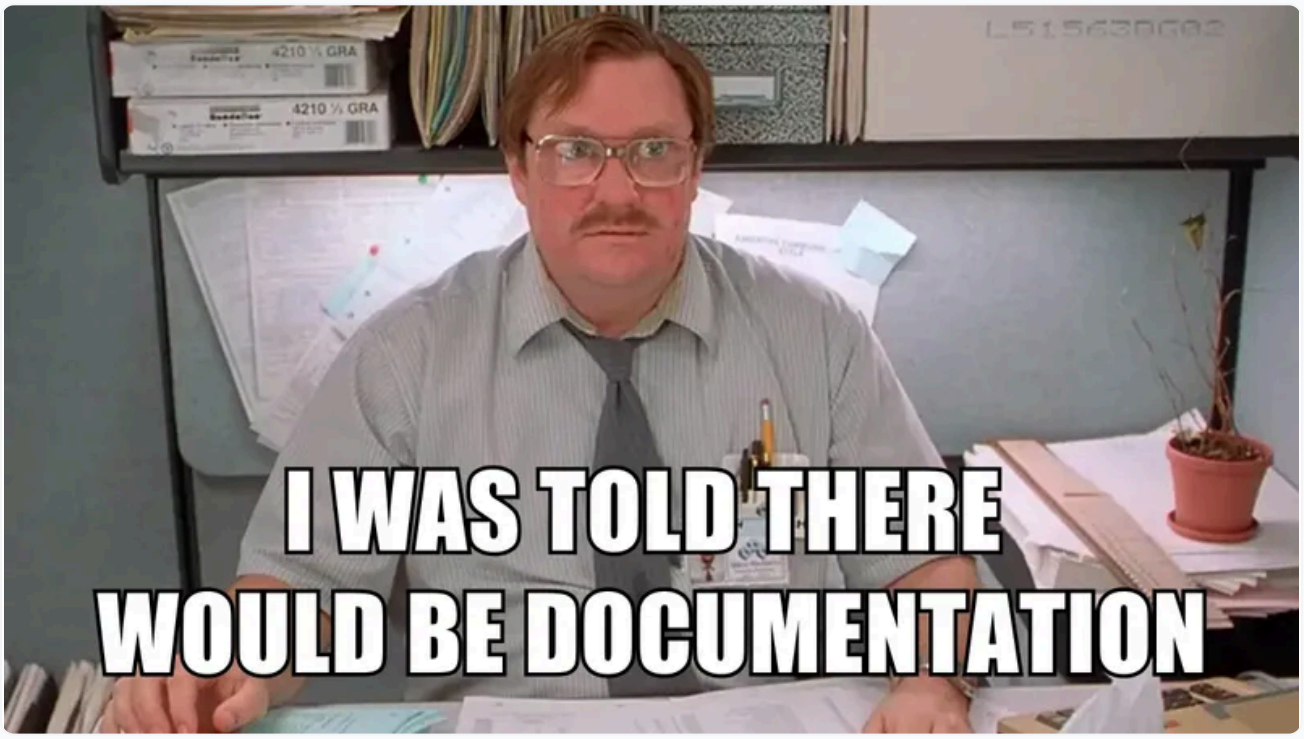
- **Administration guides** - documentation for maintaining the product post-installation.
- **User guides** - end-user instructions for using a product, software, or hardware.
- **API documentation** - instructions for developers on how to use an API.
- **Quick start guides** - concise, easy-to-follow instructions for getting started with a product.
- **Integration guides** - instructions for connecting software or hardware with other systems or components.
- **FAQs (frequently asked questions)** - answers to common user queries in a structured format.
- **Troubleshooting guides** - solutions to common issues users may encounter.
- **Knowledge base articles** - concise documents that help users self-solve problems.
- **Release notes** - short summaries of bug fixes and new features added in a product release.
- **Product specifications** - detailed technical information about the hardware's features, dimensions, and performance.
- **Warranty and support documents** - information on warranty coverage and service policies for a physical product.
- **Safety and compliance documents** - warnings, precautions, and regulatory compliance details for safe usage.
- **Datasheets** - technical summaries of a product's specifications, including electrical, mechanical, and thermal properties.
- **Service manuals** - technical documents for technicians repairing and maintaining hardware.
- **Regulatory compliance documents** - certifications and compliance details ensuring the product meets industry standards.

> Some types of documents, such as knowledge articles or release notes, can be both internal and external. For example, a company may publish a full changelog internally, but select specific issues to highlight in the public release notes.

Internal Documentation

Internal documentation is only available to employees of the company, so it is more relaxed in tone and does not need to comply with the same requirements as external documentation.

Often, internal documentation includes detailed technical or proprietary information and may describe processes specific to the company.



When you join a new company and there's no onboarding documentation

The following list includes some examples of internal documentation:

- **SOPs (standard operating procedures)** - step-by-step guides for internal workflows and processes.
- **Developer guidelines** - instructions on coding standards, infrastructure setup, and development workflows.

- **Playbooks** - structured manuals for handling specific tasks or incidents.
- **Onboarding guides** - training materials to help new employees learn tools and workflows specific to their roles.
- **Architecture diagrams** - explanations of the interactions between system components.
- **Training guides** - training materials for internal skill development.
- **Product requirement documents (PRDs) or business requirement documents (BRDs)** - detailed documentation outlining product features that should be implemented.
- **Technical design documents (TDDs)** - detailed technical documentation for product development.
- **Style guides & branding guidelines** - rules for consistent writing, design, and messaging.

> Many of these documents are often written by non-technical writers, such as developers, architects, or business analysts.

2. Information Architecture

In the context of technical writing, information architecture refers to the way the documentation is organized, structured, and labeled so that readers can navigate it efficiently and get the guidance they need.

> It's hard to say whether you should start a documentation project by thinking of your information architecture or your content strategy.
>
> In real life, you will probably switch from one to the other, and back again. Feel free to do the same in this playbook; the sections are meant to be readable independent from one another.

Structuring the Content

Topic-based authoring and book-based authoring have different audiences and require different approaches, so it's important to understand their particularities before making a choice.

Book vs Topic Structure

At a structural level, there are only two types of documentation:

Book-style

In this case, the user is expected to read through the material sequentially, from the start to the end, chapter by chapter. This type of documentation is generally delivered as physical, printed books or as PDFs.

The content is organized under the assumption that the reader is already familiar with the previous chapters, so jumping from one section to the other is not recommended. This makes them more suitable for products that fit a structured learning experience and where users do not need to quickly solve a specific problem.

That said, it doesn't mean that these user guides must be read from page 1 to page 100 in one go. Readers can navigate using the table of contents, indexes and references - and of course `Ctrl` + `F` is always available for digital formats.

Topic-based

Here, each page is a self-contained topic that can be read more or less out of sequence. This type of documentation is generally delivered in the form of an online help system.

> **Every Page is Page One**, by Mark Baker, is the most well-known book about topic-based authoring.

Traditional, book-type manuals have fallen out of style these days, for good reason: modern users don't have time to read hundreds of pages to get their answers, and the search functionality in a PDF is inferior to that of a search engine.

However, book-style documentation is not completely gone; it is commonly used in regulated industries (such as aerospace), as well as for physical products (such as industrial equipment).

The rest of this page will focus on topic-based structure.

Types of Topics

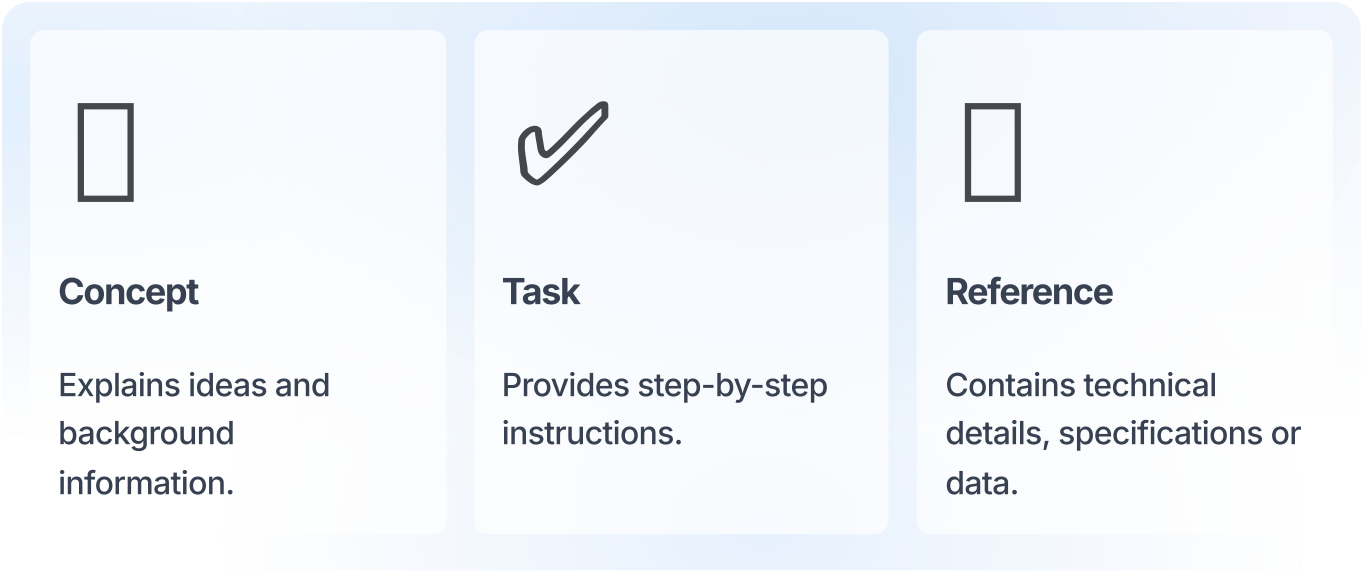
When planning out a topic-based documentation set, the first thing to decide content-wise is what types of topics you will create. Having a standardized set of topic types is essential for making your documentation easy to follow.

Two popular frameworks are described below; you can either adopt their principles as defined or modify them to suit your purposes.

DITA

Topic-based authoring was popularized in the 2000s through **Darwin Information Typing Architecture (DITA)**, a framework developed by IBM and later standardized by **OASIS**.

DITA is an XML-based standard for modular documentation that structures content into three main topic types:



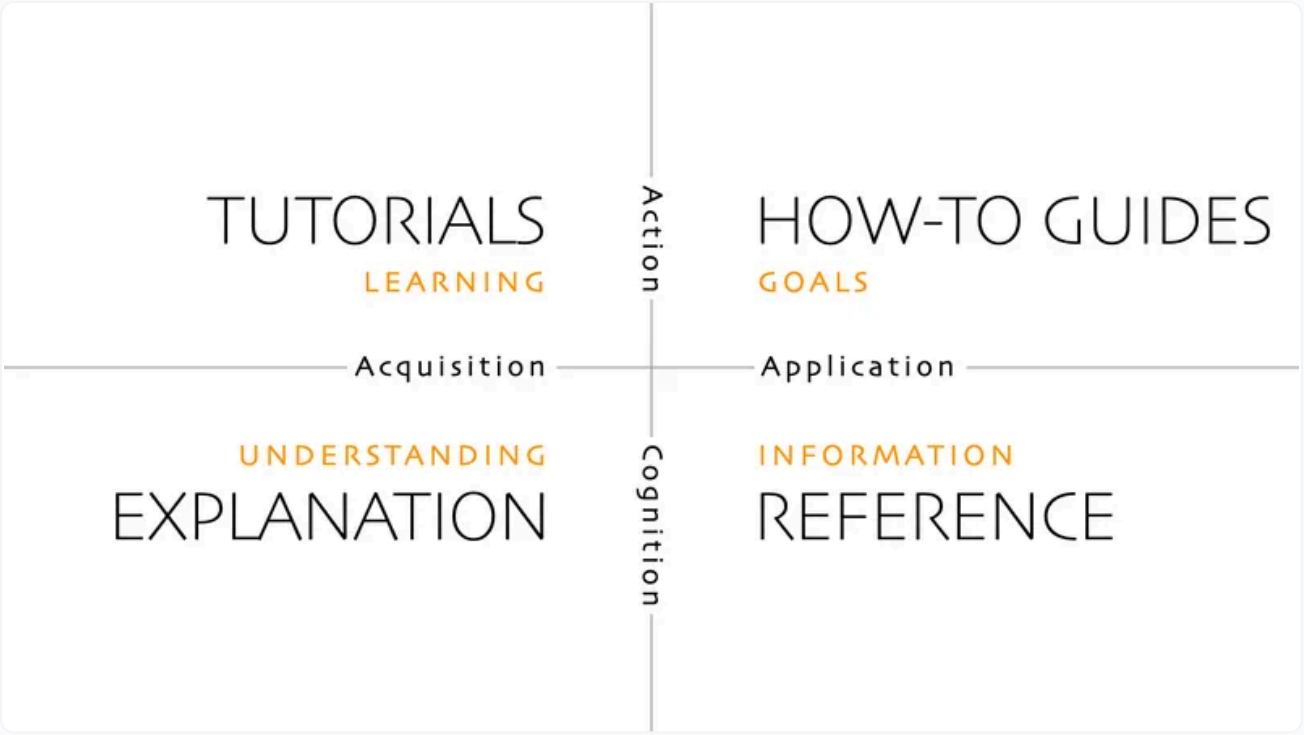
Of course, these three topic types do not fit every documentation set, but DITA allows specialization, which means that you can create new, customized information types based on existing DITA structures.

