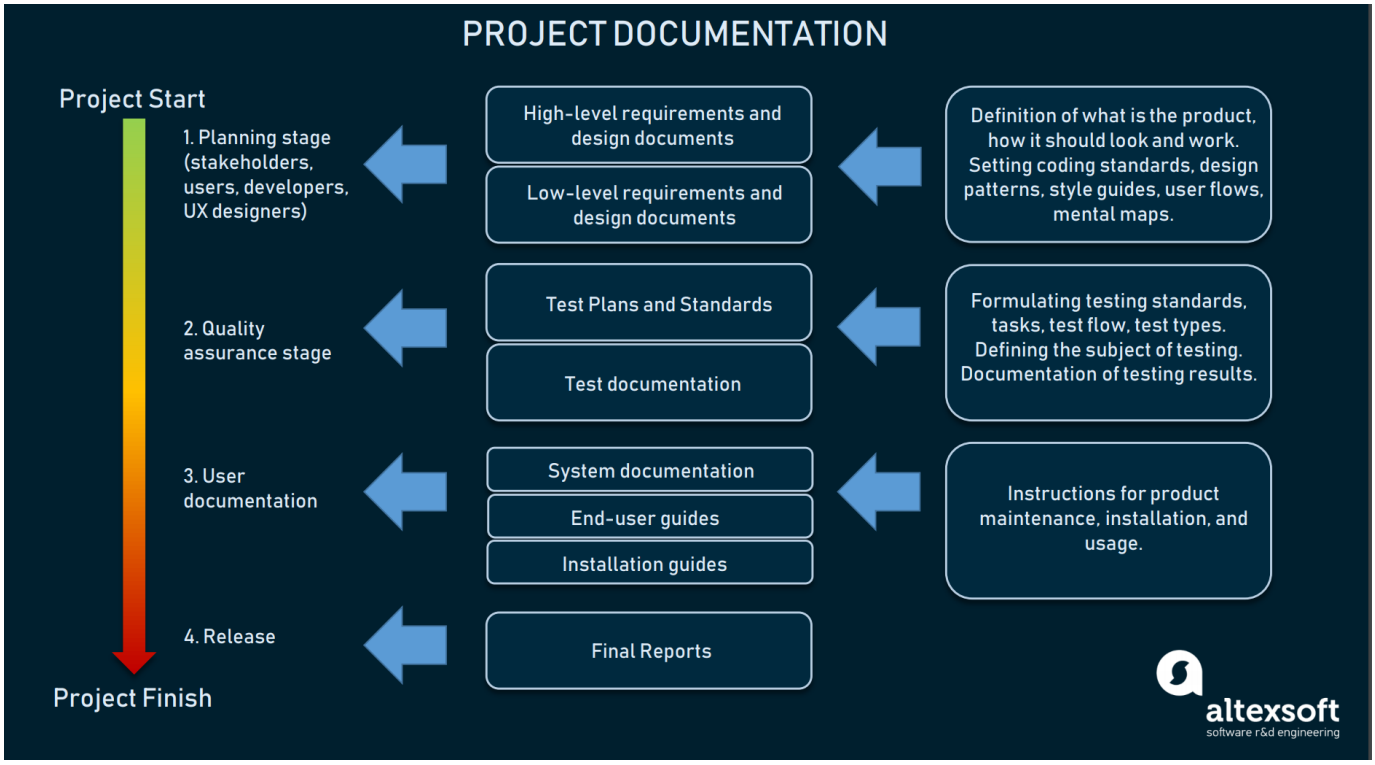
Last Updated: 01 Dec, 2020

Technical Documentation in Software Development: Types, Best Practices, and Tools

Reading time: 22 minutes

Technical documentation in software engineering is the umbrella term that encompasses all written documents and materials dealing with software product development. All software development products, whether created by a small team or a large corporation, require some related documentation. And different types of documents are created through the whole software development lifecycle (SDLC). Documentation exists to explain product functionality, unify project-related information, and allow for discussing all significant questions arising between stakeholders and developers.



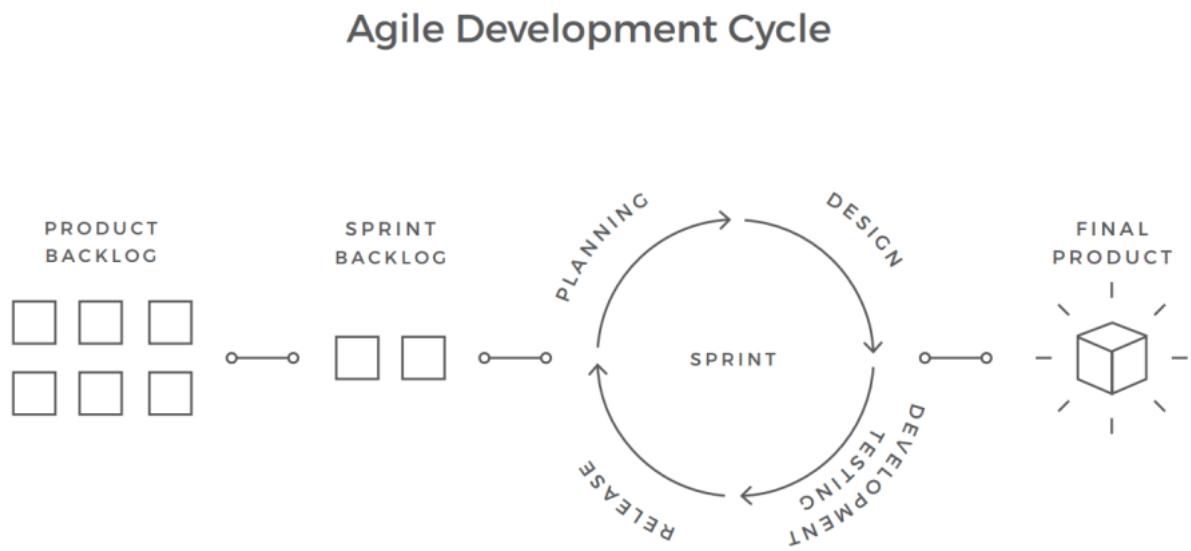
Project documentation by stages and purpose

On top of that, documentation errors can set gaps between the visions of stakeholders and engineers and, as a result, a proposed solution won't meet stakeholders expectations. Consequently, managers should pay a lot of attention to documentation quality.

Agile and waterfall approaches

The documentation types that the team produces and its scope depending on the software development approach that was chosen. There are two main ones: agile and waterfall. Each is unique in terms of accompanying documentation.

The Waterfall approach is a linear method with distinct goals for each development phase. Teams that use waterfall spend a reasonable amount of time on product planning in the early stages of the project. They create an extensive overview of the main goals and objectives and plan what the working process will look like. Waterfall teams strive to create detailed documentation before any of the engineering stages begin. Careful planning works well for projects with little to no changes in progress as it allows for precise budgeting and time estimates. However, waterfall planning has proven to be ineffective for long-term development as it doesn't account for possible changes and contingencies on the go. According to KPMG Global Agile Survey, 81% of companies initiated their Agile transformation in the last three years.



Agile development cycle scheme

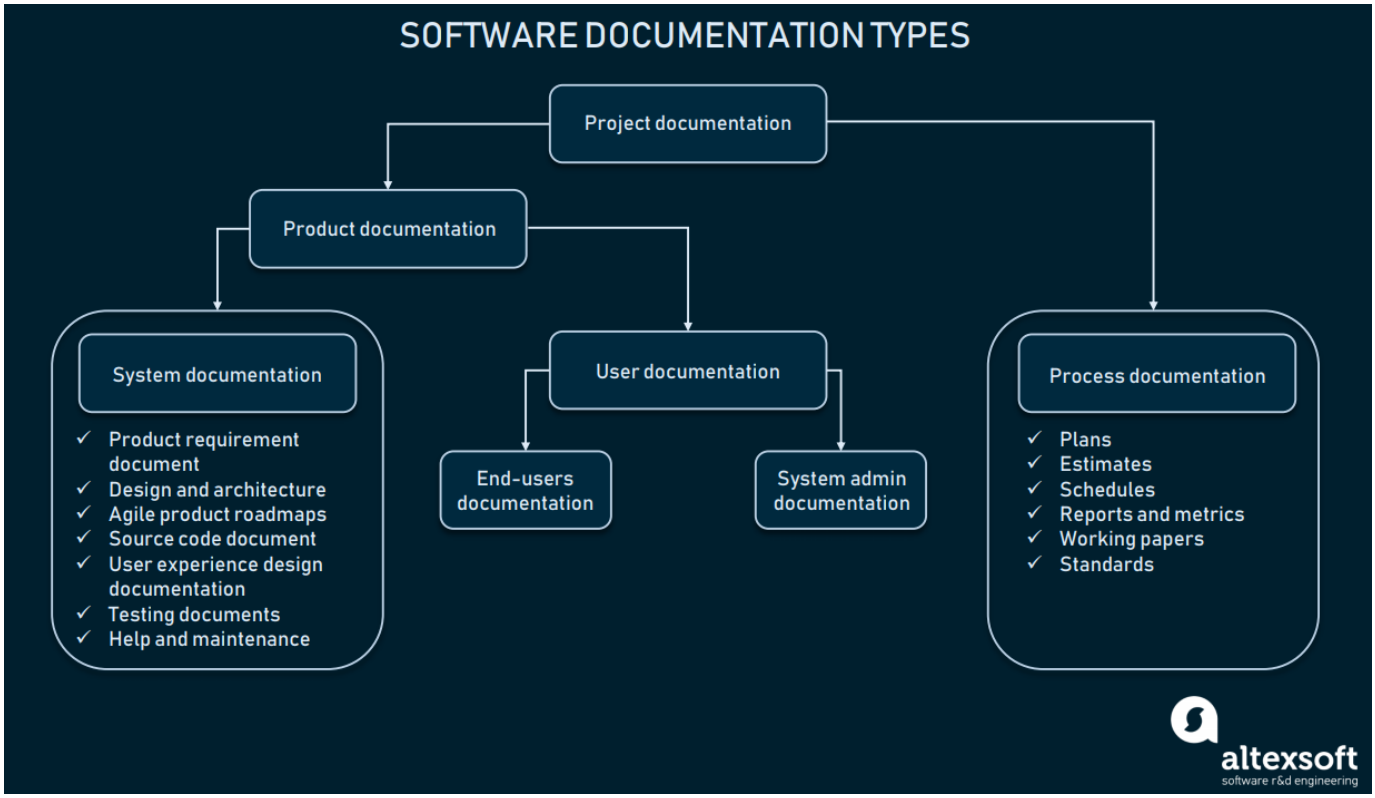
The agile approach is based on teamwork, close collaboration with customers and stakeholders, flexibility, and ability to quickly respond to changes. The basic building blocks of agile development are iterations; each one of them includes planning, analysis, design, development, and testing. The agile method doesn't require comprehensive documentation at the beginning. Managers don't need to plan much in advance because things can change as the project evolves. This allows for just-in-time planning. As one of the Agile Manifesto values suggests, putting – "working software over comprehensive documentation -", the idea is to produce documentation with information that is essential to move forward, when it makes the most sense.

