RedPenguin

Playbook

Values and strategies for successful product teams

TABLE OF CONTENTS

INTRODUCTION

This book is a guide for developing digital products following a proven procedure as well as running an effective and low overhead product development process. It describes what we do at RedPenguin and is based on experience we gained while working on digital products for international clients in various industries.

Most teams working on digital products face the same challenges – be it an inability to ship features effectively and with high quality, providing good insight into timelines and meeting stakeholder's expectations or preventing technical debt to accumulate. We believe that the techniques and practices we present in this book will prove helpful for anyone developing digital products of any kind in any industry – whether you are a CTO of an established company, a founder looking to set up their own product team, a product manager, designer or developer.

The specific tool setup that works great for one team might not work as well for another one. That is why instead of focussing on any precise software, we focus on the underlying values and techniques that are independent of any particular tools or organizational structure. Whether you are running a small team of only a few people or a large organization, whichever of the plethora of applications for managing teams you are using, you should be able to adopt the techniques presented in this book.

Organization of this Book

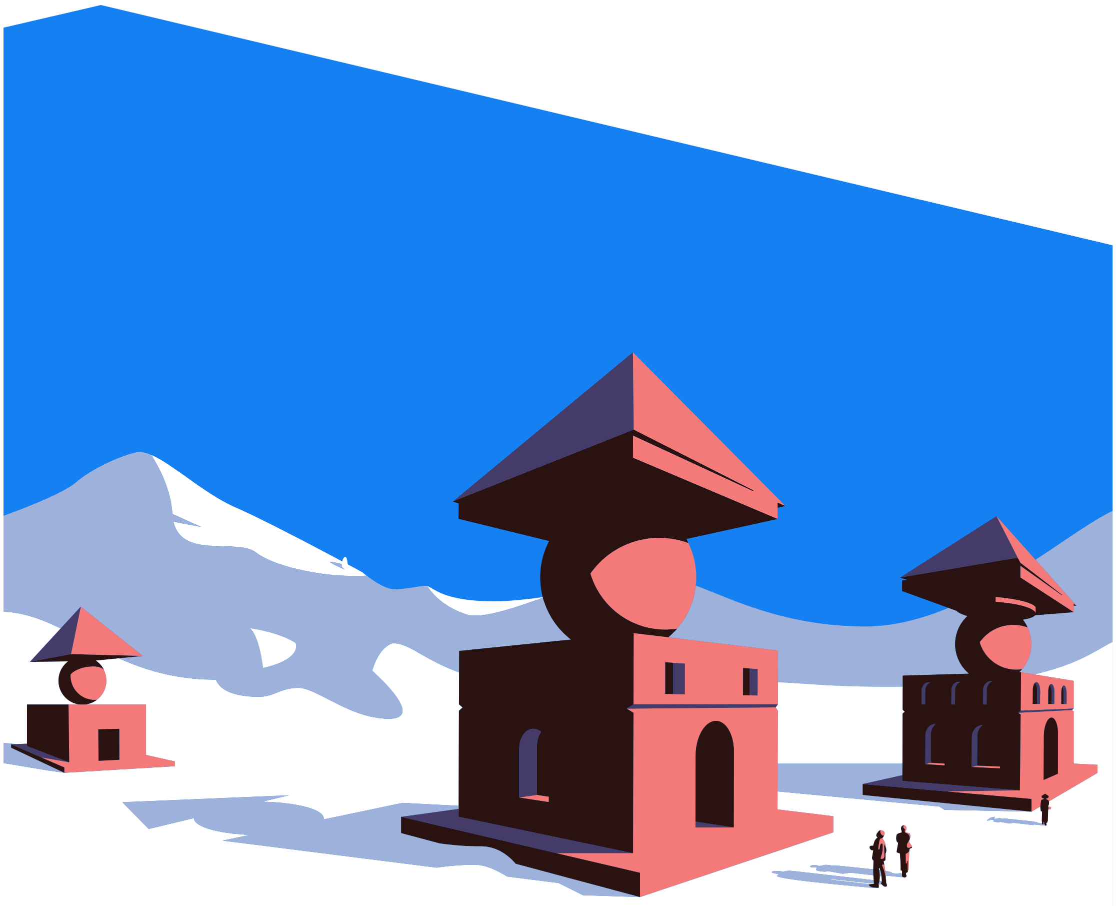
This book is currently organized into two main chapters:

1. **"Digital Product Projects"**

   describes how digital products are built effectively and in a targeted manner that involves all stakeholders

2. **"Development Process"**

   describes how product teams run an efficient process that enables the team rather than stand in its way



Part 1

Digital Product Projects

01

Digital Product Projects

Digital Product development is a complex procedure. Besides purely technical aspects, transforming a potentially vague idea into a product that users can and will use or solving a business problem through software also requires to take several other aspects into account. In order to achieve an ideal result that best pursues a project's goals, various groups with different backgrounds and motivations need to collaborate closely. A clearly prescribed sequence of steps and actions to take in a particular order helps to make this process as effective as possible while involving all stakeholders and their individual expertise.