> It is important to note that DITA encompasses two related, but different aspects:
>
> - The conceptual definitions of the basic topic types
> - The strict XML schemas used to technically validate the structure of the topics
>
> You don't have to use DITA-the-XML-standard in order to use DITA-inspired-topic-types. In fact, many companies have defined their own topic types starting from the DITA ones, but write in Markdown or HTML rather than XML.

Diátaxis

In the late 2010s, Daniele Procida developed **Diátaxis**, a framework that defines four topic types:

- **Tutorial** - teaches users through practical exercises.
- **How-to guide** - provides instructions for doing a task.
- **Reference** - contains technical details.
- **Explanation** - provides background and puts things in the bigger picture.



https://diataxis.fr/start-here/

Because Diátaxis is just a concept, not an XML schema, it's easier to use in practice: it does not depend on a specific format or tooling and it can be interpreted as needed for each documentation project.

> For a comparison between DITA and Diátaxis (and other frameworks like Information Mapping and The Good Docs Project), see **this post by Tom Johnson**.

Designing the Navigation

Especially in large documentation sets, the navigation system is essential in helping users find the information they need.

> As the majority of documentation is available nowadays in the form of an HTML-based help system, this is what this section focuses on.

These elements are typically included in modern online documentation:

Table of contents (TOC)

- Hierarchical, shows the structure of the documentation and helps users get a high-level view of the information available.
- Usually placed on the left side of the browser window.
- Sometimes a topic-level TOC is also included, usually on the right side.
- Should not have too many levels, otherwise it gets unwieldy.

Search

- A search bar is a must-have, in a visible location accessible from all pages.
- Can be enhanced with features like filters and suggestions.
- An AI chatbot can be used as an addition to the traditional search.
  It is not a good idea to replace the search *completely*, since AI agents serve a different use case. For example, if you are searching for an exact string, a search bar is a better choice than a chatbot.

Breadcrumbs

- Usually placed at the top of each topic.
- Help users understand where they are within the larger structure of the help system.

Responsive layout

- Ensures the documentation website is usable on all devices (desktop, tablet, mobile).

Contextual navigation

- "Related topics" sections or "next/previous" buttons to guide users to related content.
- Important, especially in complex help systems.

> Consistency is important in navigation too. Make sure to use predictable icons, color schemes and labels to help users orient themselves in the documentation.

3. Content Strategy

Starting out a documentation project can be overwhelming, but the process can be broken down into:

Audience Analysis

Defining the audience needs.

Research

Understanding the to

Writing

Writing the actual documentation, while following a few basic principles.

Reviewing

Checking the content of the documentation to make sure it's accurate.

Testing

Checking the technical integrity of the documentation.

Delivering

Making the documentation available to end-users.

In a documentation context, a project doesn't have to be something big, like an entire help system. The principles apply even when you are just making small updates to a legacy documentation set.

The next sections
dive deeper
into each topic.

3.1. Audience Analysis

Before writing any document,

you need to know who you are writing it for.

Your readers may be experts in the field or they may be complete novices; without proper audience analysis, documents risk being unclear, misinterpreted or ineffective.

Audience analysis should include the following information about the users:

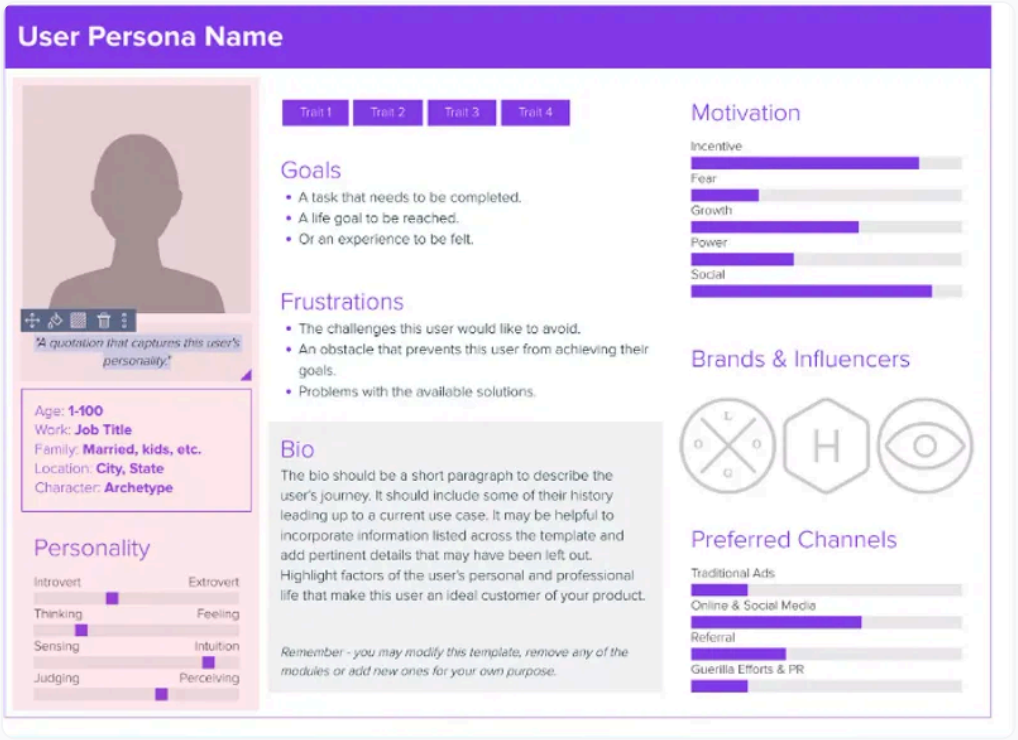
- Job role
- Experience level
- Usage context
- Learning style and preferred level of detail
- Tasks they want to accomplish
- Challenges they face

Creating Personas

Based on the user information collected as explained in **Audience Analysis**, you can segment your audience into groups and create user personas - fictional representations of typical users, based on real data and common characteristics.

The personas can then inform your documentation strategy - for example, you may need a very user-friendly getting started section for novice users coupled with a detailed API reference for developers who will implement an integration.

For more details about creating personas for documentation, see **this article on UXmatters**.



Source: https://xtensio.com/how-to-create-a-persona/

> Even if you don't create written, detailed personas, it is still useful to have a few basic types of audience in mind when creating the documentation.
>
> For example, a simple split could be between functional documentation (aimed at the end-users of an application) and technical documentation (aimed at the IT technicians supporting the end-users).

Gathering Audience Insights

To understand what your audience needs, you can collect information directly or indirectly:

- **Direct user interaction** - Directly ask users about their background, preferences, knowledge levels, and expectations.
- **User feedback** - Analyze bug reports and use feedback tools and analytics to track user reactions and assess content effectiveness.

- **Collaboration with subject matter experts** - Gain knowledge from product managers or customer success representatives who have constant interaction with the end-users.

Direct User Interaction

Surveys and Interviews

The best way to get information about user preferences is to ask them directly, either in writing or in person. An in-person or Zoom interview gives better opportunities for asking follow-up questions, but it's also more time-consuming for both the interviewer and the interviewee, so a good balance is essential. To optimize resources, you can send a survey to your entire user base, then select a representative sample to talk to directly.

The following table offers some suggestions for questions you can ask in a survey.

The following table offers some suggestions for questions you can ask in a survey.

Category	Questions
General User Information	• What is your job role or background? • How frequently do you use this documentation? • What is your level of expertise with this topic? • What is your learning style?
Usefulness & Effectiveness	• Did the documentation help you accomplish your task? • Did you find the information easy to understand? • Is the information technically accurate and up-to-date? If not, could you point out which areas need work? • Do you prefer more detailed technical explanations or high-level overviews? • Were the examples or visuals sufficient?
Findability	• How do you locate the information you need? Do you use the search engine, the chatbot, the navigation pane? • Is the organization of the content intuitive? • How long did it take you to find the information you needed?
Accessibility & Readability	• Did you experience any difficulties with font size, contrast or layout? • Would you prefer more visual aids such as screenshots, flowcharts or tables? • Is the document easy to read on different devices (desktop, mobile, tablet)? • Do you use a screen reader or other assistive technology to access documentation? If so, does the documentation support it well?

Usability Testing

Generally speaking, usability testing for documentation implies asking users to accomplish a task with the help of documentation.

You can use a more general approach by asking people to use the documentation as they normally would or you can run a more detailed test by having them focus only on the table of contents or the search function.

By observing users interacting directly with the documentation, you can:

- Evaluate the clarity of the content
- Identify usability issues
- Understand how users search for information
- Assess readability

For more information about usability testing, see **this article on the Nielsen Norman website**.

User Feedback

User feedback can highlight documentation gaps and usability issues and can help drive prioritization. You can gather feedback either directly, by embedding feedback mechanisms on the documentation pages, or indirectly, by analyzing support cases and analytics.

Embedded Feedback Forms

Feedback tools in documentation can range from the simple to the complex:

- A `mailto:` link that sends an email to the documentation team
- Thumbs up/down or star ratings that indicate if an article was useful or not
- Feedback forms that encourage users to leave written feedback

The usefulness of such features is directly tied to the management strategy behind them. Best practices include:

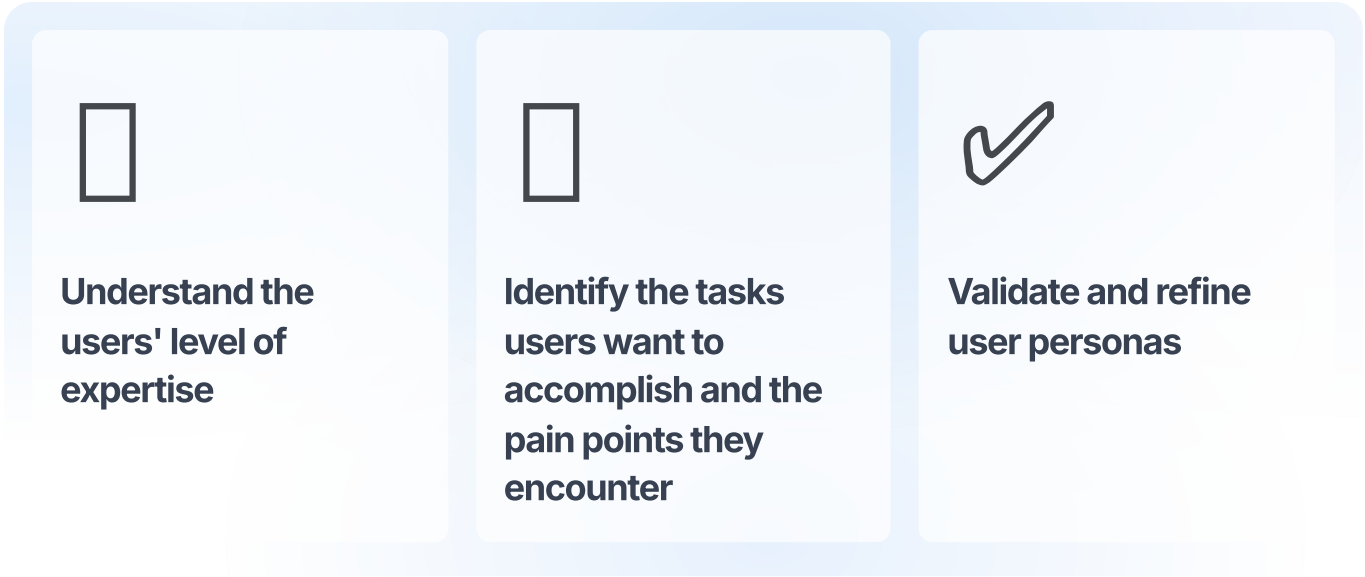
- **Assign responsibility** for reviewing the data and tracking patterns.
- **Store the feedback** in a database (even something simple, like an Excel file).
- **Filter out** non-documentation-related reports.
- **Use structured feedback forms** to capture specific issues rather than vague reactions.
- **Implement a strategy** for communicating the results of the feedback back to the end-users.