Today, agile is the most common practice in software development, so we'll focus on documentation practices related to this method.

Types of documentation

The main **goal** of effective documentation is to ensure that developers and stakeholders are headed in the same direction to accomplish the objectives of the project. To achieve them, plenty of documentation types exist.

Adhering to the following classifications.



Software documentation most commonly used in Agile projects

All software documentation can be divided into **two main categories:**

- **Product documentation**
- **Process documentation**

Product documentation describes the product that is being developed and provides instructions on how to perform various tasks with it. In general, product documentation includes requirements, tech specifications, business logic, and manuals. There are two main types of product documentation:

- System documentation represents documents that describe the system itself and its parts. It includes requirements documents, design decisions, architecture descriptions, program source code, and FAQs.
- User documentation covers manuals that are mainly prepared for end-users of the product and system administrators. User documentation includes tutorials, user guides, troubleshooting manuals, installation, and reference manuals.

Process documentation represents all documents produced during development and maintenance that describe... well, the process. The common examples of process-related documents are standards, project documentation, such as project plans, test schedules, reports, meeting notes, or even business correspondence.

The main difference between process and product documentation is that the first one records the process of development and the second one describes the product that is being developed.

Product: System documentation

System documentation provides an overview of the system and helps engineers and stakeholders understand the underlying technology. It usually consists of the requirements document, architecture design, source code, validation docs, verification and testing info, and a maintenance or help guide. It's worth emphasizing that this list isn't exhaustive. So, let's have a look at the details of the main types.

Product requirement document

A product requirement document or PRD provides information about system functionality. Generally, requirements are the statements of what a system should do. It contains business rules, user stories, use cases, etc. This document should be clear and shouldn't be an extensive and solid wall of text. It should contain enough to outline the product's purpose, its features, functionalities, maintenance, and behavior.

The best practice is to write a requirement document using a single, consistent template that all team members adhere to. The one web-page form will help you keep the document concise and save the time spent on accessing the information. Here's a look at an example of a one-web-page product-requirements document to understand various elements that should be included in your PRD. Nevertheless, you should remember that this isn't the one and only way to compile this document.

Mobile Web Requirements

Created by Mitch Davis, last modified just a moment ago

Target release	1.0
Epic	MDT-18 - Mobile optimized web app TO DO
Document status	DRAFT
Document owner	@Mitch Davis
Designer	@Cassie Owens
Developers	@Harvey Jennings
QA	@Kevin Campbell

Background and strategic fit

We all know mobile is on the rise. A recent survey to customers showed that 85% of users use their mobile on a daily basis. Most of our customers also use competitor apps, so this is something we need to have.

Customer research

- Customer interview - Netflix
- Customer interview - Homeaway
- Customer interview - Bitbucket

Requirements

#	User story title	User story description	Priority	Notes
1	Facebook Integration MDT-13 TO DO	A user wants to sign up via Facebook	Must Have	• We will need to talk to Cassie Owens. • There has also been some research done on this (see Facebook integration prototype)
2	Activity Stream MDT-14 TO DO	A user wants to view the latest updates via the mobile dashboard so that they can get a better understanding of what is in place	Must Have	
3	Post Updates MDT-15 TO DO	A user wants to be able to post status updates on the go	Must Have	The key things we will need to support: • Text status updates • Mentions • Support for images • Smart embedding for YouTube vids
4	API MDT-16 TO DO	A developer wants to integrate with the mobile app so that they can embed the activity stream on their website	Should Have	• We should chat to Team Dyno as they did something similar.

User interaction and design

Description	Login screen	Activity stream
Mockup		

Questions

Below is a list of questions to be addressed as a result of this requirements document:

Question	Outcome
What about Google Apps	• We think this is important, but not for version one. • We can look at this at a later stage. • It might be worth someone looking into a shared notification library to do this.
Are we supporting Blackberry?	• Again, not for initial version - but we haven't had much demand for this.
Should we have an offline mode?	• We've talked about the pros and cons. In brief: Seamless experience for customers, they won't notice if there is a connection issue Most of our competitors don't have this Could be expensive to build Should we spike this at a later sprint?

Not Doing

- Google Apps Authentication - out of scope, see above for details
- Blackberry support - we won't look at doing this, if demand picks up we can look at it.
- Native app. We are starting with a mobile web view first and get back to a native app depending on feedback that we get.

Like Be the first to like this requirements

Technical documentation example: One web-page software requirements document created by using Atlassian Confluence, the content collaboration software

Here are the main recommendations points to include in your product requirement document:

1. **Roles and responsibilities**. Start your document with the information about project participants including a product owner, team members, and stakeholders. These details will clarify responsibilities and communicate the target release goals for each of the team members.
2. **Team goals and business objective**. Define the most important goals in a short point form.
3. **Background and strategic fit**. Provide a brief explanation of the strategic aim of your actions. Why are you building the product? How do your actions affect product development and align the company's goals?

4. **Assumptions.** Create a list of technical or business assumptions that the team might have.
5. **User Stories.** List or link user stories that are required for the project. A user story is a document written from the point of view of a person using your software product. The user story is a short description of customer actions and results they want to achieve.
6. **Acceptance criteria.** Those are the conditions that indicate a user story is completed. The main purpose of acceptance criteria is to define a satisfactory result for a usage scenario from the end-user perspective. Check our dedicated article on acceptance criteria to learn more.
7. **User interaction and design**. Link the design explorations and wireframes to the page.
8. **Questions.** As the team solves the problems along the project progression, they inevitably have many questions arising. A good practice is to record all these questions and track them.
9. **Not doing.** List the things which you aren't doing now but plan on doing soon. Such a list will help you organize your teamwork and prioritize features.

Make all this information more comprehensive by using the following practices:

- Use **links and anchors**. They will help you make the document easier to read and search as readers will be able to comprehend the information gradually. For instance, you can provide links to customer interviews and anchors to previous discussions or other external information related to the project.
- Use **graphics and diagramming tools** to better communicate the problems to your team. People are more likely to perceive information by looking at the images than reading an extensive document. Different visual models will help you to perform this task and outline requirements more effectively. You can incorporate diagrams into your requirements process using the following software diagramming tools: Visio, Gliffy, Balsamiq, Axure or SmartArt in Microsoft Office.

User Experience Design documentation

User experience design begins at the requirements stage and proceeds through all the stages of development, including the testing and post-release stages. The process of UX design includes research, prototyping, usability testing, and the actual designing part, during which lots of documentation and deliverables are produced.

The UX documentation can be divided into stages. The research stage includes:

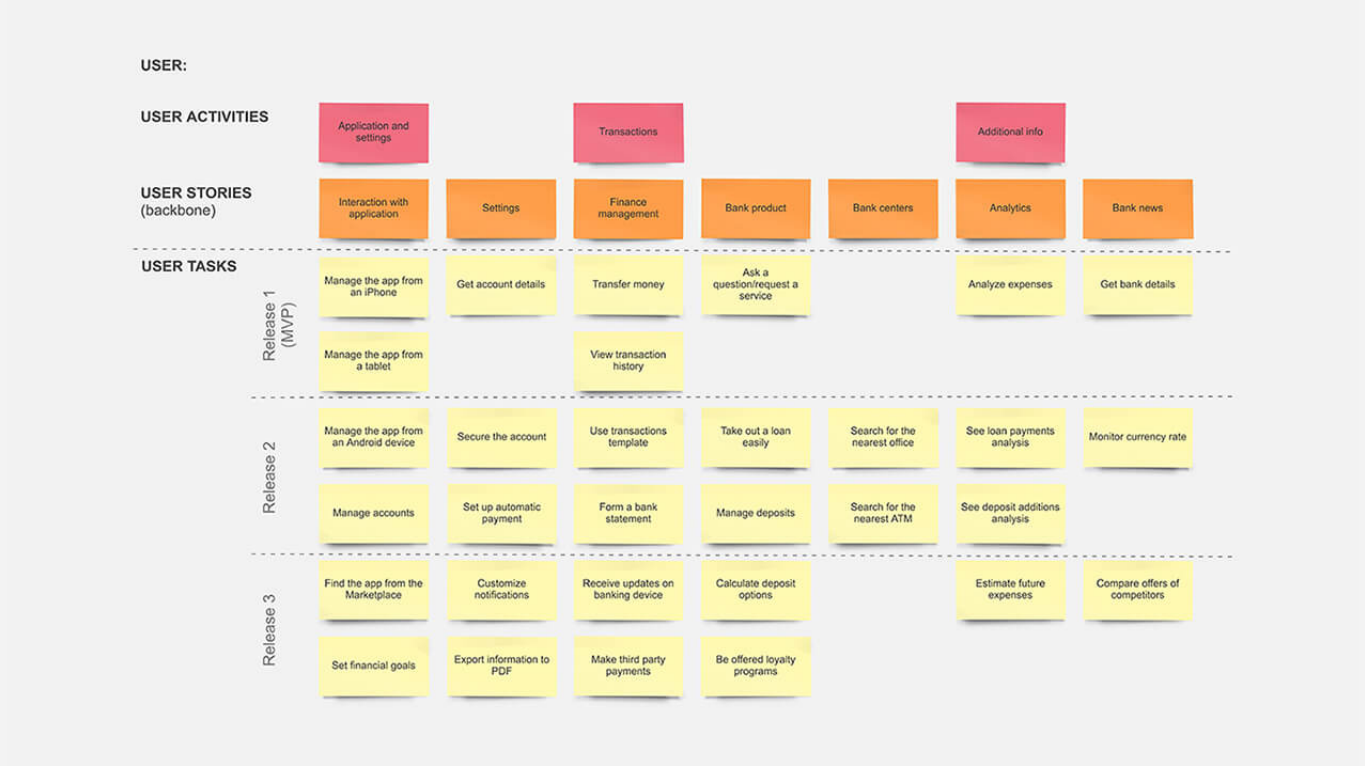
- User personas
- User scenario
- Scenario map
- User story map
- UX style guide

User Personas are created and documented during the research stage. The information gathered during user interviews and surveys is compiled into functional user persona documents. User personas represent the key characteristics of real users, focusing on behavior, thought patterns, and motivation.

A **user scenario** is a document that describes the steps a user persona will take to accomplish a specific task. User scenarios focus on what a user will do, rather than outlining the thought process. The set of scenarios can be either visual or narrative, and describe the existing scenarios or future functionality.

Scenario maps are used to compile the existing user scenarios into a single document. Scenario maps show all possible scenarios available at a given moment. The main purpose of a scenario map is to depict all the possible scenarios for every single function, as well as intersecting scenario steps.

A **user story map** is formed from the backlog of the product. This type of document helps to arrange the user stories into future functions or parts of the application. A user story map can be a scheme, or a table of user stories grouped in a particular order to denote the required functions for a certain sprint.



An example of a user story map broken down into releases

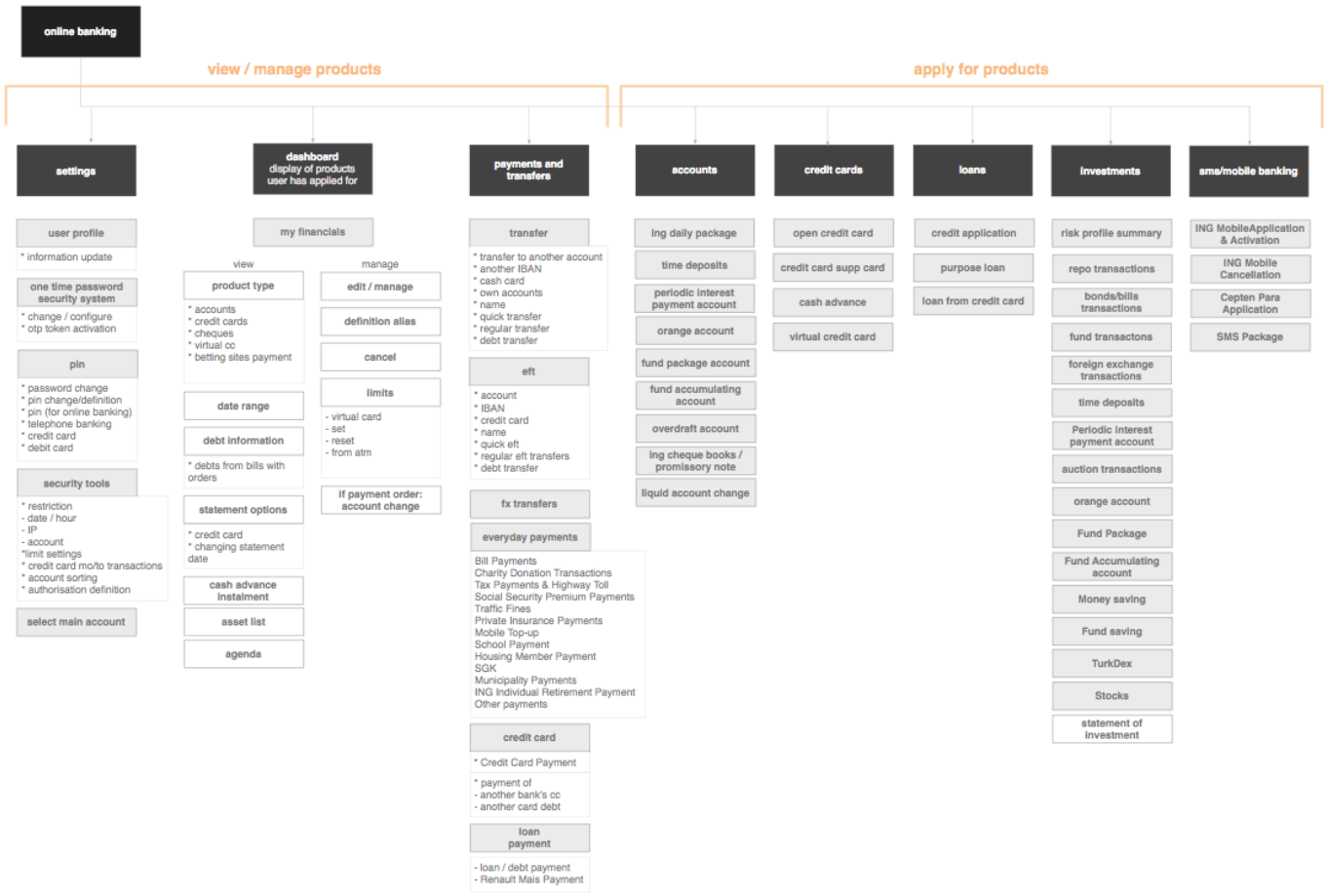
Source: feedotter.com

The **UX style guide** is a document that includes the design patterns for the future product. It also describes all possible UI elements and content types used, defining the rules of how they should b arranged and work with each other. But, unlike a UI style guide, UX designers don't describe th actual look of the interface.

On the stage of **prototyping** and **designing**, a UX designer often works with the deliverables and updates documentation on par with other team members, including product owner, UI designers, and development team. The most common documents produced at these stages are:

- Site maps
- Wireframes
- Mock-ups
- Prototypes
- User flow schemes or user journey
- Usability testing reports

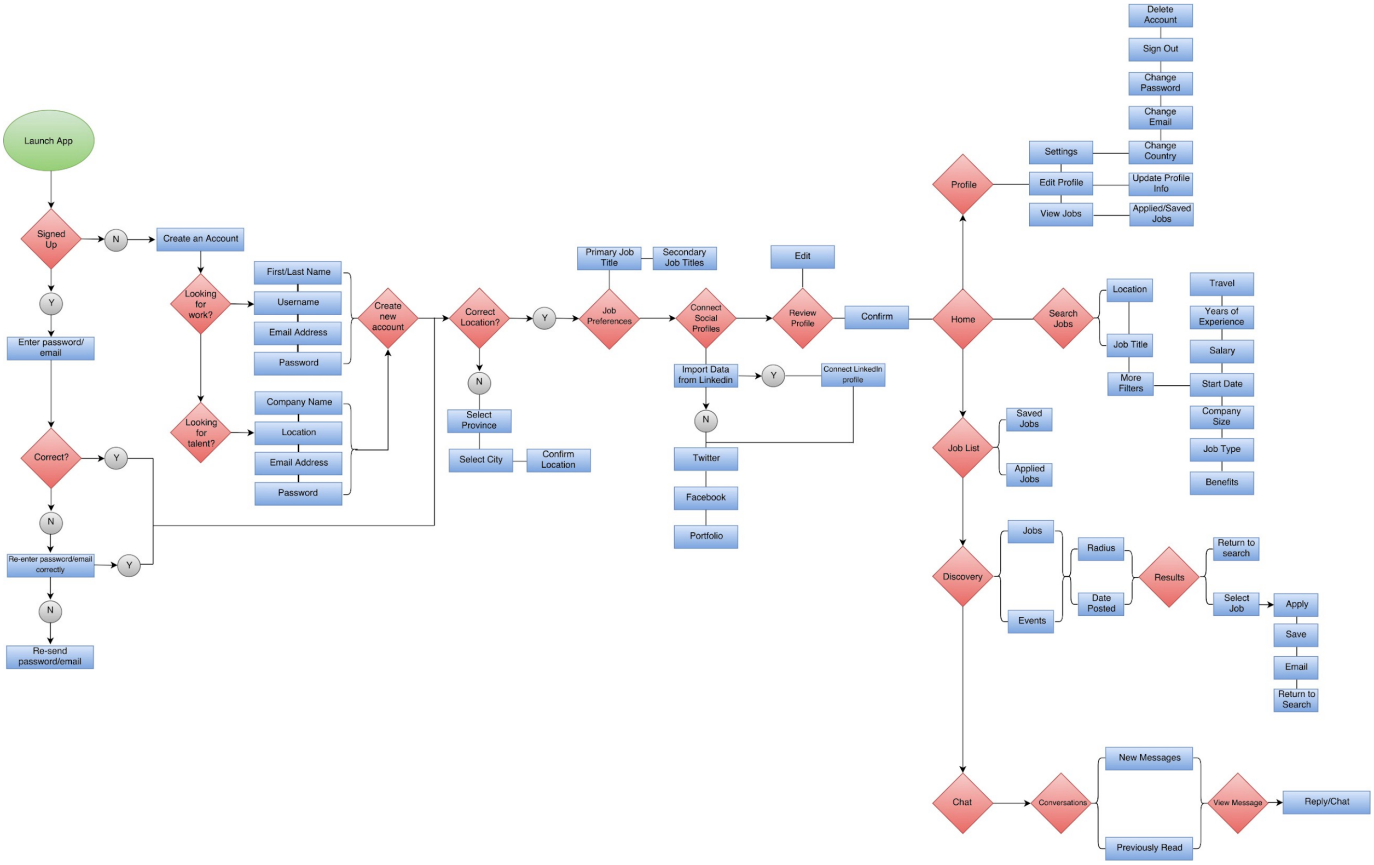
A **site/product map** is a visual scheme that represents the connection between all pages of a product. The map helps the whole team visualize the structure of a website or app and the connections between the pages/functions. Creating a site map is a part of arranging the information architecture.