Digital product projects can be broken down into four main stages:

1. **Product Strategy**

   identifying and understanding the goal of the project and the environment it operates in

2. **Product Design**

   conceptualizing a product that pursues the project's objective

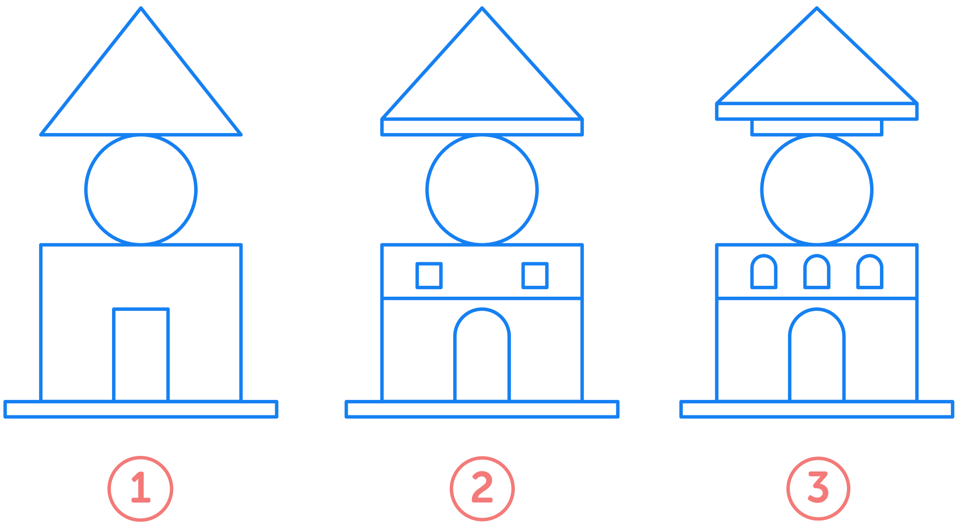
3. **Product Development**

   implementing the product and launching it

4. **Evaluation**

   validating whether the product indeed achieves the objective

Building the entire product as one pass through all of these stages maximizes scope and thus risk. Instead, we recommend splitting up the project into smaller units and passing through the stages several times, once for each unit, incrementally building up the product. We refer to the deliverable of each pass-through of all the stages as an *"increment"* of the product. For the first increment, we recommend building only the minimal set of functionality necessary to deliver value (the *"Minimum viable product"* or MVP). Each subsequent increment adds a coherent set of functionality to the product that extends it in a way that is meaningful to its users.



Breaking down a project into an MVP and subsequent increments helps reduce scope and complexity and develop the product in small and coherent steps

Software projects are highly dynamic. The bigger the scope and the complexity of a project, the likelier it is to steer off track, resulting in delays and budget overruns. By limiting the scope per increment, complexity and thus risk is minimized and also usable versions of the product that provide value to users are available earlier in the process. These early versions of the product can already be shared with users and be used to acquire valuable feedback which might even result in a shift of focus or project direction.

Before work on the first increment of a product can be started, the project must be kicked off and the infrastructure needs to be set up. These steps only happen once and are not part of later iterations through the four main stages.

Kick-Off Meeting

Every project should start with a kick-off meeting assembling all project stakeholders - ideally in person, otherwise in a video call. The main goal of this meeting is for everybody involved to get to know each other and to acquire an understanding of everyone else's roles, responsibilities and goals. The group of project stakeholders should be kept as wide and open as possible and include everyone who has any interest or relation to the project, even people that might not necessarily be involved in the project daily but still have relevant perspectives that the project team should be aware of.

Setup

After kicking off the project, the infrastructure that is going to support the team needs to be set up, as effective tooling is vital for effective collaboration among the stakeholders. Ensure tooling is cloud-based and therefore persistent and traceable. The main tasks are:

- Deciding and setting up a source control system for maintaining the project's source code; this would usually be a git-based system
- Deciding and setting up a system for maintaining and collaborating on design source files; we recommend using software that allows for component-based design, and a system that allows maintaining assets like source code with branching, pull request and review mechanisms
- Deciding and setting up a system for maintaining and collaborating on documents (e.g. feature concepts, etc.); this system should support versioning and commenting for effective collaboration
- Setting up a shared communication channel that fosters traceable group communication; while the particular tool of choice is not relevant, we advice against relying on email and for using a realtime system instead
- Ensuring all project stakeholders have access to all of these tools and will receive notifications when they are assigned tasks or are mentioned by others

Effective tooling and usage of it are critical for all teams, on-site or remote. All relevant information, decisions and decision-making processes must at all times be accessible and transparent to all stakeholders for effective collaboration.

Once the project has been kicked-off and infrastructure is set up, the team can commence passing through the main stages starting with product strategy.