> Documentation teams often **jump to implementing feedback** mechanisms with unrealistic expectations.
>
> The reality is that the vast majority of feedback will be **either too vague** or **not really about the documentation**.
>
> Make sure to weigh the **pros** and **cons**; if the cost of implementing and managing the feedback form is too large, maybe it's better to use other ways of hearing from your users.

Support Team Insights

If a company has a customer support or customer success team, there is probably a lot of data available that can be used in audience analysis.

Support inquiries can help technical writers to:



Analytics

Analytics tools, such as Google Analytics, can be easily implemented in online documentation systems.

Once you have enough data, you can:

- **Analyze page views** to see which sections of the documentation are most accessed. You can then prioritize these pages when it comes to improvement initiatives.
- **Track time spent on a page** to understand how users engage with the content, highlighting areas that might be confusing or require more in-depth explanations.
- **Monitor user search queries** to see when they don't find the information they need, which can help you make a list of topics that need to be added.
- **Review drop-off points** to analyze where users leave the documentation, suggesting areas where the content or navigation might need simplification or clearer instructions.

Collaboration with Subject Matter Experts

Subject matter experts, especially ones who have direct client interaction such as product managers, can serve as a good source of guidance for documentation.

Customer-facing SMEs can help in audience analysis by:

- Clarifying user roles, needs and skill levels

- Helping writers understanding real-world use cases

Next Step

Once you have decided on your user personas, analyzed their requirements and got feedback from the users, you are ready to move on to the next step:

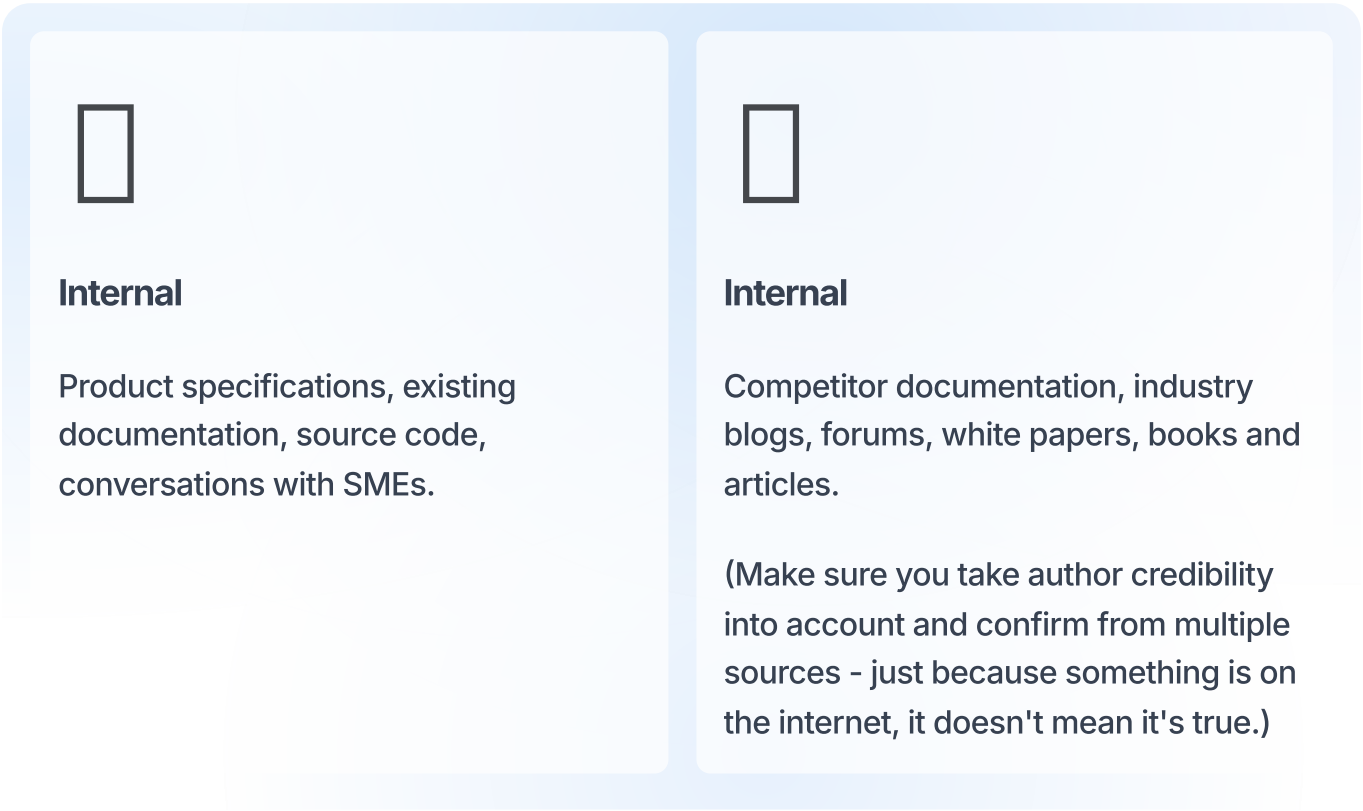
- **Research**, where you learn about the topic you are documenting.

3.2. Research

After identifying your audience, you have to find your sources of information and research the topic. (Like we already said, "technical writing" is mostly research, despite its name!)

Identifying Sources

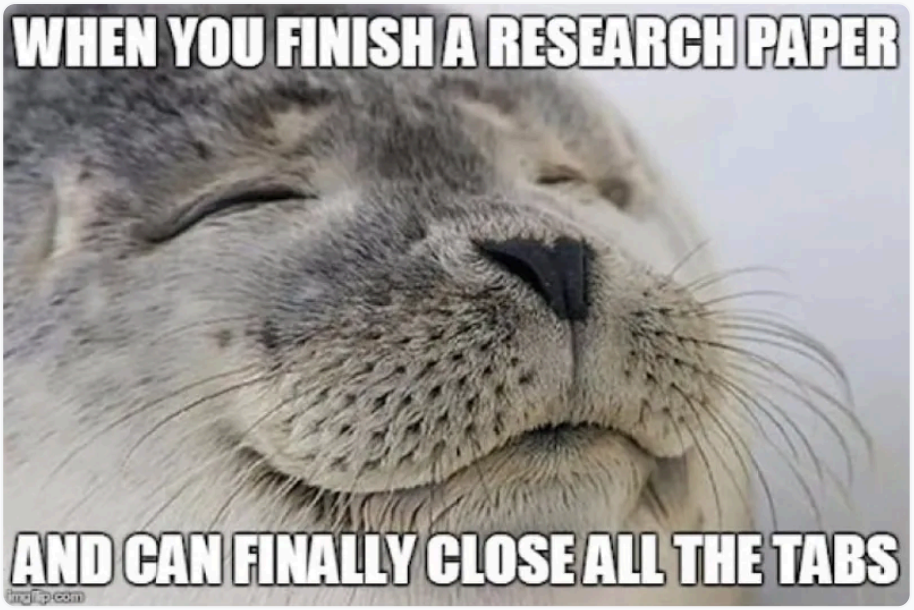
Good research is based on two types of sources:



External research sources help you quickly build a baseline understanding of a topic - how it works, what it's for and how users typically talk about it.

This broader context makes it easier to navigate **internal** sources more efficiently. You'll know what to look for in technical specs, what questions to ask SMEs and where gaps in the internal documentation might confuse users.

In short, external research gives you the map - internal sources give you the details.



Collaborating with SMEs

SMEs are often the most important source of information you will get. An SME can be virtually anyone involved with the topic you are documenting:

- **Engineers** or **developers** who built the feature
- **Product managers**, **product owners** or **business analysts** who shaped the requirements
- **QA analysts** who understand edge cases
- **Customer support representatives** who hear real user pain points
- **Sales or pre-sales representatives** who know how the features are described to end-users

Their contributions can be direct or indirect:

- **Direct**: face-to-face conversations, emails, online meetings and chat
- **Indirect**: customer/internal demos, specifications, user stories, meeting minutes, test cases

SMEs are busy people in general - follow these guidelines (in this order, as much as possible) to get the most out of their time.

1. **Do as much advance research as possible.**
   Make a list of the already-available materials, read through them and take notes.

2. **If you have access to the product itself, use it.**
   Apply what you learned through your research and see what gaps you find.

3. **Prepare questions for the SME in advance.**
   This will help you organize your thoughts and will make the discussion more structured.

4. **Ask for a meeting or a chat with the SME.**
   Be mindful of their schedule and use the time efficiently. If you set up a meeting with them, send the questions in the meeting invite or an email.

5. **Record the meeting, if needed.**
   Of course, always ask for consent first!
   A recording is useful especially if it includes a live demo of the product, whiteboard diagramming or verbal explanations of complex processes. If the meeting is in person, you can even take photos of the whiteboard.

6. **Prompt the SME.**
   Even if you sent the list of questions in advance, don't assume the SME will give you all the answers from the start. Experts often have a lot of tacit knowledge - things that "everyone already knows". (Famous last words!) As a representative of the end-users, it's your job to ask follow-up questions and surface this tacit knowledge.

7. **Write down the conclusions.**
   Take the meeting notes as soon as possible and confirm with the SME if you are unsure of something.

Next Step

Research is never really done, but when you are confident that you are ready to start the first draft, it's time to move on to the next stage:

- **Writing**. This one seems obvious, but there are some important things to consider if you want to launch something truly professional.

3.3. Writing

You've analyzed your audience, you've gathered your materials, done your research... now it's time to actually ***write***.

Following a few simple principles can mean the difference between an unreadable wall of text and a sleek, easy to parse set of instructions.

Use Clear and Simple Language

Technical writing should be precise and easy to understand. Follow these guidelines:

Use plain language.

Choose words that are simple and familiar to your readers. You are here to inform, not to show off your vocabulary.

Avoid jargon.

Sometimes you can't escape jargon, especially if you are writing about highly technical topics, but make sure to provide explanations or definitions to help readers understand.

Prefer active voice.

Active voice makes sentences more direct and makes it clear who or what is doing the action.

Use second person.

It's "you", not "the user". You are talking directly to the people who will be following the instructions.

Use present tense.

Present tense makes actions feel immediate and actionable.
Avoid "should", though! It implies something is supposed to happen, but *may not*, which can make users lose confidence.

Break down complex concepts.

If a concept is difficult, provide examples, analogies, or step-by-step explanations to make it easier to understand.