Site map structure example

Source: uxforthemas

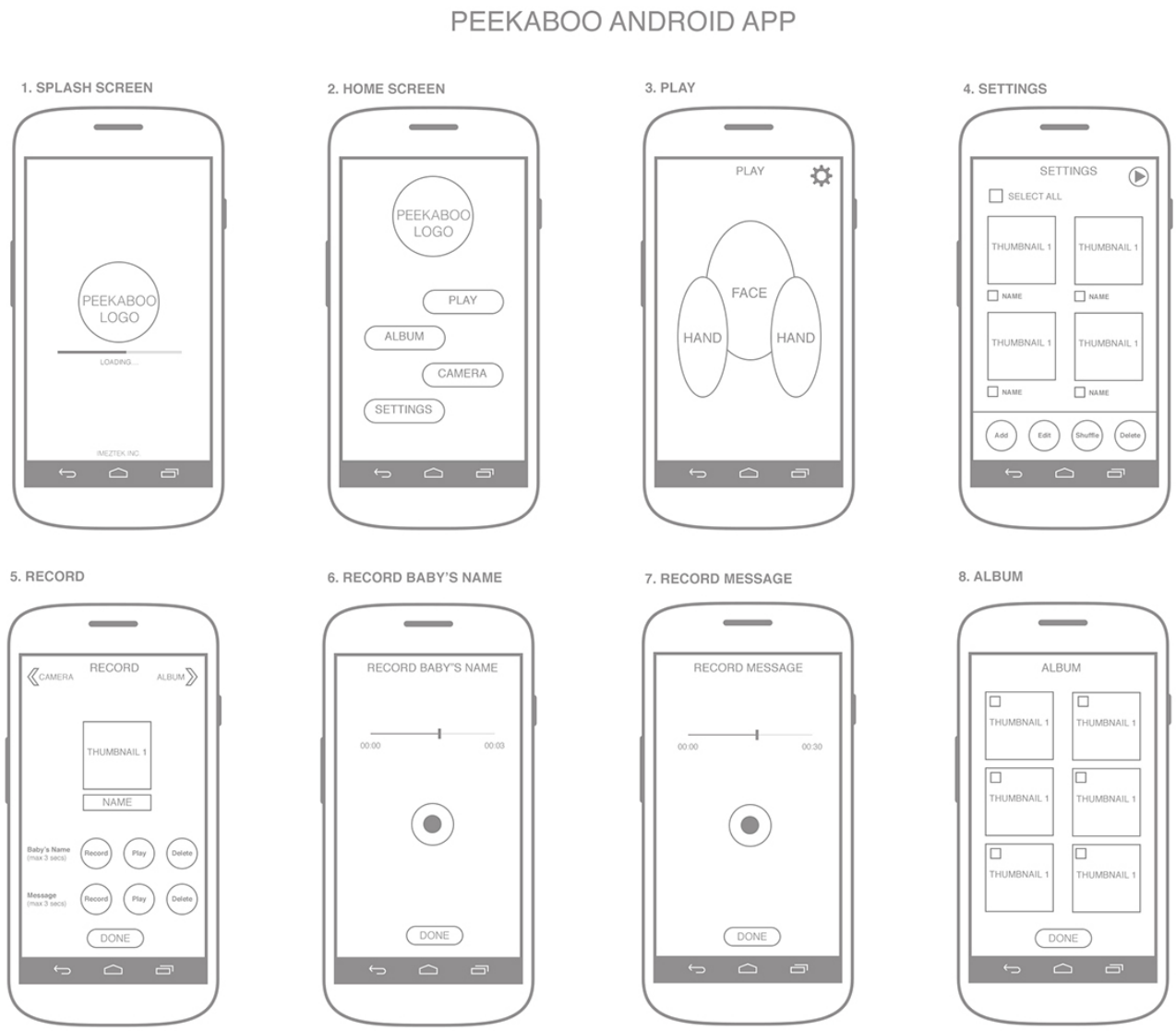
User flow or **user journey schemes** help the team to map the steps a user should take through the whole product. The main task of a user flow scheme is to depict the possible steps a user may take, going from page to page. Usually, the scheme includes all the pages, sections, buttons, and functions they provide to show the logic of user movement.



Job search application user flow scheme

Source: medium.com

Wireframes are the blueprints for future UI. Basically, wireframes are the schemes that show how to arrange the elements on the page and how they should behave. But, wireframes don't depict what those elements should look like.



Wireframe example for Peekaboo mobile app

A **mock-up** is the next product design stage, showing the actual look and feel of a product. Basically, mock-ups are static images representing the final product design.

A **prototype** is a mock-up that you can interact with: click some buttons, navigate between different pages, and so on. A prototype can be created in a prototyping tool like Sketch or MockFlow. Using templates, UX designers can create interactive mock-ups on the early stages of development to be employed for usability testing.

A **usability testing report** is a short-form feedback document created to communicate the results of usability testing. The report should be as short as possible, with visual examples prevailing over text.

Software architecture design document

Software architecture design documents, sometimes also called technical specifications, incluc main architectural decisions made by the solution architect. Unlike the product requirement

document mentioned above that describes *what* needs to be built, the architecture design documentation is about *how* to build it. It has to describe in what way each product component will contribute to and meet the requirements, including solutions, strategies, and methods to achieve that. So, the software design document gives an overview of the product architecture, determines the full scope of work, and sets the milestones, thus, looping in all the team members involved and providing the overall guidance.

We don't recommend going too much into detail and listing all the solutions to be used, but rather focus on the most relevant and challenging ones. An effective design and architecture document comprises the following information sections:

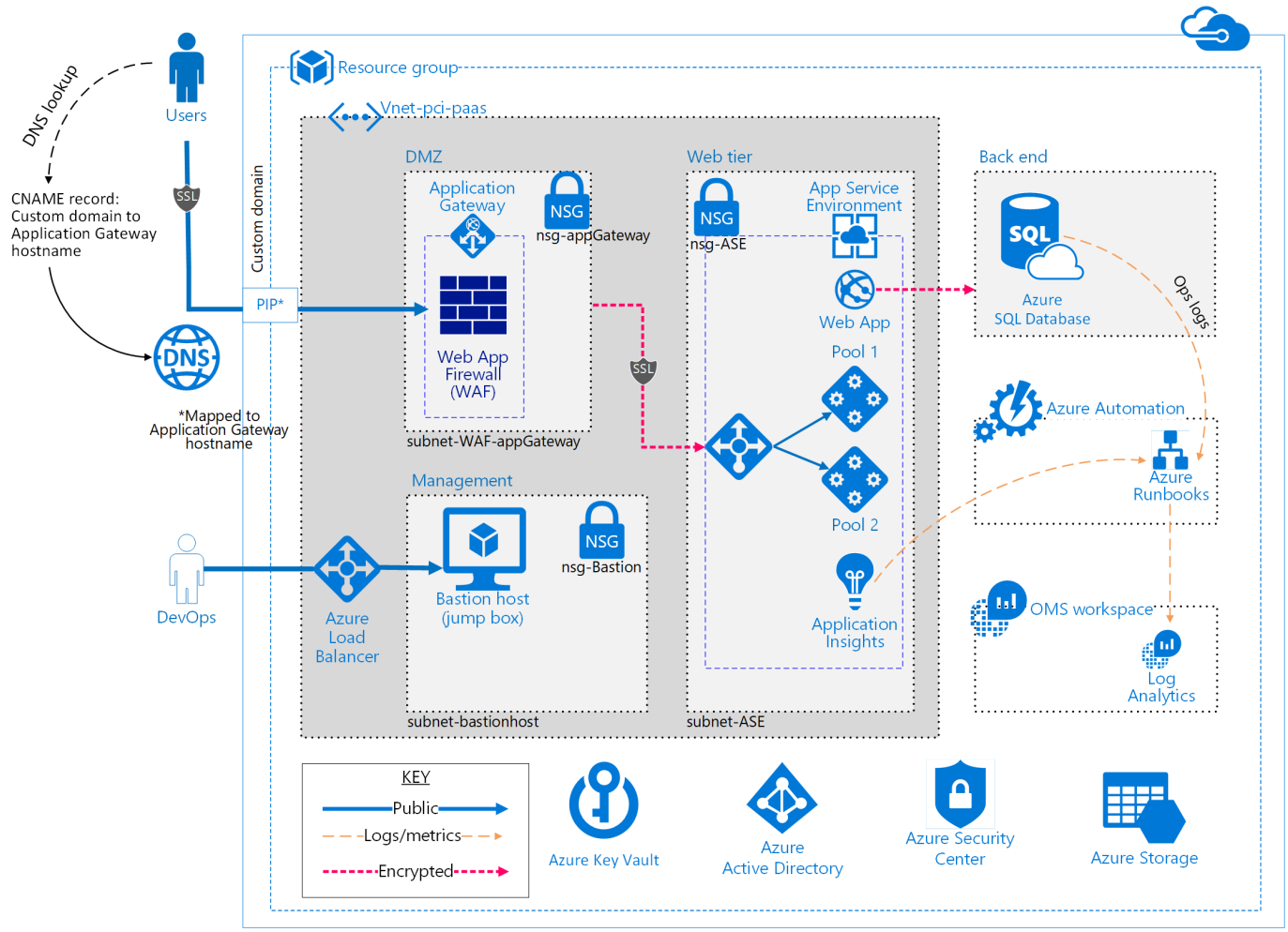
Overview and background. Briefly describe the main goals of the project, what problems you are trying to solve, and the results you want to achieve.