Product Strategy

In order to build an application that effectively achieves a particular objective, one must first understand the subject matter and then clearly define the current state of the situation. Next, the desired objective must be established to develop an effective strategy for all subsequently taken actions. All project stakeholders need to work closely together during this stage to identify answers to these questions:

- What exactly is it that is currently lacking or requires improvement (an example could be customers having to call a company's sales representative to configure a product specifically to their needs) and how does that manifest in the business? (e.g. lost sales)
- Who is affected by the present problem and who is going to use the product that is being built? (e.g. the project teams' own organization or its customers)
- What should be the effect on the business once the solution is in place? (e.g. increased sales, reduced workload for the sales team, etc.)

The goal of the product strategy stage is to find answers to these questions that are based on facts and knowledge rather than assumptions. The techniques for acquiring said facts will vary with each new project and depend on the particular problem, the respective industry and the time or budget constraints. We recommend using one or several of the following techniques:

- Data analysis of existing data e.g. from CRM systems, event logs or analytics data from existing web applications; utilizing these data sources often helps to create a better understanding of the current situation and sometimes reveals that the actual problem is different from what it seems; if none of this data is available, think about whether it is possible to start collecting it in potentially already existing systems
- User research: getting direct feedback from users of an existing product that is to be replaced or potential users of an entirely new product is the best way to understand their backgrounds, motivations and challenges
- Market analysis: besides understanding the future users of a digital product, it is key to understand the market the product will operate in, its rules and limitations, what competitors with similar products are offering and what the relevant trends are

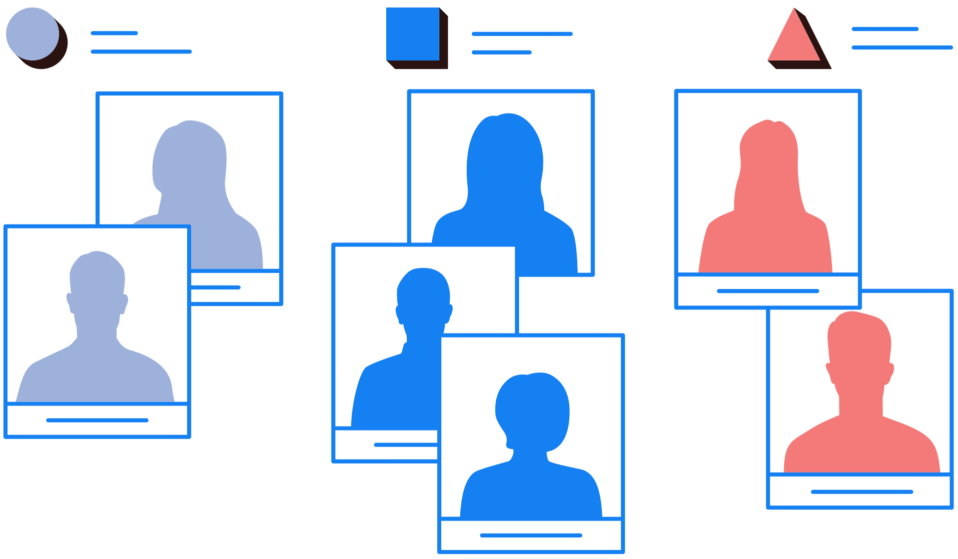
The product strategy stage will typically be conducted as one or several workshops in which the project team has a facilitated discussion based on patterns and insights uncovered through the research. It is critical to get all project stakeholders involved in these workshops, in particular all of the business experts from various backgrounds like marketing, sales, customer service, etc. All of these groups will have unique insights and experiences that must all be heard in order to clearly understand the situation and define the project's objective.

Mission Statement and KPIs

Based on the gathered facts, the project team will be able to define a high-level objective for the project (e.g. a web application that allows customers to configure the company's products according to their needs and order that configuration via the company's website). That information should be written down in a short (aiming for about one page) mission statement document to clearly express the project's scope and goals. We also recommend defining one or more Key Performance Indicators (KPIs) (e.g. *"number of sales via the website"*, *"number of support requests from users"*) to quantify the situation before and after the project for easier evaluation in later stages. If no data is available for these KPIs, it might be necessary to set up additional tracking or conduct research.

Personas

In addition to the mission statement, the project team also needs to identify potential future users, along with their prior knowledge, skills, needs, and motivations concerning the application. We recommend defining one or multiple personas (typical representatives of the future user base) based on all acquired facts in order to gain a good understanding of a product's users, which is a key prerequisite for developing an application that effectively caters to their specific needs. Personas can be a useful tool for aligning within a team and communicating who those people are.



Defining Personas helps develop a better understanding of who a product is built for

In order to identify personas, all possible future users (as derived from the acquired data and/or based on the business experts' experience) of the product are listed. For each of those the following questions are answered:

- Why are they interested in the product and what problem are they trying to solve with it?
- What previous experience do they have in relation to the product?
- What will make the solution appealing and valuable for them?
- What makes them unique compared to the other personas?