Write for a global audience

Ensure all users, including non-native speakers, can easily

Cultural Sensitivity

Pay attention to variations in date formats, units of measurement, and

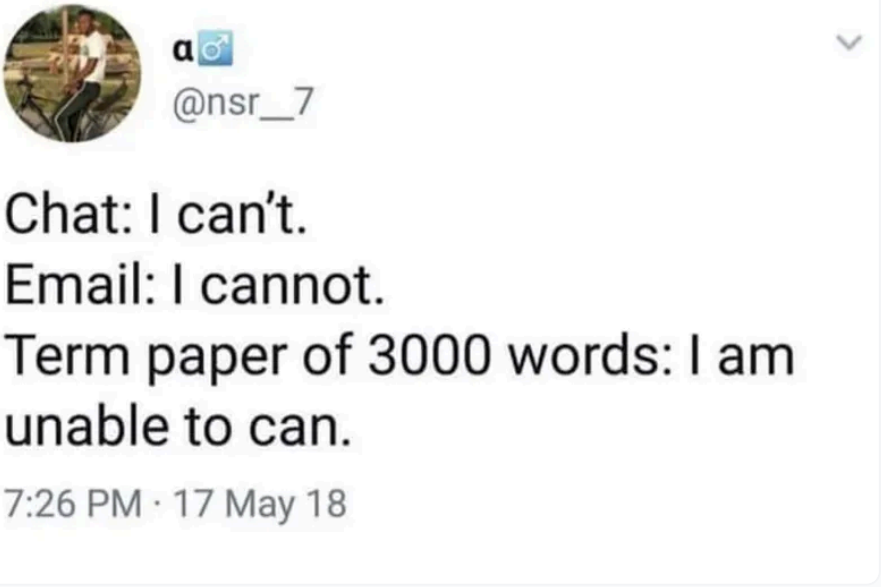
understand the documentation.

Avoid idioms and slang that might not translate well or be universally understood.

symbols across different regions.

Use culturally neutral examples to avoid unintended offense or confusion.

Be Concise and Focused



Readers want to find answers quickly, without obstacles. Follow these guidelines:

- **Use short sentences and paragraphs.**
  Technical manuals are not novels! Aim for sentences with 20 words or fewer for easy readability.
- **Get to the point quickly.**
  Avoid lengthy introductions or unnecessary background information.
- **Eliminate unnecessary words.**
  Be direct and remove filler words that do not add value.

- **Avoid repetition.**
  If something has already been explained, don't restate it, just point readers to the prior explanation.

- **Avoid unnecessary explanations.**
  If your audience is familiar with certain concepts, don’t over-explain them. Assess their level of knowledge and adjust accordingly.

Organize Information Logically

At a micro level, a well-structured page is easier to follow and helps users find information quickly. At a macro level, **Information Architecture** dictates how the entire documentation is organized. Both aspects are crucial for making your documentation effective!

Follow these principles when creating the structure of a page or topic:

- **Use headings and subheadings.**
  Clearly labeled sections help readers navigate the content efficiently.

- **Group related information.**
  Keep similar topics together and follow the same structure everywhere.
  For example, if you are documenting a user interface, always have a topic that describes the window, followed by a topic that describes the available buttons. Don't mix the descriptions of the buttons with the descriptions of the tasks.

- **Present information in a logical sequence.**
  Document steps in the order they should be performed. If a task has prerequisites, make sure to mention them before starting to explain the task.

- **Use numbered lists for step-by-step instructions.**
  Numbered list indicate to readers that they should follow the instructions in order.

- **Use bullet points for options.**
  Bullets help break up text and make critical information easier to scan. Use bullets where the order of the items doesn't matter.

- **Tables and charts enhance understanding.**
  When presenting complex data, visual elements like tables, charts, and graphs can improve clarity and retention.

Use Visuals to Support Text

Images, diagrams, and screenshots can clarify instructions and make content more engaging. Follow these guidelines:

- **Ensure the visuals are high quality and relevant.**
  Avoid low-resolution or badly cropped images. Full-screen screenshots help the users orient themselves, but if you need to focus on a part of the screen crop the image carefully so that the results looks professional.

- **Label images and provide captions.**
  A brief description helps the reader understand the purpose of the visual.

- **Use callouts to highlight important elements.**
  Marking specific areas of an image can guide the reader's attention to key details. Use software that allows you to save the callouts as a separate layer, so you can update them if needed.

- **Don't use *too many* images.**
  Images should support the written content rather than replace it entirely. Always provide text explanations for key points.

- **Plan for screenshot maintenance.**
  If you are documenting a user interface, odds are it will eventually change and you will need to replace all the screenshots. Keep this in mind when deciding whether or not to insert a screenshot - sometimes plain text is just fine.

- **Use diagrams to explain complex processes.**
  Flowcharts and infographics can make complicated procedures easier to understand.

Follow a Consistent Style

Technical writers don't aim for creativity, so don't be scared

A consistent writing style improves readability, so a style guide is a must. Follow these guidelines:

- **Choose an existing style guide or create your own.**
  The following section has some ideas and **Reference: Style Guides** lists the most popular guides you can choose from.
- **Use standard formatting for headings, punctuation, lists, and so on.**
  It's essential for the professional aspect of your documentation.
- **Use consistent terminology.**
  Never refer to the same concept with different words - it will just cause mental load.
- **Consider defining templates for frequently used document types.**
  They don't have to be very formal - even a checklist of questions to ask is useful. The **Good Docs Project** has an extensive (and ever-growing) collection of templates you can use and adapt.

Choosing a Style Guide

Adopting a publicly available style guide is the simplest way to ensure consistency and standardization in your documentation, but a single style guide will never cover all of an organization's needs.

The best way to solve this problem is to use a hierarchy of guides:

1. First, define **internal guidelines** that describe word usage, branding, tone and voice specific to your company or product.
   Internal guidelines should contain only things that are *too* specific to be included in a general-purpose style guide. For everything else, it's not worth the effort to reinvent the wheel.

2. For general technical writing matters, use the **technical writing style guide** most appropriate for your industry - for example, the Microsoft guide for software that runs on Windows or the Red Hat guide for open source. You can find an overview of the most popular style guides in **Reference: Style Guides**.

3. For general grammar or language queries, use the **general-purpose style guide** that best aligns to your chosen tone of voice. The Chicago Manual of Style is a popular one.

4. For spelling, use a **standard dictionary**, such as Merriam-Webster, for American English.

> Always remember: consistency is the most important. Don't agonize over a guide choice and don't overanalyze word usage.
>
> If the guides don't give a definitive answer, just make a choice, document it in the internal guidelines and use it consistently.

Edit Ruthlessly

The work does not end after you've written the documentation. The first draft is rarely the best. Follow these guidelines:

- **Cut out anything that does not directly contribute to the main message.**
  Re-read the *Be Concise and Focused* principles and make sure they are applied throughout.

- **Check for inconsistencies.**
  Proofread your text and fix any mistakes in terminology, spelling or formatting.
- **Review the document as a whole.**
  Ensure that sections flow logically and that any references to previous content are correct and consistent.
- **Check consistency with the product you are documenting.**
  Before sending the text off for review, make sure it matches the final version of the interface or physical product.

Next Step

The hard part is done, but there are still some stages until you can deliver the documentation to your end-users.

The next one:

- **Reviewing**. Whether it's just proofreading from a coworker or a thorough review from a business analyst, it's always a good idea to have another pair of eyes on your work.

3.3.1. Reference: Style Guides

A style guide serves as a set of standardized rules and guidelines that define the formatting, language and tone of technical documents.

> A style guide also means that writers have more time to think about **what** they say instead of **how** they say it.
>
> Once you internalize the standards of a style guide and you don't have to choose your words, writing becomes much faster!

Style Guides Designed for Technical Writing

The following list includes some of the most popular English-language guides that are written specifically for technical documentation.

Microsoft Manual of Style

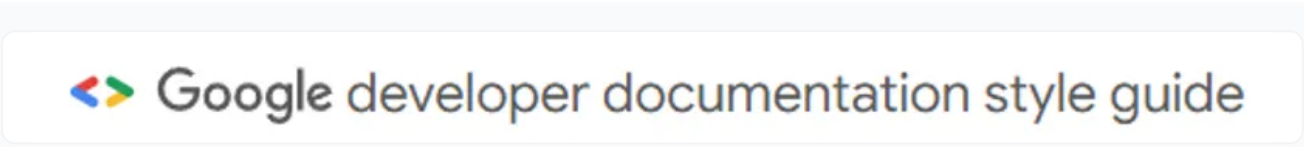
The Microsoft style guide covers a broad range of topics, including grammar, terminology, formatting and style preferences for digital content. It is mostly used in the software industry, particularly for products that run on Microsoft Windows.

Microsoft Manual of Style (online, free)

Be aware that there is some controversy in the community! The online version is an abbreviated form of the previous, printed guide, and some writers feel it's lacking in content as a result.

You may still be able to buy used copies of the **last printed edition**, the 4th edition.

Google Developer Documentation Style Guide



Google's style guide covers the usual technical writing topics, such as content structure, terminology and how to write for an international audience. It is more concise than other guides and focuses on short explanations and examples.

Google Style Guide (online, free)

Apple Style Guide

Apple’s style guide covers the tone and voice of Apple's product documentation, focusing on clarity and precision. The guide also includes information for developers working on user interfaces for Apple devices.

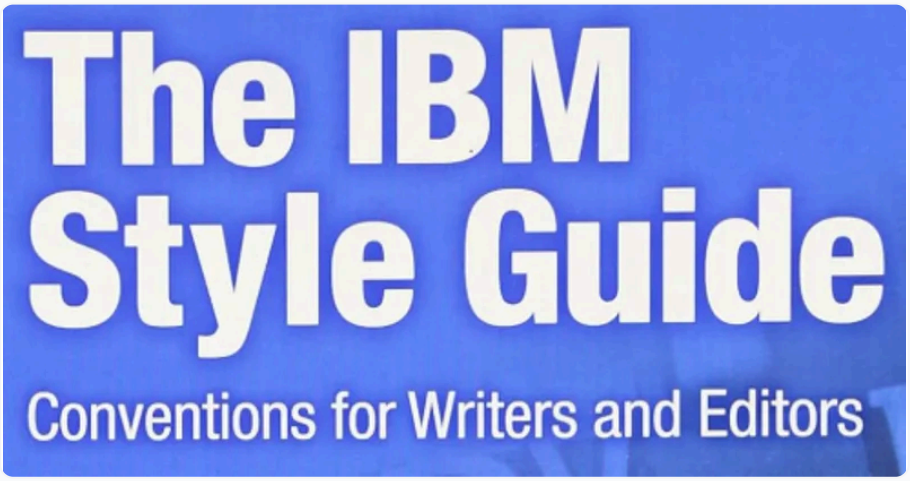
Apple Style Guide (online, free)

Red Hat Style Guide

Red Hat’s style guide is aimed at open-source software documentation, providing detailed instructions on formatting, writing conventions and technical terminology.

Red Hat Style Guide (online, free)

IBM Style Guide



IBM's style guide is used especially for large-scale technical documentation. It offers detailed guidance on terminology, tone and writing conventions.

IBM Style Guide (online or paperback, paid)

ASD-STE100

ASD-STE100 is not a style guide per-se, but it is a controlled language standard that serves the same purpose. STE stands for Simplified Technical English and the standard was developed by the Aerospace and Defence Industries Association of Europe (ASD). It is mostly used in aerospace, defense, military, manufacturing and engineering.

ASD-STE100 Standards (link to form to request the standard documentation, free)

IEEE Editorial Style Manual

This style guide was developed by the Institute of Electrical and Electronics Engineers (IEEE) and is primarily used for academic papers, research articles and technical documentation in engineering.

IEEE Editorial Style Manual (online, free)

ASME Style Guide

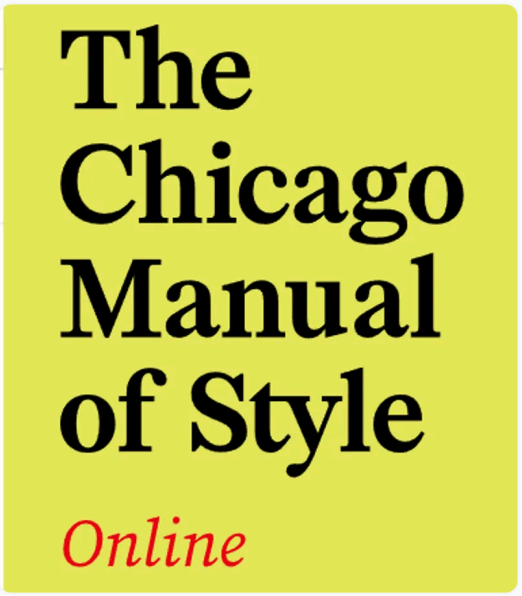
The ASME Style Guide is developed by the American Society of Mechanical Engineers (ASME) and provides guidelines for writing and formatting technical papers, journal articles, and engineering reports.