Architecture & Design Principles. Underline the guiding architecture and design principles with which you will engineer the product. For instance, if you plan to structure your solution using microservices architecture, don't forget to specifically mention this.

User Story description. Connect user stories with associated business processes and related scenarios. You should try to avoid technical details in this section.

Solution details. Describe the contemplated solution by listing planned services, modules, components, and their importance.

Diagrammatic representation of the solution. Provide the diagrams and/or other graphic materials to help understand and communicate the structure and design principles.



Azure web application architecture diagram

Source: docs.microsoft.com

Milestones. Include the overall timeline, deadlines for completion, and/or functional milestones, i.e., independent modules of the application developed. That will help organize the work process and provide a clear metric to monitor progress. This section can be very brief as it's closely related to the process documentation described below.

Source code document

A source code document is a technical section that explains how the code works. While it's not necessary, the aspects that have the greatest potential to confuse should be covered. The main users of the source code documents are software engineers.

Source code documents may include but are not limited to the following details:

- HTML generation framework and other frameworks applied
- Type of data binding
- Design pattern with examples (e.g. model-view-controller)
- Security measures

- Other patterns and principles

Try to keep the document simple by making short sections for each element and supporting them with brief descriptions.

Quality assurance documentation

There are different types of testing documents in agile. We have outlined the most common:

- Quality management plan
- Test strategy
- Test plan
- Test case specifications
- Test checklists

A **quality management plan** is an analog of a requirement document dedicated to testing. This document sets the required standard for product quality and describes the methods to achieve this level. The plan helps to schedule QA tasks and manage testing activity for product managers, but, it is mainly used for large-scale projects.

A **test strategy** is a document that describes the software testing approach to achieve testing objectives. This document includes information about team structure and resource needs along with what should be prioritized during testing. A test strategy is usually static as the strategy is defined for the entire development scope.

A **test plan** usually consists of one or two pages and describes what should be tested at a given moment. This document should contain:

- The list of features to be tested
- Testing methods
- Timeframes
- Roles and responsibilities (e.g. unit tests may be performed either by the QA team or by engineers)

A **test case specifications** document is a set of detailed actions to verify each feature or functionality of a product. Usually, a QA team writes a separate specifications document for each product unit. Test case specifications are based on the approach outlined in the test plan. A good practice is to simplify specifications description and avoid test case repetitions.

Test checklist is a list of tests that should be run at a particular time. It represents what tests completed and how many have failed. All points in the test checklists should be defined correctly

to group test points in the checklists. This approach will help you keep track of them during your work and not lose any. If it helps testers to check the app correctly, you can add comments to your points on the list.

Maintenance and help guide

This document should describe known problems with the system and their solutions. It also should represent the dependencies between different parts of the system.

API documentation

Nearly any product has its APIs or Application Programming Interfaces. Their documentation informs developers how to effectively use and connect to the required APIs.

API documentation is a deliverable produced by technical writers as tutorials and guides. This type of documentation should also contain the list of all available APIs with specs for each one.

Product: User documentation

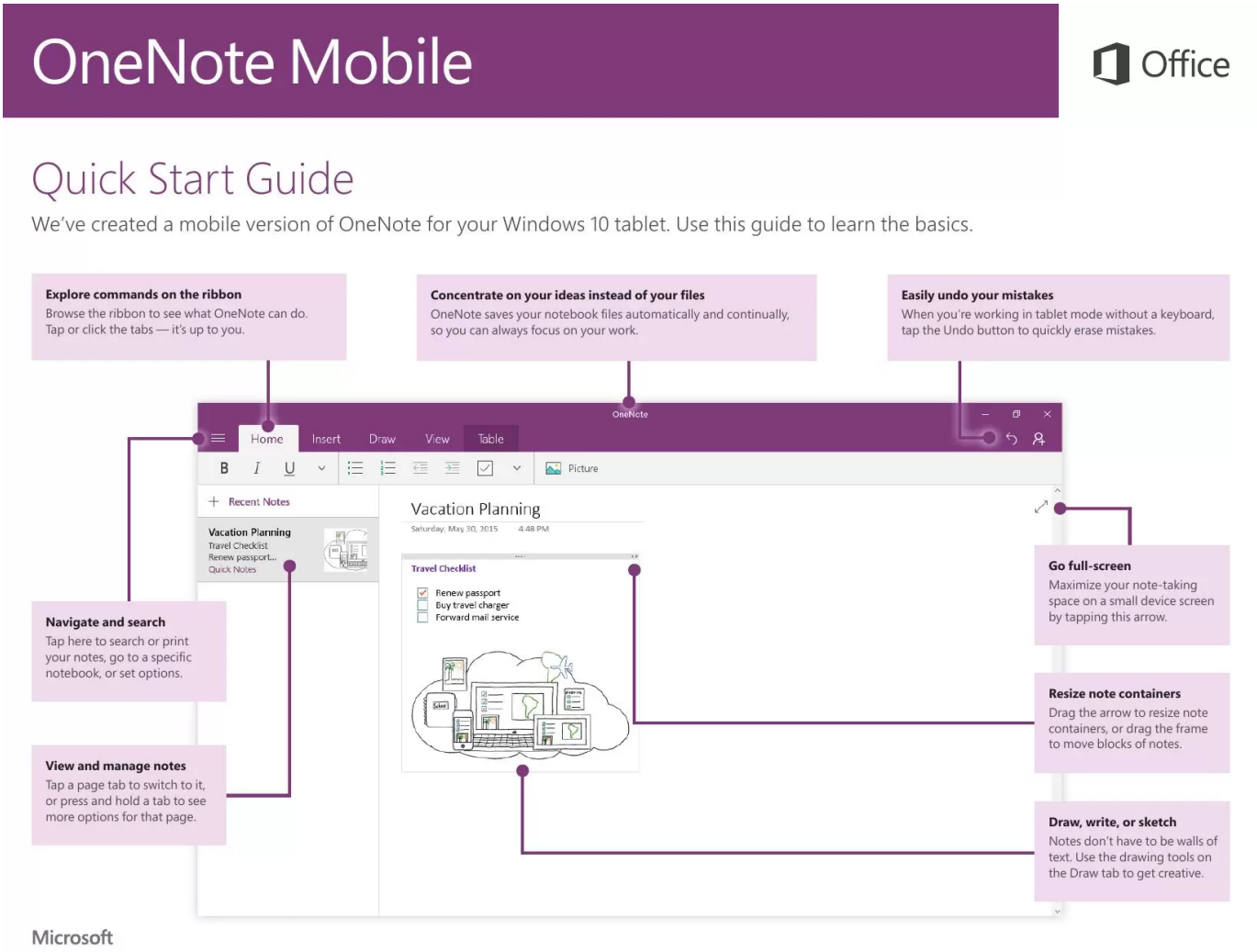
As the name suggests, user documentation is created for product users. However, their categories may also differ. So, you should structure user documentation according to the different user tasks and different levels of their experience. Generally, user documentation is aimed at two large categories:

- end-users
- system administrators

End-user documentation

The documentation created for end-users should explain in the simplest way possible how the software can help solve their problems. Such user instructions can be provided in the printed form, online, or offline on a device. Here are the main types of the user documents:

The quick start guide provides an overview of the product's functionality and gives basic guidelines on how to use it.



OneNote quick start guide, source: slideshare

The complete manual includes exhaustive information and instructions on how to install and operate the product. It lists the hardware and software requirements, detailed description of the features and full guidelines on how to get the most out of them, example inputs and outputs, possible tips and tricks, etc.;

The troubleshooting guide gives end-users information on how to find and resolve possible issues that might arise when using the product.

Some parts of user documentation, such as tutorials and onboarding, in many large customer-based products are replaced with onboarding training. Nevertheless, there are still complex systems remaining that require documented user guides.