These users are then grouped by related concepts and patterns that emerge from the answers to these questions. Each of the resulting clusters constitutes a persona that is given a name so it can be easily referred to and described with its main characteristics in a persona document (aiming for about one page per persona).

Make each persona memorable, associating them with characteristics of a real person such as an image, name, age, attributes, desires, and behaviors. It is vital that personas are not defined once and then forgotten, but rather that they are brought to the table daily. Seeing a feature through the eyes of a persona creates empathy for the customer segment that they represent, ensuring that we meet their needs as much as possible. They become a valuable tool when they are integrated into the everyday decision-making process (e.g. when prioritizing features, planning feature concepts, and recruiting for usability testing).

Product Design

Once the strategy for the product has been established and is well understood and agreed upon by all stakeholders, the next step is identifying the set of functionality to be added in the respective increment of the product. If the product is going to replace an existing system, the functionality provided by that system can be used as a reference. In the first increment, we recommend starting with the minimal possible version of the product, addressing the most basic needs of the project's personas (e.g. building only a bare product configurator but no payment mechanism or order management system yet). Subsequent increments will extend and optimize what was built in previous increments with coherent blocks of functionality (e.g. adding a checkout flow with payment, adding an order management system for the company's sales representatives, etc.).

While applying a systematic approach for developing a coherent understanding of what will be built, we recommend keeping the product design stage limited and not do excessively detailed up-front design and specification. Detailed mockups tend to give a false impression of a finished product even though it is basically impossible to cover every possible aspect and prevent new challenges for particular features from being uncovered in the development stage. Also, static mockups will never give as realistic an impression for the final product as one will get from an actual application even if it starts out with very limited functionality initially and is only built up incrementally over time. Therefore we recommend aiming for a quick transition to the development stage, using the real application to collect feedback and validate assumptions as early as possible.

Feature Concepts Workshop

In the Feature Concepts Workshop, the project team defines concrete concepts for individual, coherent features of the application. The goal is to develop an understanding of the structure of each feature's core functionality, the UI elements the users interact with and the transitions between individual states in the flow.

This workshop requires input from all project stakeholders:

- The business experts have unique expertise and understand best what goal a particular feature enables the user to achieve
- Designers have the expertise to understand which interaction design lets the user achieve the above goal in the most efficient way
- Engineers can assess different alternatives for implementing a feature for their technical feasibility and associated effort

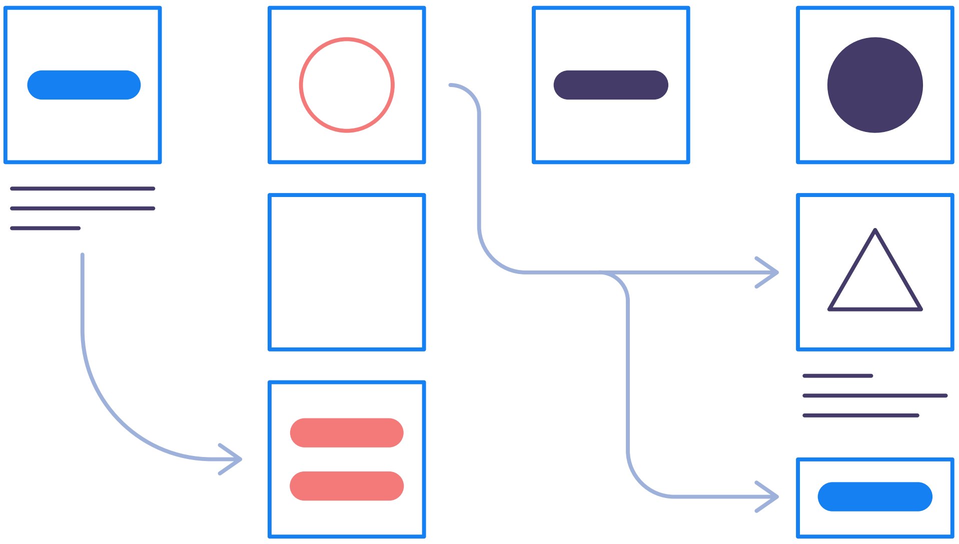
Features concepts capture features on a relatively high level of abstraction. They focus on describing the functionality as such without going into details that would prescribe a particular implementation or any concrete user interface.

For each feature, identify:

- The navigational structure: screens or pages, dialogues, menus, etc.
- Interactive elements: buttons, inputs, etc. that the user needs to trigger actions in the flow or needs to acquire information that is relevant to them
- Flows: transitions between sections of the navigational structure that are triggered by interactive elements

These conceptual descriptions of features are visualized using coarse diagrams (e.g. hand-drawn marker sketches) including all of these identified pieces and the relationships between them. During the workshop participants will discuss and reconstruct these diagrams several times until an ideal solution emerges that all stakeholders agree on. That final concept is then written down in a feature concept document consisting of:

- A textual description of the feature, the need that it addresses and the solution that it provides
- The persona(s) that the feature (mainly) caters to
- The low-fidelity diagram showing steps in the feature, focusing on functional aspects and transitions between steps



Feature Concepts are described focussing on the flow and relevant interactive elements rather than concrete visuals

Once a feature concept has been clearly described, revisit it and look for non-essential aspects that are not strictly necessary for the feature to provide value as well as edge cases that might not immediately be apparent. These will be noted in the feature concept document as well and can be used as escape hatches later on to limit the scope and effort for the feature as they can potentially be left out in a first implementation.