ASME Journal Information for Authors (online, free)
ASME Author Guidelines for Proceedings (online, free)

> For more details and examples of popular style guides, you can also read **this Archbee blog post**.

Style Guides from Other Industries

Chicago Manual of Style (CMOS)



While it's a general-purpose style guide, CMOS is often used by technical writers. It is extremely detailed and covers most usage of the English language.

Chicago Manual of Style (printed or online, paid)

Oxford Style Manual

This style guide is written for writers and editors in all domains and it's often referenced by technical writers for its grammar and punctuation rules. It also includes a dictionary

Oxford Style Manual (printed, paid)

Associated Press (AP) Stylebook

While the AP Stylebook is traditionally used in journalism, it is also widely adopted in technical writing, particularly for web content. It's a common choice for technical writing aimed at a broad audience.

AP Stylebook (printed or e-book, paid)

3.4. Reviewing

Publishing wrong documentation can have consequences.

Messing up the guidelines for complicated machinery can harm operators. In software, a wrong step can lead to a system failure costing large amounts of money.

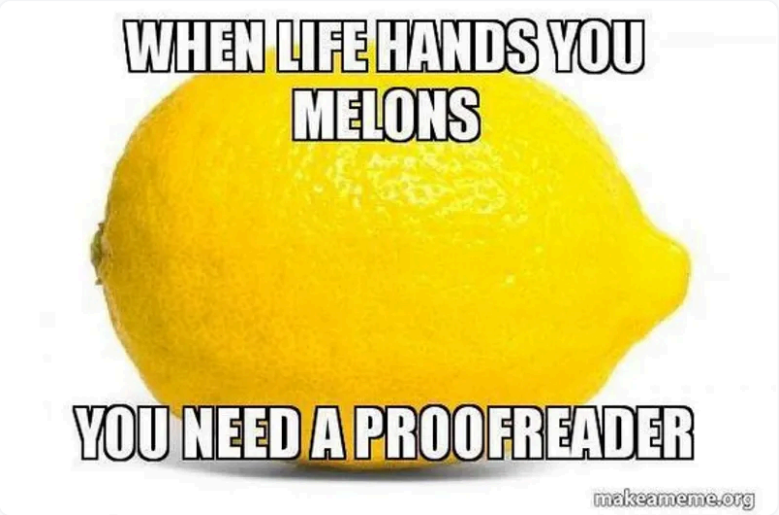
It's obvious why reviewing documentation prior to publishing is essential.

A good review process follows three steps:

1. **Self-review**
   The writer checking their own work to ensure clarity, consistency and accuracy before sharing with others.

2. **Subject matter expert review**
   One or more SMEs checking the documentation for technical accuracy.

3. **Peer review**
   A peer checking the final draft for readability and flow, providing feedback from the perspective of the intended audience.



Self-Review

A self-review allows you to catch obvious errors and inconsistencies. Even if you did all these things during the research and writing phase, it's important to have a dedicated review phase where you read the documentation end-to-end.

Even with good research and helpful SMEs, double-checking your information is critical. SMEs forget details, specifications are not updated, last-minute changes are introduced... and it's not uncommon for technical writers to actually uncover scenarios that no one had thought about!

The self-review phase can be split into reviewing technical accuracy and reviewing content and style.

Reviewing Technical Accuracy

Even with a helpful and involved SME, errors can still slip through. You should always try to validate the documentation, to the best of your ability, by:

- Following the documented steps yourself in a test environment
- Testing commands and code snippets
- Cross-referencing information sources and flagging any discrepancies

Reviewing Content and Style

Even if it's just a first draft, the document you send out for SME review should be as close to finished as you can make it.

Ask yourself these questions:

- Did you follow the company style guide?
- Did you use consistent terminology?
- Does the structure align to the guidelines and is it easy to follow?
- Did you proofread and check for typos?

> It's best to wait some time between the moment when you finished the draft and the moment you start the review.
>
> The break will help you return to the document with a clear mind, allowing you to spot errors or inconsistencies you might have missed while writing.

Subject Matter Expert Review

Since you have (hopefully) already had conversations with the SME during the research phase, the SME review should not uncover *new* information.

However, this is probably the first time the SME sees the documentation in its final structure. To ensure a thorough review:

- **Send the document out for review only after the feature is finalized or the code is frozen.**
  This ensures that no further changes can occur. (At least in an ideal world! You know your environment best.)

- **If you added bits and pieces of information in an existing document, point out the additions.**
  The SME needs to know which parts to focus the review on.

- **If needed, ask the SME to review the legacy documentation as well.**
  Sometimes the old documentation also needs to be updated to accommodate a new feature, and it's not always obvious to a non-expert.

- **Encourage the SME to *really* check for accuracy.**
  Some SMEs tend to assume that, just because they gave you some information in the research phase, you understood everything. This is not true - technical writers may misinterpret, misquote or rephrase SME information in incorrect ways.

> Sometimes SMEs want to have input on the grammar, tone, phrasing and structure of the documentation.
>
> This is not necessarily a bad thing - but remember that, when it comes to the documentation itself, **you** are the SME. Implement their suggestions if they make sense, but not if they go against the style guide or technical writing best practices.

Peer Review

Once the technical details are verified, it’s time to get a fresh set of eyes on the document.

The type of peer you choose depends on the type of review you're looking for: you can ask for a fellow technical writer to check alignment to the style guide or you can ask a customer support representative to see if it's user-friendly enough.

Next Step

Once the review is done, you are almost ready to deliver, but you should still dedicate some time to an often-overlooked step:

- **Testing**. Sometimes it's extensive, sometimes it's just a spot check, but it's still a good idea to do it.

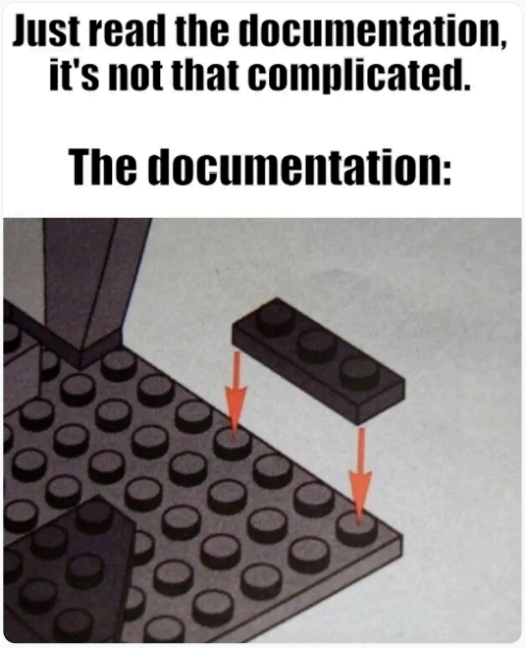
3.5. Testing

It might seem like final approval from the SME is the last step in your documentation journey. Think again! Testing, and particularly *regular* testing, is essential for keeping your documentation usable, accurate and complete.

Types of Tests

Testing technical documentation involves a combination of **manual** and **automated** approaches. Depending on the tools and knowledge you have available, you can choose the right mix of the two.

- **Manual testing** works best for subjective or visual issues that tools can't reliably detect.
  For example, evaluating the tone of a document or checking image clarity.
- **Automated testing** is recommended for repetitive or technical issues.
  For example, link validation or spell checking.



Documentation Testing Checklist

This is a non-exhaustive list of tests you can run in you documentation:

- **Check all internal/external links**, to avoid broken links or bookmarks.
- **Validate alt text on images**, for accessibility.
- **Ensure consistent heading structure**, to make sure you didn't skip a heading level (for example, H1 followed by H3).
- ✍ **Spell check**, to avoid typos.
- **Test all embedded code snippets and commands**, to ensure that they can be copied and run correctly.
- **Preview layout on mobile and desktop**, to ensure good UX.
- **Confirm navigation and TOC integrity**, to catch any topics that may be missing from the TOC.
- **Check metadata (title, description, tags)**, for correct indexing.
- **Check image quality**, to have a professional-looking documentation
- **Find unused files**, to avoid storing and publishing useless or out of date content.
- **Check file naming**, for standardization purposes and to avoid cross-OS issues.
- **Update variables**, to make sure they apply to the current version.

Validate adherence to the style guide, for consistency.

Periodically check if old content needs to be updated, to avoid stale pages.

> The content-level tests are generally done during the review phase and are detailed in **Reviewing**

Some tools that can be used for testing documentation are presented in **Documentation Workflows**.

Next Step

After testing, you can call your documentation complete. The only remaining step is making it available to the users: **Delivering**.

3.6. Delivering

The final choice in the documentation content strategy is how to deliver the documentation. Each delivery methods has its own benefits, depending on the target audience and the type of content.

Delivery Methods

Some of the most common methods are:

- **Online documentation**
  Web-based documentation is often the most accessible and user-friendly option. It allows for easy updates and is accessible from any device with an internet connection.

- **Printable documentation**
  PDFs are a popular format for delivering downloadable, printable documentation. This is ideal for users who need offline access or need physical copies of documents.

- **In-app documentation**
  In-app documentation, such as tooltips or pop-ups, provides users with immediate, contextual support while they are interacting with an application or tool.

- **Video tutorials**
  Videos are a highly engaging and effective way to deliver complex instructions or demonstrate how to use a product or service.

- **Documentation in version control systems**
  For teams working on software development, documentation can be maintained directly within version control systems (like GitHub or GitLab), in the form of README files or wikis.

- **Chatbots**
  Some applications use chatbots to assist users in real-time as they navigate through the interface.

> It is important to remember that "HTML help" does not necessarily equal "online" and "PDF" does not necessarily equal "printed". Always think of your audience: a company in a regulated industry may not have internet access, but may be able to install HTML help on an internal web server to provide a more modern way of accessing the documentation.

Next Step

...nothing! There are no more steps. You have successfully delivered your documentation project and you can relax.

(Or move on to the next project, of course.)

4. Documentation Workflows

Choosing a documentation workflow isn't just about process - it's about shaping the entire toolchain. Sometimes *you* won't get a choice, but just know that the way you or your team approach documentation directly influence the technologies you use.

From Git-based static site generators to collaborative WYSIWYG editors, your workflow determines **how content is created**, **reviewed**, **published** and **maintained**.

Broadly speaking, there are two types of documentation workflow:

Traditional documentation

A content-focused approach using WYSIWYG tools and separate publishing workflows. It prioritizes ease of use and polished output, often maintained by dedicated documentation teams.

Docs as code

A developer-centric workflow where documentation is written in plain text, versioned with Git, and integrated into the software delivery pipeline. It emphasizes automation, collaboration, and treating docs like code.

> Your particular workflow doesn't need to align 100% to one of these definitions.
>
> For example, you could author in a tool like MadCap Flare or oXygen, which are based on HTML or XML files (but not Markdown), and yet have the documentation integrated into the software delivery pipeline.

This table shows a high-level comparison of the two approaches.