User documentation requires technical writers to be more imaginative. Online end-user documentation may include the following sections:

- FAQs
- Video tutorials

- Embedded assistance
- Support Portals

Since user documentation is a part of customer experience, it's important to make it easy to understand and logically structured. Written in plain language with visual materials and step-by-step instructions included, user guides can become a powerful marketing tool and increase customer satisfaction and loyalty.

Besides, to provide the best service for end-users, you should collect your customer feedback continuously. The wiki system is one of the more useful practices. It helps to maintain the existing documentation. If you use the wiki system you won't need to export documents to presentable formats and upload them to the servers. You can create your wiki pages using a wiki markup language and HTML code.

System administrators' documentation

System administrators' documents don't need to provide information about how to operate the software. Usually, administration docs cover installation and updates that help a system administrator with product maintenance. Here are standard system administrators documents:

- **Functional description** – describes the functionalities of the product. Most parts of this document are produced after consultating a user or an owner.
- **System admin guide** – explains different types of behaviors of the system in different environments and with other systems. It also should provide instructions on how to deal with malfunction situations.

Process Documentation

Process documentation covers all activities surrounding product development. The value of keeping process documentation is to make development more organized and well-planned. This branch of documentation requires some planning and paperwork both before the project starts and during the development. Here are common types of process documentation:

Plans, estimates, and schedules. These documents are usually created before the project starts and can be altered as the product evolves.

Reports and metrics. Reports reflect how time and human resources were used during development. They can be generated on a daily, weekly, or monthly basis. Consult our article on agile delivery metrics to learn more about process documents such as velocity chats, sprint burndown charts, and release burndown charts.

Working papers. These documents exist to record engineers' ideas and thoughts during project implementation. Working papers usually contain some information about an engineer's code, sketches, and ideas on how to solve technical issues. While they shouldn't be the major source of information, keeping track of them allows for retrieving highly specific project details if needed.

Standards. The section on standards should include all coding and UX standards that the team adheres to along the project's progression.

The majority of process documents are specific to the particular moment or phase of the process. As a result, these documents quickly become outdated and obsolete. But they still should be kept as part of development because they may become useful in implementing similar tasks or maintenance in the future. Also, process documentation helps to make the whole development more transparent and easier to manage.

The main goal of process documentation is to reduce the amount of system documentation. In order to achieve this, write the minimal documentation plan. List the key contacts, release dates, and your expectations with assumptions.

Agile product roadmaps

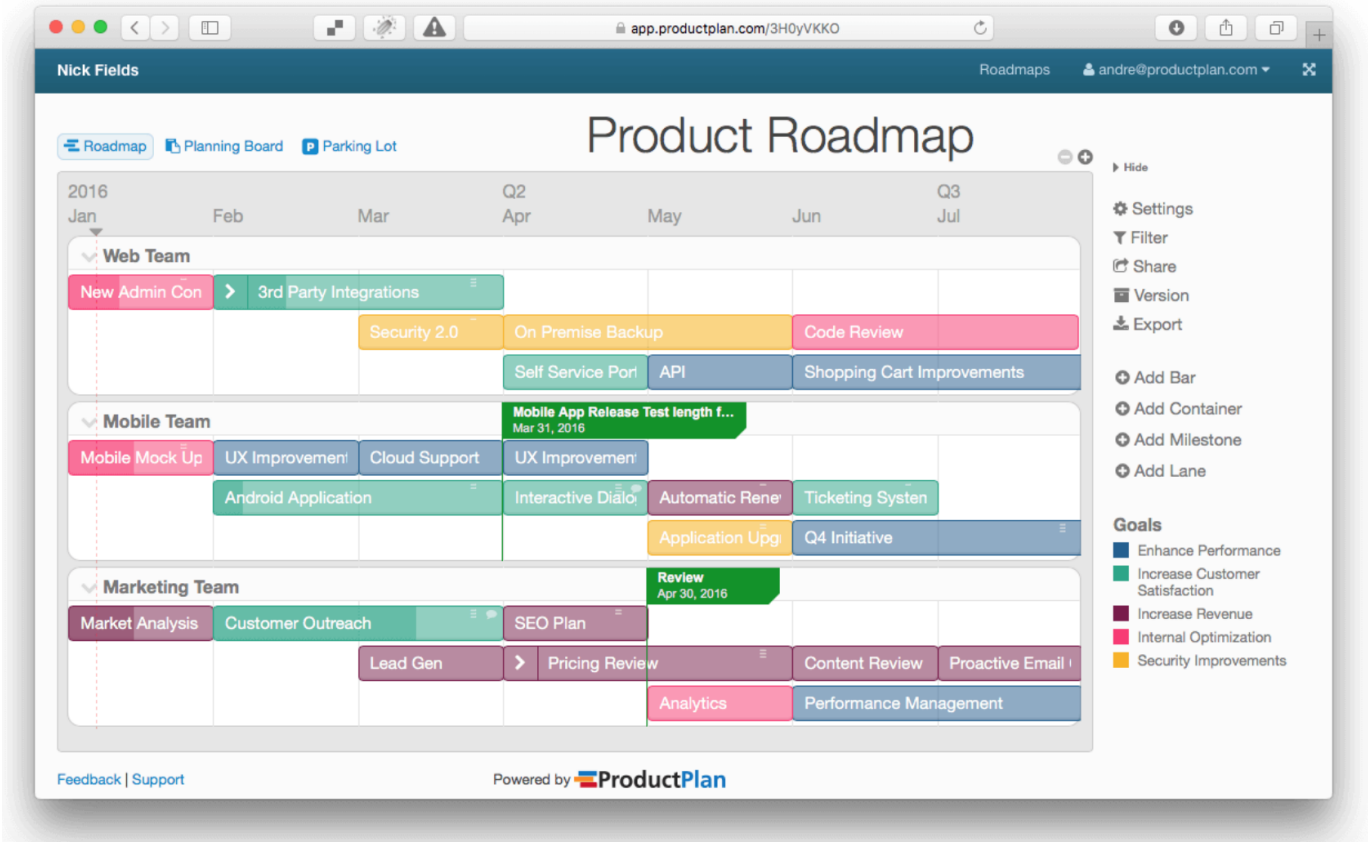
Product roadmaps are used in Agile software development to document vision, strategy, and overall goals of the project. Roadmaps are used as process documents to keep the course of development in sync with initial goals. Depending on the type of product roadmap, it can express high-level objectives, prioritization of tasks, the sprint timeline, or low-level details.

There are three types of product roadmaps that are used by Agile product teams:

- Strategic roadmap
- Technology or IT roadmap
- Release plan

A **strategic roadmap** is a high-level strategic document, that contains overall information on the project. Strategic roadmaps usually state a vision and long-term goals. In the case of agile product development, a roadmap can be arranged in themes. Themes are multiple tasks that a team must complete and are somehow connected. For instance, a theme may sound like "enhance page-loading speed," which entails a handful of actions.

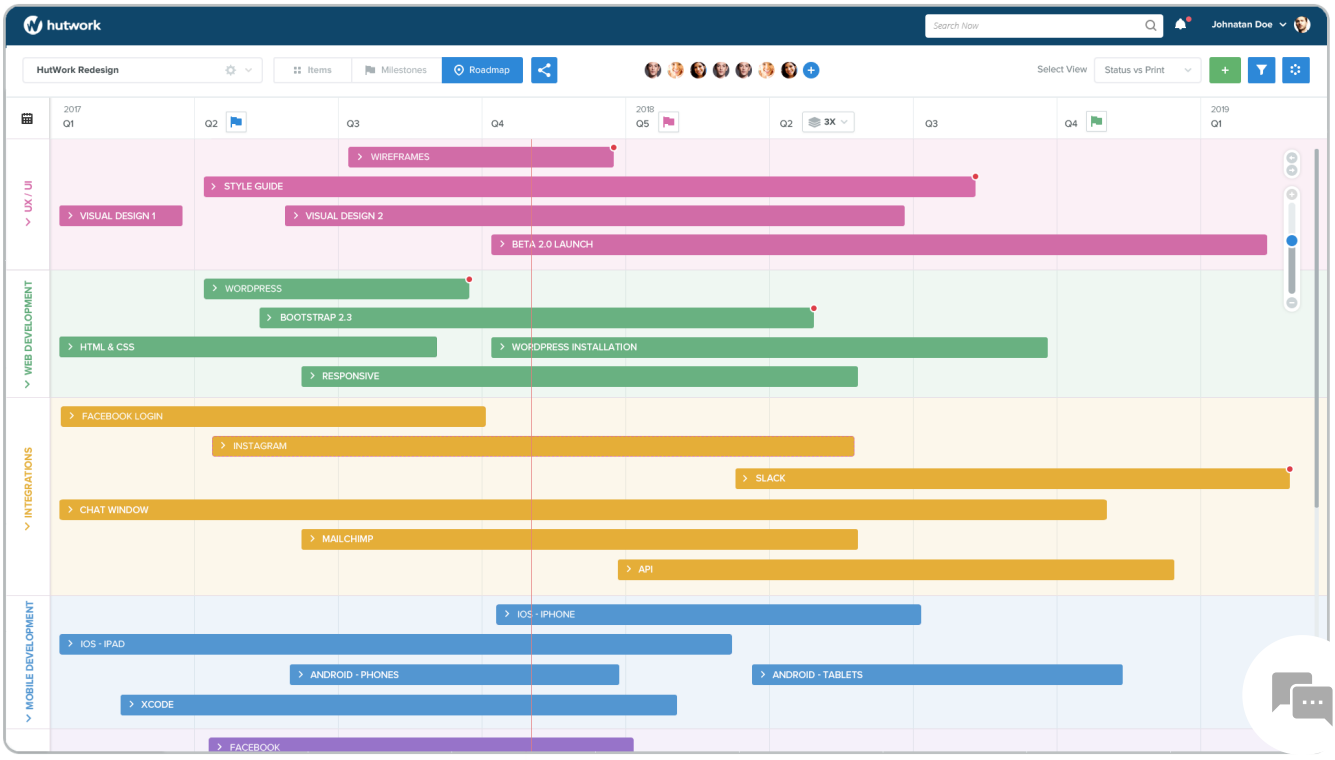
Grouping the information around the themes makes a roadmap highly flexible and updatable, which is a great fit for sprint-based development. The best advice concerning strategic roadmapping is to include only important information. Otherwise, you risk turning your roadmap into a clumsy scheme, difficult to both understand and maintain.