User Interface Design Foundation

The initial stage of user interface (UI) design entails establishing a visual foundation that can be applied consistently to the product. Take some time up-front to create brand guidelines (if none exist) and establish variables such as typography, color, and spacing.

Next, create mid-fidelity wireframes of the first feature or coherent set of features to convert the functionality to concrete UI elements, but without potentially distracting stylistic details of high-fidelity designs. Once the wireframes are agreed upon by all relevant stakeholders, mockups of the feature can be created. Use the visual foundation developed previously as a base to style the mockups, creating UI components with future reusability in mind. As the design grows, reuse components as often as possible to ensure consistency and efficiency in the project. Later design steps will likely be focused on smaller sets of UI and build upon decisions and visual elements that were introduced previously.

Establishing principles and guidelines on how to use components as well as the creation of a UI component library create the basis of a design system.

It's helpful to accept a bit of uncertainty and iterate on the real product rather than focus too much on high-effort prototypes. The goal is to get something real in the hands of our users as fast as possible to validate our ideas. Their insights inform the design, helping evolve it iteratively according to their actual needs.

The first increment of a product will require creating wireframes and mockups for one of the features in order to agree on a layout and visual style for the product

Increment Planning

Once the features are described in the product design stage, define a plan for the current increment of the application. For each feature that is part of an increment, list which of the non-essential aspects and edge cases will be included in the increment (e.g. configuring products of the most successful category), which ones will not (e.g. configuring products with little sales volume) and which ones will be moved to a later increment in case the team is running out of time or budget (e.g. allowing customers to see the status of past orders). Introducing these escape hatches allows the project team to decide between implementing essential versions of more features versus covering more non-essential aspects and edge cases for fewer features later in the process if time should become scarce.

Once all stakeholders agree on the plan for the product increment, the feature concepts together with included as well as excluded aspects and according edge cases are written down in an increment plan document, which is used to estimate a rough timeline. That plan is not irrevocable and neither is the estimate more than an assessment based on the information available at the time. Previously unknown challenges will likely be uncovered during product development and some details and implications will only be understood once work on a particular feature has actually begun. Teams should in fact constantly question assumptions and priorities and change the plan at any time they see necessary. It is also possible at any time to go back and re-plan an increment, adding a different set of features, include or exclude different aspects of them or even start over with the product strategy or the product design if new business objectives are identified. The goal of the increment planning is to get a common understanding of what value will be delivered during the increment and how long that should roughly take (or what the maximum available time and budget is) but not lock the team into a binding specification and schedule that might have a short validity anyway.

Product Development

In the product development stage of a product increment, feature concepts are turned into usable functionality in the application, iteratively building up the product over time. During this process, each feature concept is broken down into fine-grained and more detailed tasks which are then proceeded to be implemented by the project team (see development process). Breaking down feature concepts is done in close collaboration between engineers and designers with the business experts. They discuss individual facets of features as well as different alternatives for designing and implementing those along with the associated effort.

The project team will build slices of the system at once, including everything from the design, the backend, and the frontend code of a particular feature so those slices can be released as functional units that can be interacted with by all project stakeholders. Once a slice is finished, it will be released to a staging system that is available for all project stakeholders. All stakeholders should be encouraged to make active use of the staging system to follow the project's progress and validate that the product is being built according to their expectations. Feedback given by stakeholders will be collected by the project team and addressed in tasks in a subsequent iteration of the product development stage or moved into a later product increment depending on the respective finding's priority.

Once a product increment is completed, the respective state of the product will be released - ideally into a production environment - so real users can access the system and deliver first-hand feedback.