Aspect	Docs as Code	Traditional Documentation
Tooling	Git, Markdown, static site generators (e.g., Sphinx, MkDocs)	Word, Google Docs, SharePoint
Format	Plain text (Markdown, AsciiDoc, reStructuredText)	Rich text formats (.docx, HTML in WYSIWYG)
Workflow	Pull requests, branches, code reviews	Email approvals, manual edits, centralized review
Publishing	CI/CD pipelines, automated deployment	Manual publishing, uploading to intranet or PDF export
Versioning	Git-based, tied to code versions	Often ad hoc or based on manually tracked versions
Collaboration	Developers and writers collaborate in same repos	Writers work separately; developers may not see docs
Audience Fit	Ideal for developer-centric content and APIs	Better for policy, compliance or non-technical audiences
Maintenance	Continuous updates, co-evolves with code	Periodic updates, can easily get outdated
Review Process	Integrated with code review (e.g., GitHub PRs)	Document-specific workflows, approval chains
Access Control	Repo-based permissions, open or private	Often restricted, siloed by department or role

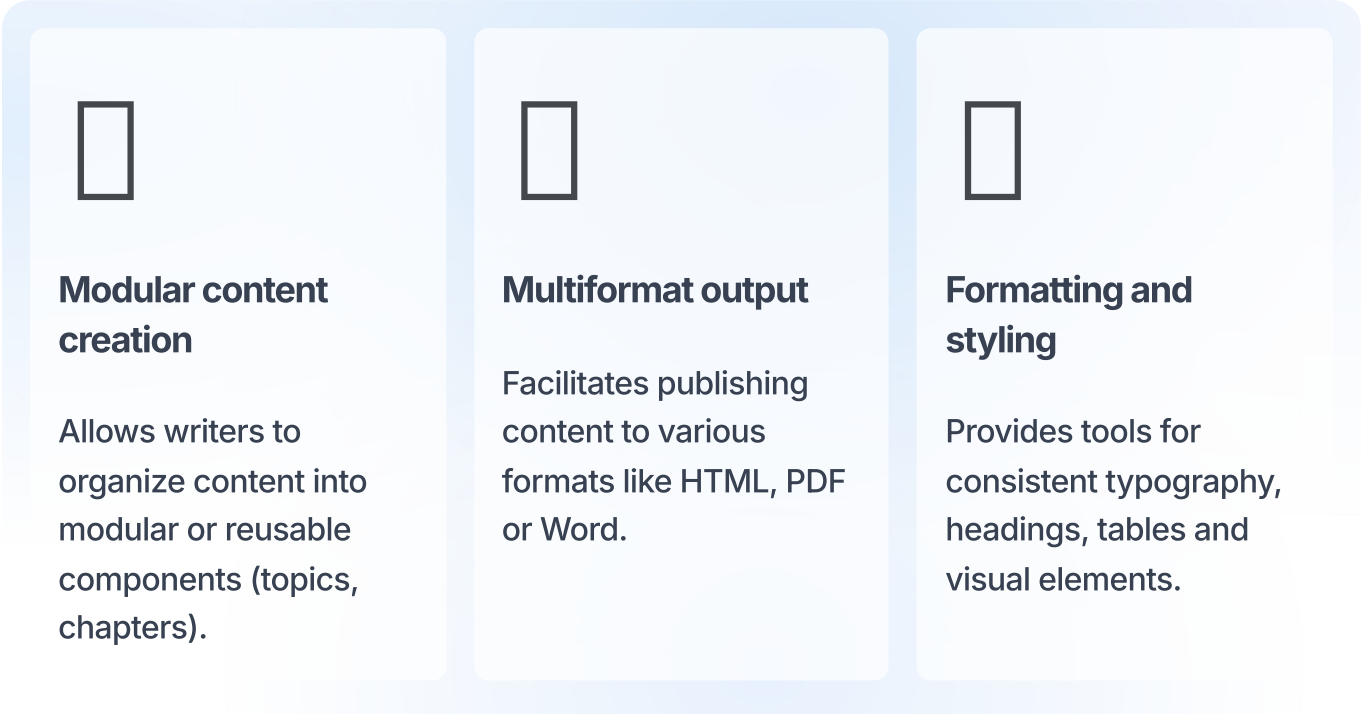
> If you want to learn more about docs-as-code, you can start from **this beginner-friendly article**.

The following section serve as an overview of tools and technologies frequently used in technical writing, and they should help you make the best choice for your team.

4.1. Authoring Tools

Generally speaking, an authoring tool is a software application or platform that enables technical writers to create, edit, structure and manage content for documentation.

An authoring tool typically supports:



Depending on the tool, you could author in:

- **Pure WYSIWYG** ("what you see is what you get")
  Tools like Microsoft Word or Google Docs provide a visual interface for formatting text, inserting images and creating tables, without needing technical skills.
  With pure WYSIWYG tools, users don't have access to the underlying structure of the documents.

- **Wiki-style**
  Wiki platforms such as Confluence allow collaborative, web-based documentation creation with real-time updates and internal linking.
  Most wikis give users access to edit the underlying code (HTML, Markdown), but not always out-of-the-box.

- **HTML-based**
  Tools like MadCap Flare or Adobe RoboHelp support complex layouts and multi-

format publishing.
Users have access to a WYSIWYG editor, but can also edit the HTML source files directly.

- **Plain text**
  Markdown, AsciiDoc and reStructuredText have lightweight syntax, ideal for developers. (AsciiDoc and reStructuredText offer more complex features compared to Markdown.)
  To publish documentation written in a plain text format, you need a static site generator like Docusaurus or Hugo.

Choosing an Authoring Tool

The choice between docs as code and traditional documentation equally depends on two things: **people** (who is writing and maintaining the documentation) and **features** (what functionality is mandatory for your use case).

> The "best documentation tool" doesn't exist. All that exists is the **best tool for a specific scenario, within specific constraints**. Use the tables below, add your own questions, then review the list of popular authoring tools for ideas.

When it comes to the people, focus on these aspects:

Consideration	Comments
Technical vs non-technical contributors	The contributors' comfort level with technical things is important - someone who only uses Word will not want to learn Git and Markdown just to add a sentence in your docs.
Level of involvement	Make sure your coworkers actually want to write documentation. Some genuinely prefer to just explain things and have a professional handle the writing.
Management stance	If management doesn't support their employees in contributing to documentation, it's not going to happen.

For the technology part, focus on these features:

Feature	Comments
Collaboration	If multiple non-technical people have to contribute, choose tools like **Confluence** or **Google Docs**. If multiple technical people are involved, consider a **docs-as-code** workflow.
Advanced features (versioning, variables, conditions)	Some tools are simple to set up, but lack advanced features. Tools that do have these features vary in their ease of use.
Scalability	For large volumes of documentation, build times matter.
Offline documentation	Offline documentation is mandatory in some industries, so if you choose a cloud-based tool, make sure it has a way to export the documentation for offline consumption.
Automation support	If you want to set up a delivery pipeline for the documentation, choose a tool that supports this (e.g., command-line-only tools or those with batch versions).
Customization	You may need to adapt the look and feel to your company brand. Some tools offer visual customization, while others require web development knowledge.
Text editor	The choice of editor (WYSIWYG vs plain text) depends on who is contributing to the documentation.
Access control	If the documentation needs to be password-protected, choose a tool that supports it or budget time and resources to implement it.
Output type	Do you need to generate a website or a standalone document? Not all documentation tools generate PDFs natively, and Word support is ever rarer. The support for customizing PDF output varies from tool to tool.
Migration	If you already have documentation in a different tool, consider the migration cost.
Integration with external	If you need to publish to platforms like Salesforce or Zendesk, choose a tool that offers native export.

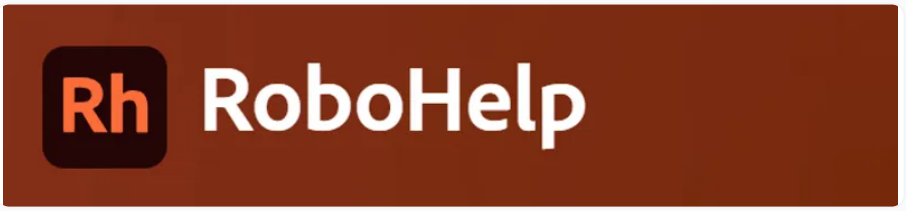
Feature	Comments
platforms	
Support	Analyze the support offerings before choosing a tool—this could range from official vendor support to StackOverflow and GitHub issues.
Costs	Most free tools still have a cost—generally a bigger time investment for setup and maintenance.

> An more in-depth analysis of the decisions to make when choosing a tool is available from Brian Dominick **here**.

Authoring Tools Reference

This section describes some of the most popular authoring tools on the market (as of 2025). They are loosely grouped by category, but a definitive grouping is difficult to define, since most tools can be used for different purposes.

Adobe RoboHelp



Attribute	Description
Website	Adobe RoboHelp
Description	A help authoring tool used mostly for HTML help systems. It supports conditional content, variables and output customization.&##xA;It has a high learning curve for complete newbies, but requires no programming knowledge.
Deployment	Desktop
Authoring Format	HTML, using a WYSIWYG editor
Output Format	HTML5, PDF, Word, CHM, etc

MadCap Flare

Attribute	Description
Website	MadCap Flare
Description	A help authoring tool used mostly for HTML help systems. It supports conditional content, variables and output customization. It has a high learning curve for complete newbies, but requires no programming knowledge. Flare can integrate with MadCap Central, a cloud-based content management platform which offers version control and hosting/publishing for Flare projects.
Deployment	Desktop (Windows-only)
Authoring Format	XHTML, using a WYSIWYG editor
Output Format	HTML5, PDF, Word, CHM, ePub, etc

Adobe FrameMaker

Attribute	Description
Website	**Adobe FrameMaker**
Description	A desktop publishing and structured authoring tool, best suited for complex documents published in PDF. Newer versions can also generate online help.&##xA;It supports conditional content and variables. It has two modes of authoring: structured (for example, DITA) and unstructured.&##xA;It has a high learning curve for complete newbies, but requires no programming knowledge.
Deployment	Desktop
Authoring Format	Proprietary format, using a WYSIWYG editor
Output Format	PDF, HTML5, ePub, RTF, Word

Oxygen XML Editor

Attribute	Description
Website	**Oxygen XML Editor**
Description	An XML authoring tool for creating structured content in DITA, DocBook and other standards. &##xA;Offers validation, transformation scenarios and integration with content management systems.&##xA;It has a high learning curve for complete newbies, but requires little programming knowledge. However, users must understand the basics of XML and customization for PDF output is done with XSLT.
Deployment	Desktop
Authoring Format	XML, using a WYSIWYG editor
Output Format	HTML, PDF, ePub, Word, XML, DITA

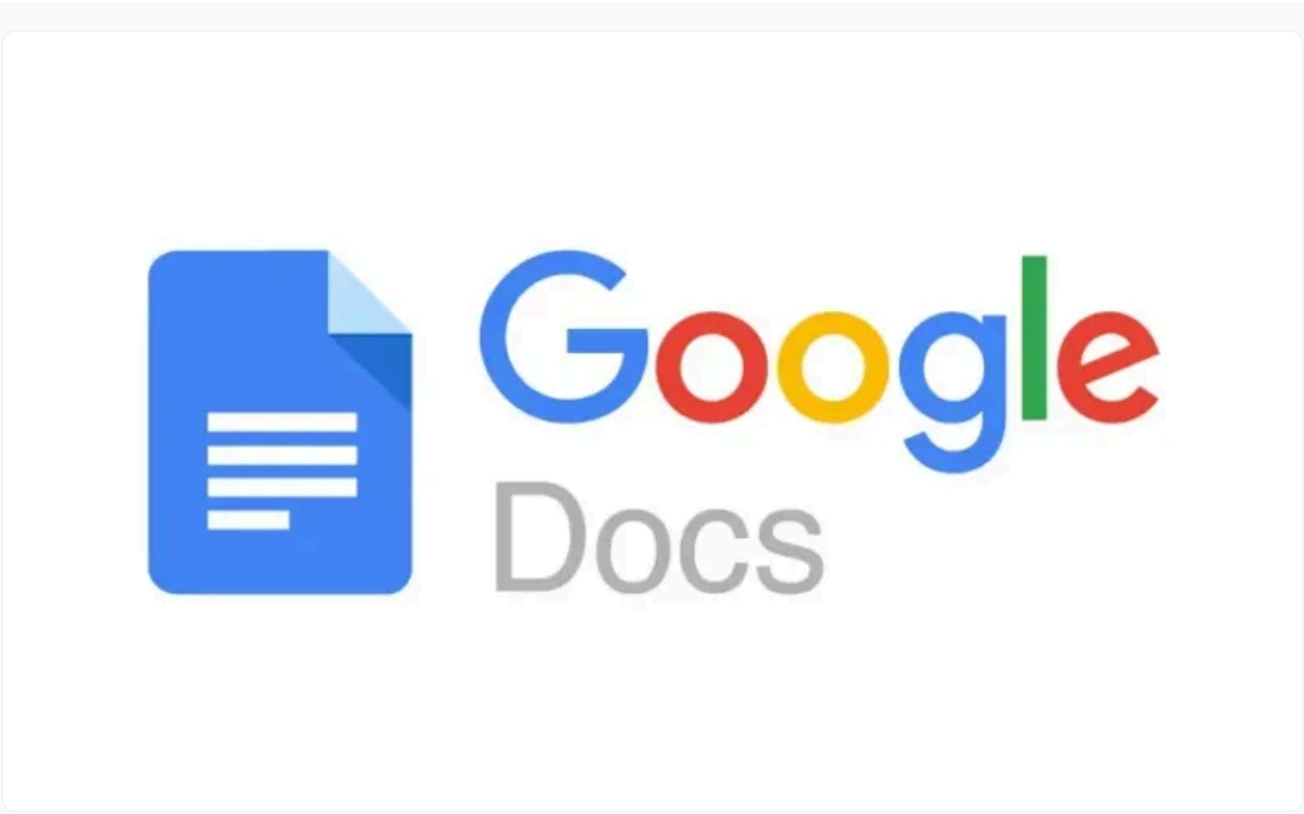
RWS Tridion Docs

Attribute	Description
Website	**Tridion Docs**
Description	Enterprise-level documentation management system for creating, managing and delivering DITA structured content.
Deployment	Cloud
Authoring Format	XML, using a WYSIWYG editor
Output Format	HTML, PDF, Word, XML