Strategic software product roadmap example

Source: productplan.com

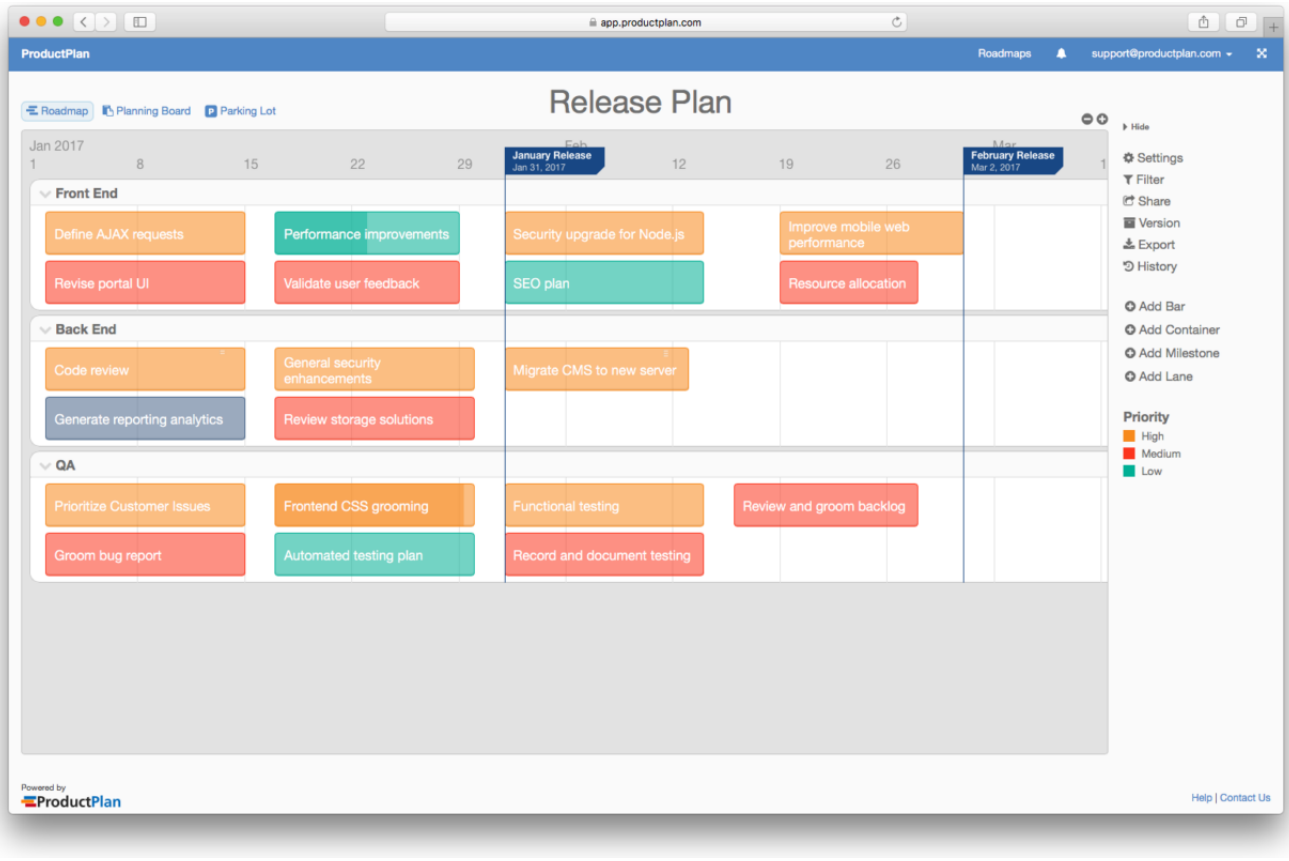
A **technology roadmap** or **IT roadmap** is a low-level document that describes technical requirements and the means of technology implementation. IT roadmaps are quite detailed. They contain the information on each deliverable, explaining the reason for such a decision.



Technology roadmap example

Source: hutwork.com

A **release plan** is used to set strict time limits for releases. A release plan should focus on the actual deadlines without specifying release details.



Release plan example

Source: productplan.com

It is highly recommended to use roadmap specific tools to create your own roadmaps. Online tools like Roadmunk provide various templates for product roadmaps, allow quick editing, and provide easy sharing across all team members.

Keep in mind, that a roadmap, depending on its type, can be a product document that states requirements. It also describes the process and guides your team through development.

Tools for software documentation

General purpose tools

There are countless collaborative tools for software development teams. Those can help to state requirements, share information, and document features and processes:

- Atlassian Confluence is the most popular collaborative project tool that has the whole ecosystem for managing product requirements and writing documentation. Confluence is known for a stable wiki system and an efficient user story management interface.
- Document 360 is a self-service knowledge base/software documentation platform designed for Software-as-a-Service products.
- bit.ai is a tool for collaborative documentation creation, storing, data sharing, and using a wiki system. The documentation is interactive, meaning that developers can embed blocks or snippets of code right into the document and share it in one click. Once you finish editing your documentation, you can save it in PDF or markdown format, and post on any other platform.
- Github needs no introduction, except for those who want to use it for software documentation. It provides you with its own wiki system and allows for converting your documentation into compelling website showcases.

Markdown editors

As software documentation is easier to be used on the web, it has to be created in a proper format. That's why text-based markup languages are used. The most popular one is Markup, which can be easily converted into HTML, doesn't require any special knowledge to use it. Markup is used on GitHub and Reddit, and basically everywhere for web-based documentation. So, here are some Markdown editors that can be useful for creating documents for your project:

- Visual Studio Code
- Typora
- iA writer
- Quiver

Roadmap specific tools

It's a good practice to use roadmap specific tools, as they allow you to share the information quickly, update timelines or themes, add new points, and edit the whole structure. Most roadmapping tools provide templates for different roadmaps to let you start working with this document right away.

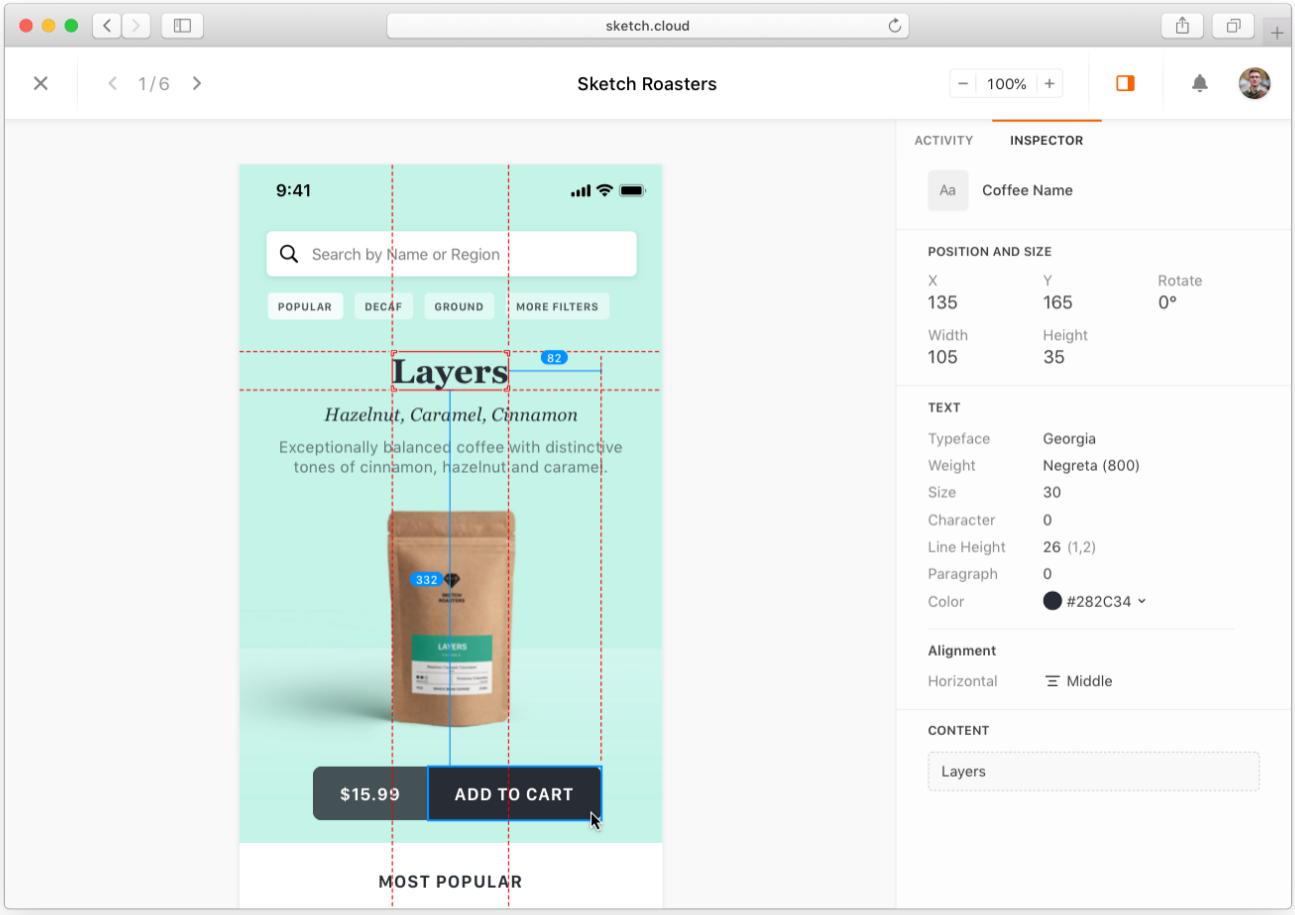
- ProductPlan
- Aha!
- Roadmunk
- Roadmap Planner

Basically, all the tools offer free trials and paid plans with differences in templates, number of roadmaps, and people you can share them with.