Growing a Foundation

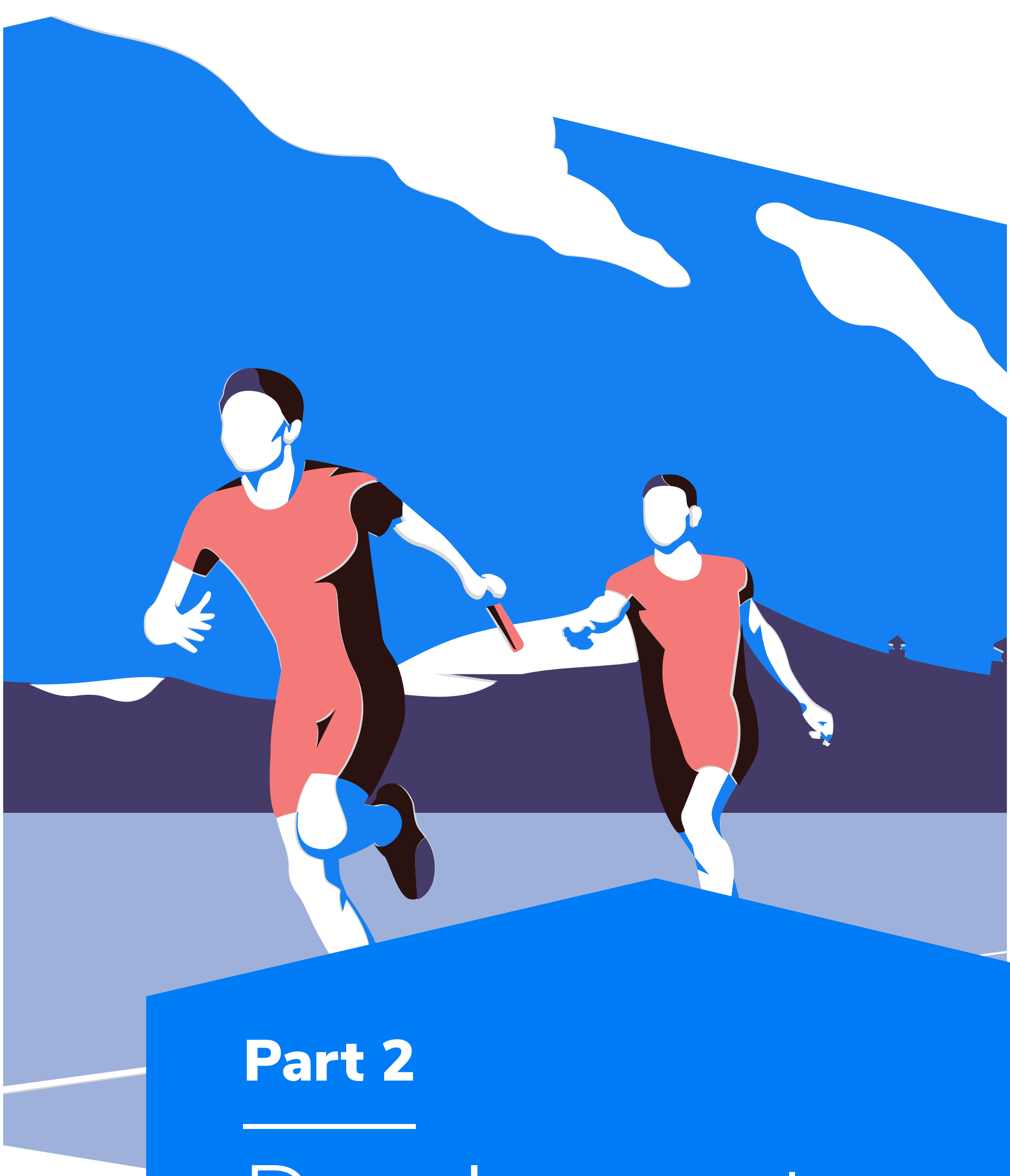
As more features are being completed and the product evolves, foundational elements that will later be reused in other, similar contexts will be identified - both in the application's codebase as well as in the visual design system. That way, each increment of the application results in its foundation becoming broader, stronger and more refined, resulting in increased effectiveness over time, decreasing effort per feature and increasing planning reliability.

Evaluation

Every increment of the application that is completed and released should be evaluated on whether the assumptions made in the product strategy and product design stages proved themselves to be true and whether the project's objective has been achieved as intended. Ideally, that evaluation can be conducted based on the interaction of real users with the product, either based on data captured by an analytics system or ideally by receiving feedback from users directly. Metrics should be collected for the KPIs defined in the product strategy stage. By analyzing those metrics, the actual feedback can now be compared with the initial expectations.

Iteration

Once an increment of the product is completed and released, the project team will go on to build the next increment as long as there is functionality to add and aspects of the product to improve. Depending on the learnings from the previous increment, it might be necessary to go through the product strategy stage again, reconsider some of the assumptions and adapt the project's objective. If this is found not to be necessary, the product design stage can be entered directly to conceptualize, scope and implement the next set of features, followed by executing that new plan, evaluating what was built, etc.



Part 2

Development Process

02

Development Process

An effective development process supports the team rather than providing another obstacle to work around. It does not introduce a set of formalities only for the sake of it and favors communication between all contributors over top-down management. A good process also ensures the right tasks are being worked on at the right time (and in an appropriate manner) and provides a reasonable level of short term predictability. At the same time, it remains flexible enough to adapt to unexpected events.

The RedPenguin development process distinguishes itself through certain characteristics and underlying values:

- It ensures all project stakeholders are being heard and the project's priorities are not being dictated by a single party.
- It is designed to deliver high-quality results in a structured, comprehensible manner.
- It fosters open and replicable communication and sharing of know-how among team members which is particularly important for remote teams.
- It puts a focus on planning and preparing all work carefully before getting started with the actual implementation in order to ensure execution as smooth as possible.
- It does not rely on any particular tools and works for projects and teams regardless of the available infrastructure.