Microsoft Word

Attribute	Description
Website	**Microsoft Word**
Description	The most used word processor. It supports basic styles, templates and review features like track changes and comments. &##xA;Not ideal for complex documents due to lack of support for topic-based authoring and conditions, but available on almost every computer.&##xA;It is easy to learn how to do basic tasks.
Deployment	Desktop
Authoring Format	Office Open XML, using a WYSIWYG editor
Output Format	DOCX, PDF, RTF, HTML (single-page)

Google Docs



Attribute	Description
Google Docs Website	**Google Docs**
Description	Word processing tool available online, with limited features such as basic styles, templates and review.&##xA;Not ideal for complex documents, but is easy to learn and is free.
Deployment	Cloud
Authoring Format	WYSIWYG editor
Output Format	DOCX, PDF, HTML, RTF

Static Site Generators (MkDocs, Sphinx, Jekyll, Hugo, etc)

Attribute	Description
Website	MkDocs / Sphinx / https://jekyllrb.com/https://jekyllrb.com/ Jekyll / https://gohugo.iohttps://gohugo.io Hugo &##xA;An extensive list of static site generators is https://jamstack.org/generators/https://jamstack.org/generators/ available on JamStack .
Description	Lightweight tools that convert plain text files into static websites. &##xA;Popular in developer communities due to support for authoring in any text editor and lack of upfront costs. &##xA;Support various features out-of-the-box (depending on the generator) and are extensible, but require programming/DevOps knowledge to extend and maintain.
Deployment	Desktop
Authoring Format	Markdown, AsciiDoc, reStructuredText, using a plain text editor (VSCode, NotePad++, Sublime, etc)&##xA;(depending on the generator)
Output Format	HTML, PDF (with additional plugins)

GitBook

Attribute	Description
Website	**GitBook**
Description	A lightweight platform for writing and publishing documentation, popular with development teams.
Deployment	Cloud
Authoring Format	Markdown, using a WYSIWYG editor
Output Format	HTML, PDF

Paligo

Attribute	Description
Website	**Paligo**
Description	Structured authoring tool based on DocBook XML. Supports multi-channel publishing and content reuse.
Deployment	Cloud
Authoring Format	XML, using a WYSIWYG editor
Output Format	HTML, PDF, ePub, Word, etc

Confluence

Attribute	Description
Website	Confluence
Description	A wiki platform by Atlassian. &##xA;It is popular for its collaboration features, for ease of authoring for non-technical people and integration with Jira and other Atlassian tools.&##xA;Supports advanced features like conditional content and variables only through plugins, such as those offered by https://www.k15t.com/https://www.k15t.com/ K15t .
Deployment	Cloud
Authoring Format	HTML, using a WYSIWYG editor
Output Format	PDF, HTML

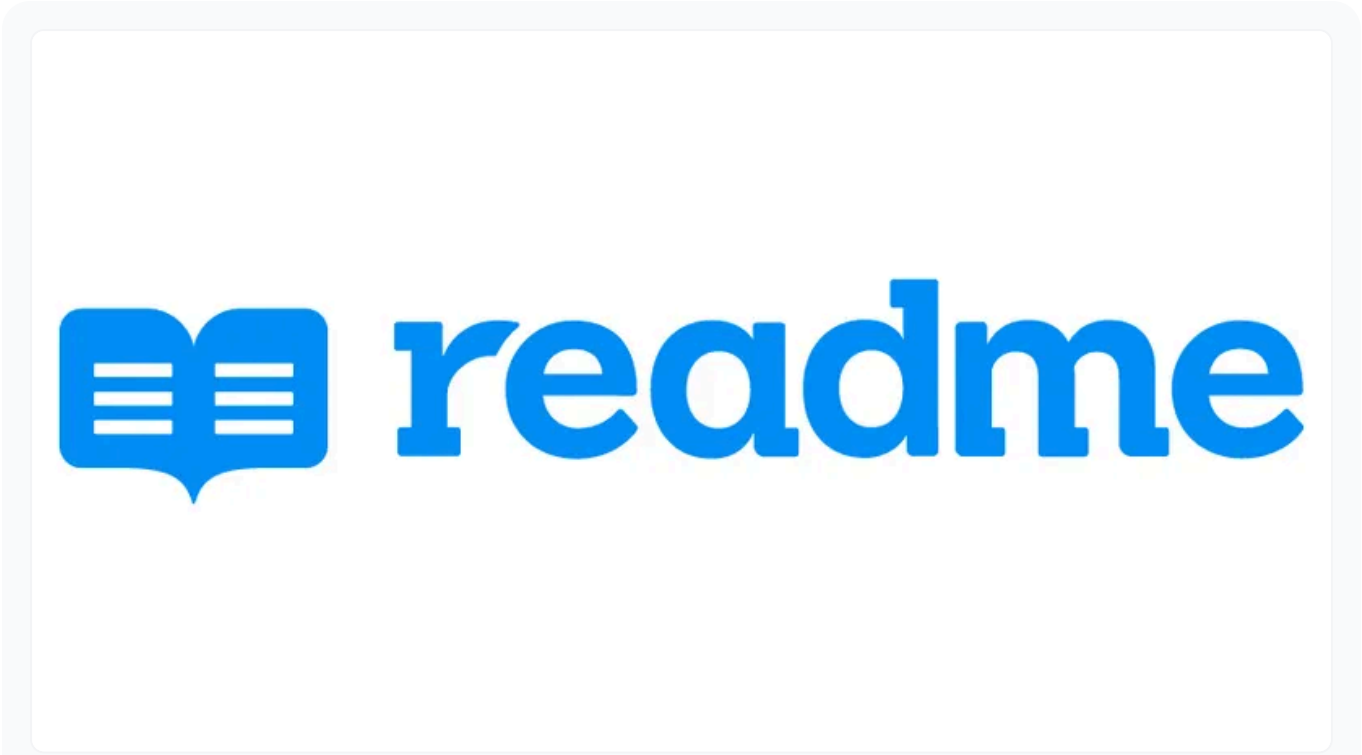
Archbee

Attribute	Description
Website	Archbee
Description	A cloud-based knowledge management and documentation platform. Supports conditional content, variables, versioning and collaboration.
Deployment	Cloud
Authoring Format	Markdown, using a WYSIWYG editor
Output Format	HTML, PDF

Document360

Attribute	Description
Website	Document360
Description	A cloud-based knowledge management system for creating internal documentation and knowledge bases. &##xA;Supports conditional content, variables and collaboration.
Deployment	Cloud
Authoring Format	Markdown or WYSIWYG Editor
Output Format	HTML, PDF

ReadMe



Attribute	Description
Website	**ReadMe**
Description	Platform designed for creating API documentation.
Deployment	Cloud
Authoring Format	Markdown, using a WYSIWYG editor
Output Format	HTML, PDF

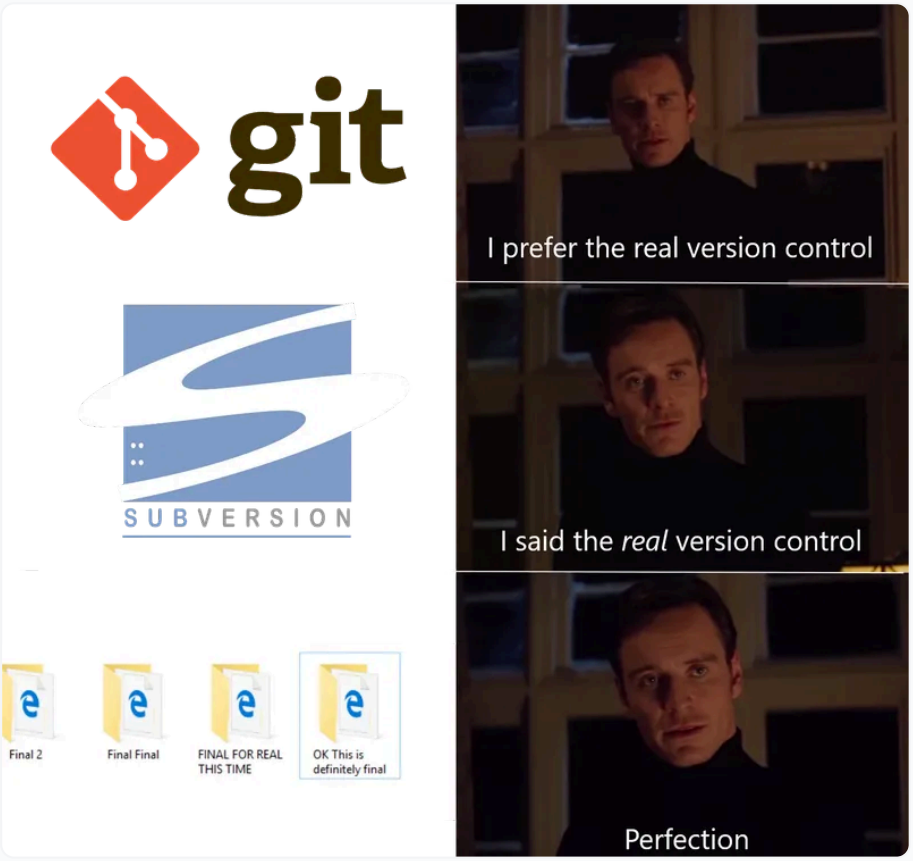
Redocly

Attribute	Description
Website	**Redocly** / Reunite
Description	Reunite, part of the Redocly offering, is an API documentation platform for managing and versioning API content.
Deployment	Cloud
Authoring Format	OpenAPI, Markdown
Output Format	HTML

4.2. Version Control

Version control ensures that changes to documentation are tracked and managed efficiently. By using version control systems (VCS, also called source control systems), teams can collaborate better, easily access a history of changes and be protected against loss of content.

And yet, so many people don't use it! This page will try to explain that even basic version control is better than nothing.



Using a version control system, you can benefit from:

- **Collaboration**
  Multiple people can work on the same document simultaneously, without overwriting each other's work.

- **Change tracking**
  Version control keeps a clear history of document changes, including who made them and why, which is essential for accountability and auditing.

- **Reverting to previous versions**
  If any mistakes happen, it's no problem - version control allows to revert to an earlier version.

- **Branching**
  If you need to support multiple versions of the documentation at the same time, a version control system enables you to work on them independently.

> This description best fits a "true" source control system, like Git. However, the principles stay the same even if you don't work in a developer-like environment. Even storing your documents in a SharePoint library is better than nothing! Documents stored only on a local hard drive are a recipe for disaster.

The type of version control system to use depends on the type of content being managed and the workflows that need support:

- **Software development version control systems** like Git or SVN are best suited when you are working on text files, such as Markdown or HTML.
  (That said - if you have access to a VCS, feel free to store binary files too. It's true that VCS are not meant for binary files and you won't be able to view diffs, but the ability to revert to a previous version and the prevention of data loss are worth it.)

- **Document management systems** such as SharePoint are recommended when working with Word documents.

- Cloud authoring tools, like Google Docs, Confluence or Archbee, have a **built-in version history system**.

4.3. Testing Tools

The importance of testing documentation was already covered in the **Testing** section.