Tools for UX documentation

The most popular tools for user experience design are prototyping tools that help create sketches, mock-ups, wireframes, and interactive prototypes:

- Sketch is a simple but powerful vector-based design tool that has a web-application and a Mac desktop client. Sketch is well-known and quite simple, offering enough capabilities for designing interfaces.



Sketch interface

- InVision is one of the most popular tools for prototyping. InVision is famous for its collaborative features and cross-platform capabilities, making it a great tool for designing future interfaces.
- UXPin is a Mac and Windows design tools that allows you to build any type of blueprint. You can also upload your sketches or wireframes from other products and make an interactive prototype of it.
- Adobe XD – where *XD* stands for experience design. The product is aimed at UX specialists. It allows designers to create high-fidelity prototypes and share them via the app.

Tools for API documentation

The process of creating API documentation is most often automated. Programmers or tech writers may write the documentation manually or use API documentation generators:

- Swagger is a free self-documenting software service designed to create and update RESTful web services and APIs.
- RAML 2 HTML is a simple documentation generator that uses RAML specifications.

Tools for technical writers

Professional tech writers often use specialized software for creating high-quality tech documentation. Such tools are called *content management systems*, or CMSs, and allow for easier building, organizing, and managing various documentation. A CMS can operate different file formats, import and store content, and let multiple users contribute to content development. Some popular CMSs include:

- MadCapFlare — a powerful cloud-based software with a multi-channel publishing feature, multilingual support, extensive learning resources, and more.
- Adobe RoboHelp — a full-featured CMS that allows for creating media-rich content, convenient managing of microcontent, collaborating for version control, etc.
- ClickHelp — an award-winning platform offering easy migration from other programs, flexible permission options, and a number of reporting capabilities.

Samples and templates for software documentation

Many of the tools described in the previous section provide a variety of templates for creating tech documentation. However, if your team is still struggling to find a qualitative template for some type of software documentation, here are more specialized sources to check out.

General project documentation templates

The following sources provide a wide variety of templates related to software development and project management:

- Atlassian Confluence Templates offers general-purpose project documentation templates with their product out of the box.
- ReadySET Pro is a large library of software documentation templates in HTML that include planning documents, architecture, design, requirements, testing, and many more.
- ReadTheDocs is an all-in-one template made with ReadTheDocs platform, providing instructions on writing each type of document you may need, from architecture and UML diagrams to user manuals.

Product roadmap templates

Downloadable templates might be harder to manage and collaborate on, but can still get you started quickly. Here are some sources where you can find a number of roadmap templates:

- Office Timeline
- Template.net
- TemplateLab
- FYI

Quality assurance documentation templates

If you are still looking for QA-related templates, you might want to check here:

- StrongQA.com has various documentation templates for QA-specialists. These include testing checklists, smoke testing templates, test plans, and more.
- Template.net has a section with quality assurance plan templates.
- EasyQA offers an SDK for software testing and provides templates with detailed guidance on how to create a qualitative test plan.
- Software testing is a big platform, including a blog, forum, and all sorts of information materials for testing specialists.

Software design document templates

Software design documents are sometimes also called product or technical specifications. It's one of the most important pieces of software documentation. You can adjust one of these templates to fit your needs:

- Sample Templates
- FYI
- CX Works
- cs.iit.edu
- CMS (.docx file download link)
- cs.dal.ca (.doc file download link)

Specialized architecture samples: AWS, Microsoft Azure, and Google Cloud

Today, as more businesses prefer to migrate to the cloud, there are some well-known trusted providers that offer training and architecture samples to facilitate operating in their environments:

- Amazon — the AWS architecture center provides AWS architectural guidance, frameworks, tools, and best practices for running architectural workloads in the cloud.
- Microsoft — this resource suggests a lot of useful materials on the Azure architecture, including example scenarios, architecture diagrams, and more.
- Google — visit the official icon library of samples for building Google cloud architectural diagrams.

How to write software documentation: general advice

There are several common practices that can be applied to all the major types of documentation we discussed above.

Write just enough documentation

You should find a balance between no documentation and excessive documentation. Poor documentation causes many errors and reduces efficiency in every phase of software product development. At the same time, there is no need to provide an abundance of documentation and to repeat information in several papers. Only the most necessary and relevant information should be documented. Finding the right balance also entails analyzing the project's complexity before development starts.

Consider your audience

Try to keep your documentation simple and reader friendly. It has to be logically structured and easily searchable, so include the table of contents. Avoid long blocks of text whenever possible and use visual content as it is easier to absorb information this way for most people.

You also have to remember who the document is written for. If it is for end-users, it definitely has to be written in plain language so that the readers are able to understand it without consulting the tech dictionary. If the documentation is addressed to stakeholders, it's also worth avoiding complex, specialized terminology, tech jargon, or acronyms as your client might not be familiar with them. However, if it is for your team tech specialists, make sure you provide all the accuracy and details they need to stick to the development plan and build the needed design and features.

Use cross-links

Use cross-links between documents, whether those are product pages or user guides. Proper navigation through your documentation is important to give the reader the right understanding of a subject. Such practice can be considered user-flow, but for your project documentation.

Don't ignore glossaries

Documentation can be dedicated to internal or external usage. In the case of external documents, it is better to provide a clear explanation of every term, *and* its specific meaning in the project. Documentation should communicate ideas in clear language to set lingua franca between stakeholders, internal members, and users.

Keep your software documentation up to date

Proper maintenance is very important as documents that are outdated or inconsistent automatically lose their value. If requirements change during software development, you need to ensure that there's a systematic documentation update process that includes information that has changed. And, if any updates take place when the product is already on the market, it's crucial to inform the customers and refresh all the user documentation.

It is a good practice to establish some sort of maintenance and update schedule. You can either make it at regular intervals, i.e., weekly or monthly, or relate it to your development plan and, say, update the documents after every release. Automated emails or release notes can help you follow the changes made by the development team.

You can also use a version control tool to manage this process more efficiently. It will let you track changes made, retain previous versions and drafts, and keep everyone aligned.

Documentation is the collaborative effort of all team members

The agile method is based on a collaborative approach to creating documentation. If you want to achieve efficiency, interview programmers and testers about the functionalities of the software. Then, after you have written some documentation, share it with your team and get feedback. You can also attend the team's meetings to be in the loop or check the Kanban board regularly. To get more information try to comment, ask questions, and encourage others to share their thoughts and ideas. Every team member can make a valuable contribution to the documents you produce.

Hire a tech writer

If you can, it's worth hiring an employee who will take care of your documentation. The person who generally does this job is called a technical writer. A tech writer with an engineering background can gather information from developers without requiring someone to explain in detail what is going on. It's also worth embedding a technical writer as a team member, locating this person in the same office to establish close cooperation. He or she will be able to take part in regular meetings and discussions.

More tips for creating and maintaining your documentation

Here's a few more suggestions that can help you optimize and speed up the process of document writing and further managing:

- think of the most efficient medium for conveying information. For example, making audio or video recordings can be a great option for requirements capture, user guides, etc.;
- insert links to the relevant online articles or information pages instead of reproducing them in your documentation;
- generate diagrams from code or databases when possible rather than creating them from s

- use screenshots and other pictures — that would help you quickly find what needs to be updated so you won't have to read the entire text;
- consider storing your technical documentation together with the source code, just keep them separated. That can help with keeping it updated and will let everyone know where to find it;
- customize access to avoid extra changes. Give editing permissions to potential authors, while those with view-only access can still see the information, but not modify it;
- make sure the authors can have quick and easy access to the documentation for reviewing and updating. Remove such barriers as unnecessary authorizing and/or approval procedures;
- keep previous versions and archive emails on the project as you might need to get back to them in the future;
- remember to back up;
- use tags to make search easier;
- if documentation is a way to share knowledge, think of other ways of communication or find out why team members don't just talk about that. It can be beneficial for overall teamwork and reduce the amount of documentation needed.

Final word

The agile methodology encourages engineering teams to always focus on delivering value to their customers. This key principle must also be considered in the process of producing software documentation. Good software documentation should be provided whether it is a software specifications document for programmers and testers or software manuals for end-users. Comprehensive software documentation is specific, concise, and relevant.

As we have mentioned above, it's not obligatory to produce the entire set of documents described in this article. You should rather focus only on those documents that directly help achieve project objectives.

14 Comments

Further Reading

WHITEPAPER

Agile Project Management: Best Practices and Methodologies

Agile Software Development Metrics and KPIs that Help Optimize Product Delivery

WHITEPAPER

Estimating Software Engineering Effort: Project and Product Development Approach