Iterations

Our development process is based on *"iterations"* in which a clearly defined set of tasks is being worked on and ideally completed within the same iteration. Iterations share similarity to what is often referred to as *"sprints"* but we made a conscious decision to refrain from that terminology due to its negative connotations - after all, the goal is not to rush work out but instead to build a project incrementally and iteratively at a sustainable pace in order to ensure a high level of quality while avoiding the accumulation of technical debt.

The concrete tasks for an iteration are identified, defined and prepared collaboratively with all project stakeholders before the iteration starts. We do not recommend maintaining a backlog filled with all tasks that eventually will need to be completed for a particular project. Many of these tasks will become relevant only further out as the project progresses and not be actionable at the time. Thus, these tasks are likely to change anyway and preparing them early leaves teams with a large number of tasks that end up never being tackled or only in a substantially different form so that they become outdated or are even closed untouched. Instead, only the tasks that are relevant for the upcoming iteration should be defined, prepared and then planned as those are well understood and known to be needed at the time.

The main purpose of iterations is to set expectations on all sides and provide short-term predictability and foresight for the stakeholders requiring it. Iterations should be short enough to remain predictable and long enough to supply enough time to finish meaningful work – a length of 2 weeks typically achieves that best. The concrete iteration length for a project must be defined when kicking off the project but can still be adapted afterward. Iterations are a team effort and should therefore be planned and executed collaboratively with team members supporting each other to complete the work as planned.

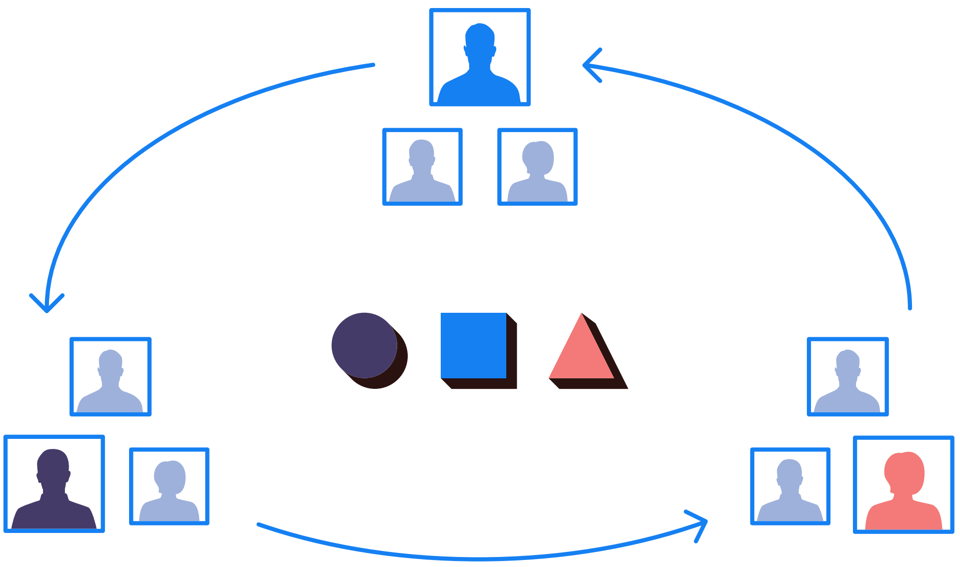
Roles

Our development process functions with flat project teams without dedicated project managers. We believe the traditional project manager role with responsibilities focussed around time and resource management is an organizational anti-pattern and often does not provide the intended benefits but in fact has a negative impact:

- The project manager role can constitute an intermediary between project stakeholders, intercepting direct communication and discussion between them. That results in communication between stakeholders being less direct and effective and can potentially even lead to details and nuance being lost in the process. Sometimes blocking direct communication channels is even the intention behind bringing in project managers in an effort to *"shield"* team members from the direct influence of others. In these cases though, the project manager role only covers a more fundamental problem that is really a dysfunctional project team.
- Intercepting or blocking direct communication between stakeholders via project managers can also have another, potentially more substantial negative impact. Since a project can only succeed when the different involved stakeholders work together and each one's motivations and goals are respected and addressed, it is vital for each stakeholder to understand and appreciate each other's viewpoints. Many projects fail due to the disproportionate weight being assigned to one of the stakeholder's interests and them running the entire project.
- If project managers indeed assume a management role and have decisive authority regarding priorities and deadlines etc., teams end up in a situation where a party that is not actively contributing to the project is in a position to make decisions and thus drive the project. That takes the decision making away from the parties that have first-hand experience and insights and necessarily leads to decisions that are not as well-informed as they could be and often, to time and budget overruns and frustration among the team.