The following lists offers some ideas of things you can test for and some suggestions of tools to use:

- **Grammar, punctuation, spelling**
  - **Grammarly** - AI-powered writing assistant that checks for grammar, punctuation, and spelling errors.
  - **ProWritingAid** - Style and grammar checker with in-depth reports on writing mechanics.
  - **Vale** - A customizable, command-line linter for prose that enforces style guides.
  - The **built-in spellchecker** of your authoring tool.
- **Style and readability**
  - **Hemingway Editor** -Highlights long, complex sentences and suggests simpler alternatives.
  - **Readable** - Analyzes text readability using various readability scores (Flesch-Kincaid, Gunning Fog, etc.).
- **Links and navigation**
  - **Screaming Frog SEO Spider** - Crawls websites to audit links, images, scripts, and more.
  - **W3C Link Checker** - Checks the validity of links on a webpage.
  - **Xenu's Link Sleuth** - A legacy Windows tool that scans entire websites for broken links. Has the added bonus of working offline too.
  - The **built-in reports** of your authoring tool.
- **Compliance and accessibility**

- **axe Accessibility Checker** - Browser extension or CLI that tests web content for accessibility issues.
- **WAVE** - Web accessibility evaluation tool that provides visual feedback about accessibility problems.

Some of these tool need manual intervention and decisions; in other cases, testing can be automated as part of a CI/CD pipeline.

> Remember that you can always write your own custom tests too (either by yourself or with the help of a friendly neighborhood dev or AI tool).

4.4. Automation

By automating your processes, you can ensure consistency in your documentation, while at the same time reducing manual effort.

Automation can be:

- **Ad hoc** - quick, one-off solutions
- **Systematic** - repeatable processes integrated into a pipeline, with consistent triggers

This list provides some examples of ad hod and systematic automation scenarios:

- **Create a Jenkins job** that gets the latest documentation changes from source control, then builds and publishes the online help to a staging area.
- **Automatically identify and insert internal links** between related sections of documentation.
- **Use search & replace** in a text editor to change file paths as a result of a migration to a different tool.

- **Set up automated alerts or notifications** to inform stakeholders when documentation has been updated or requires review.
- **Send drafts for review to appropriate stakeholders** automatically, then track their approval or feedback.
- **Set up a pre-commit hook** in a Git repository that prevents you from committing files if you used inline formatting instead of HTML classes.

4.5. AI

Is AI going to replace technical writers? A definitive answer is impossible, but what is clear is that AI can't (for now) do *everything* a good technical writer can.

Rather than catastrophizing, we should try to understand it and use it where it can add value, freeing us for tasks where a human understanding is crucial.

GenAI, like ChatGPT or Copilot, can help with:

- **Content creation** - generate first drafts of documentation based on existing materials, saving time for technical writers.
  Archbee, through its **Documentation.new** tool, can even generate an entire website from a single prompt. You can find a more detailed overview of the platform in its dedicated page in this playbook: **Documentation.new**.
- **Editing** - analyze the content and suggest improvements in grammar, style and structure.

- **Search enhancement** - allows users to locate the most relevant content quickly, even if their queries are vague or ambiguous.
- **Automating tasks** - AI shines where there is structure; use it for things like converting tables from one format to another and even writing scripts to automate documentation processes.

Getting Results out of AI

To ensure the best results, it is essential to **train** AI tools with domain-specific knowledge. Tailoring AI models to the specific terminology and nuances of your product or industry will lead to more relevant and accurate documentation suggestions.

Prompting is also a skill that technical writers need to develop. Feeding a collection of specifications, bug reports and enhancement tickets into GenAI will result in a document that *seems* to make sense, but the garbage-in-garbage-out principle always applies: the output is only as good as the input.

A good technical writer will:

- **Instruct the AI model how to format the output.**
  This can include the tone to use, the length of the output, and you can even feed a mini-style guide into the prompt.
- **Include caveats.**
  It takes a human to notice that there are inconsistencies in the source materials and tell the AI to deal with them!
- **Review the initial output and refine it through further prompts.**
  For example, if the first draft uses a wrong term, you can tell the AI to rewrite it using the correct word.
- **Work in chunks, if needed.**
  A large document can be handled iteratively - first the introduction, then the procedures, etc.
- **Choose the scenarios wisely.**
  Converting a bulleted list into a Markdown table? Yes, this is exactly the manual task that GenAI can make trivial.

Extracting a 20-step procedure from 5 conflicting documents? A human brain is definitely needed to make sense of the intricacies.

Limitations of AI for Technical Writing

That being said, there are many aspects where **AI struggles**:

- **Generating documentation for proprietary software**
  Writers who document niche products that don't have public documentation have deep knowledge of the particularities of the software, which AI can't have. Getting a good documentation draft from an AI can be nearly impossible in this case, as the AI does not have any context about your product.

- **Chatbots can't answer without the underlying sources**
  If no one writes the documentation in the first place, chatbots can't use it. Answers from stale data or hallucinations are worse than no answer.

- **Understanding complex concepts**
  Writers can break down intricate technical details into clear, understandable content, something AI struggles with due to a lack of deep contextual understanding.

- **Audience-centric communication**
  Technical writers tailor content to suit the audience’s needs, adjusting tone and complexity, while AI may miss these nuances.

- **Contextual decision making**
  Writers consider the broader context, product evolution, and user needs when creating or updating documentation, while AI might overlook these elements.

- **Collaborating with SMEs**
  Writers work closely with subject-matter experts to clarify complex ideas, ensuring content is accurate and user-friendly, something AI cannot replicate in dynamic conversations.

- **Ethical content creation**
  Writers ensure documentation is inclusive and sensitive, avoiding potential ethical pitfalls, which is a challenge for AI without a moral framework.

- **Creative problem solving**
  Writers creatively address unforeseen issues in documentation, offering solutions that AI, based on patterns, cannot generate.

4.6. Documentation.new

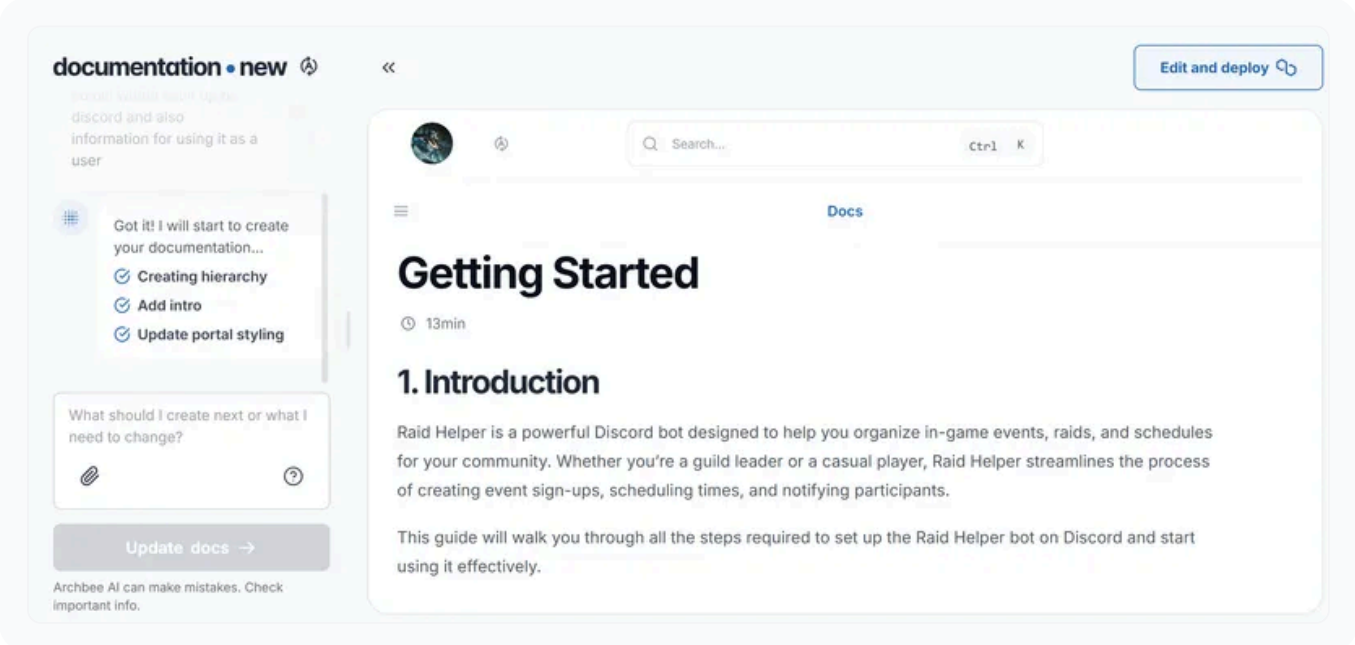
Like with any writer, the blank page (or computer screen)

is a technical writer's enemy.

The best advice? Just start with something. It's called a "draft" for a reason!

Unless you're really unlucky, you'll usually have some materials to begin with, maybe a specification document, a few tickets in the bug tracking system, notes from a meeting with an engineer, an OpenAPI file, or a readme. Once you've gathered everything available, you're ready to produce the first draft.

And, since we're in 2025, you don't have to do it all manually. Archbee's Documentation.new was built to help you skip the slow start. Just hand it your materials—upload files like Markdown, YAML, JSON, DOCX, or point it to a URL or write a short prompt and it will generate your v0 documentation in seconds. Bonus: it builds the documentation site for you too.



The more specific your input, the better the output. Refine your prompt to guide the structure or content focus, and Documentation.new will generate accordingly.

Once you've got that first version, you can, and probably should, go further: revise, rewrite, and continue prompting until the result feels right for your audience.

> You can find some tips and tricks about prompting in the **AI** section.

Once you're happy with your initial documentation, it's just one click to deploy it into your Archbee space. From there, you can invite collaborators, build out sections, and polish everything for publication.

Starting documentation doesn't have to be painful. With a little input, Documentation.new gets you to your first usable draft, fast.

5. Conclusion

So far, this playbook has shown technical writing the way it should be, in a company that knows how to appreciate what we bring to the table.

Things are not always rosy, though, and this guide would not be complete without the dark side.

Oftentimes, documentation is an afterthought and the technical writer is seen as "less than", just an editor, someone who can be overlooked. Sometimes processes are chaotic and the technical writer is never notified of changes. Less-technical technical writers can be looked down upon by engineers.

But when you're in a good place with good people? Technical writing is fun.

You get to learn new things every day.

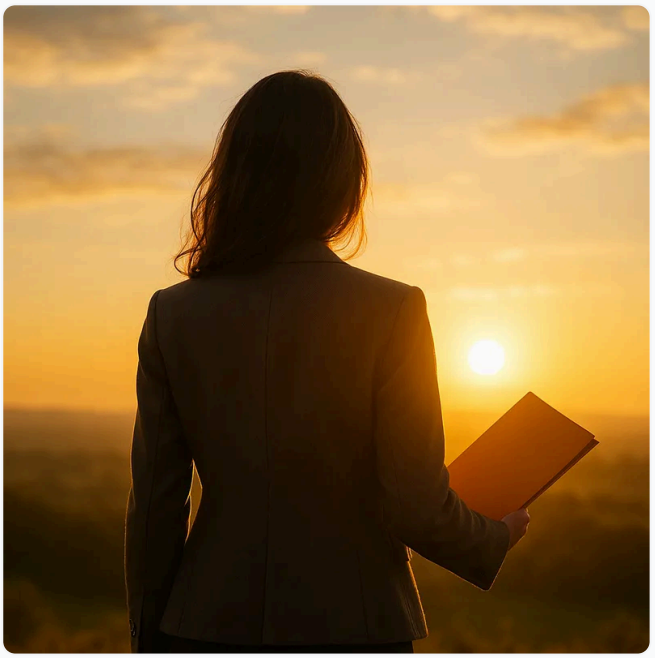
You get paid to care about grammar and style.

You get to contribute a different point of view, maybe even find scenarios that no one thought of.

You get to be the link between the teams, gathering information from every source and bringing it together.

You get to support users achieve their goals without frustration. Even more satisfying when the user is internal and you can get that instant feedback!

So if you think you fit the bill and this would be a job you'd enjoy, why not give it a try?



Written by Ioana Stanescu

Technical Documentation Manager