That said, we do support a project manager role that is more oriented towards acting as a communication coach that supports the team by moderating meetings, introducing workshop techniques, etc. We don't think *"project manager"* is a fitting term for such a role though.

Instead of bringing in project managers, we recommend that for any given iteration one of the team members will take on the role of *"Iteration Lead"* . That person is responsible for planning the iteration and ensuring smooth execution. It is conceptualized as a rotating role so that every team member will assume it every once in a while (unless they opt out of the rotation).



The iteration lead role rotates to a different project team member with every iteration

Making the iteration lead a rotating role ensures that all team members realize the perspectives of all project stakeholders instead of getting stuck on their own which leads to mutual trust and alignment of all stakeholders. It also gives everyone on the project team the notion of ownership, responsibility and empowerment contrary to the feeling of being mere executors of someone else's commands. This leads to an overall amplified mindset for each individual on the team.

The main responsibility of the iteration lead is to consult with the business experts (and all other relevant project stakeholders like marketing, designers or engineers) and prepare the iteration. Once the iteration starts, any requested changes to it go through the iteration lead for assessment (who might consult with other project stakeholders for prioritization).

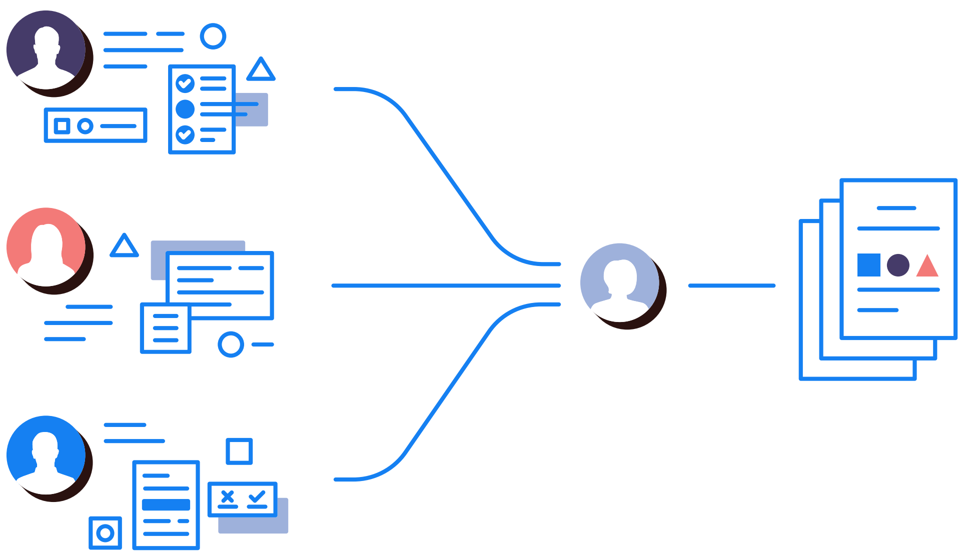
Iteration Preparation

The purpose of the iteration preparation phase is to define the tasks that are most relevant to be worked on during the next iteration. The tasks assigned to an iteration should reflect **all** the upcoming work, not only feature work and bug fixes - in particular, they should reflect design, UX work, and purely technical tasks like refactorings as well. The prepared tasks will then be presented to the team as part of the planning meeting that kicks the iteration off. In order to prepare these tasks, the iteration lead synchronizes with the business experts and other project stakeholders to:

- Identify the most relevant tasks from each project stakeholder's perspectives; the goal here is to find a good balance between work on features and other aspects like bug fixes, refactoring, dependency updates, addressing tech debt in general but also addressing other requirements for instance coming in from the marketing department or other stakeholders
- Help the respective stakeholders translate features or other change requests into actionable tasks; this might include discussing different options for implementing a change along with their potential implications and related effort; the iteration lead will often not possess the required knowledge to do that personally but will then involve the respective experts in these conversations
- Uncover implications and hidden complexities in any of the tasks; while it is not possible to think every task through completely from start to finish and eliminate all inherent risks, we recommend trying to uncover as much of it as possible in the preparation phase to reduce the likelihood of the team running into unforeseen problems later on, potentially leading to delays and deadlocks then; the iteration lead might delegate this work to the respective experts for a particular topic
- Prepare well-written issues for each of the identified tasks or spikes for tasks that require more research in order to be ready to be addressed
- Make sure all of the preconditions are met in order to be able to work on each issue, e.g. all necessary assets have been delivered, translations are ready or legal implications have been checked, etc.
- Prioritize the issues so it is clear which ones need to be worked on first; in reality priorities will often overlap and the expected effort, potential deadlines for individual tasks, etc. also need to be taken into account when defining the order in which tasks should be worked on

> *We use the term "issue" to refer to descriptions of tasks as they are kept in a project's work management system of choice (e.g. Jira, GitHub, etc.). Other common terms are "stories", "tickets" etc. Some tools also allow for structuring issues hierarchically (e.g. with "epics"). We don't think the details of that or the particular term used to refer to these items are relevant for a successful process though.*

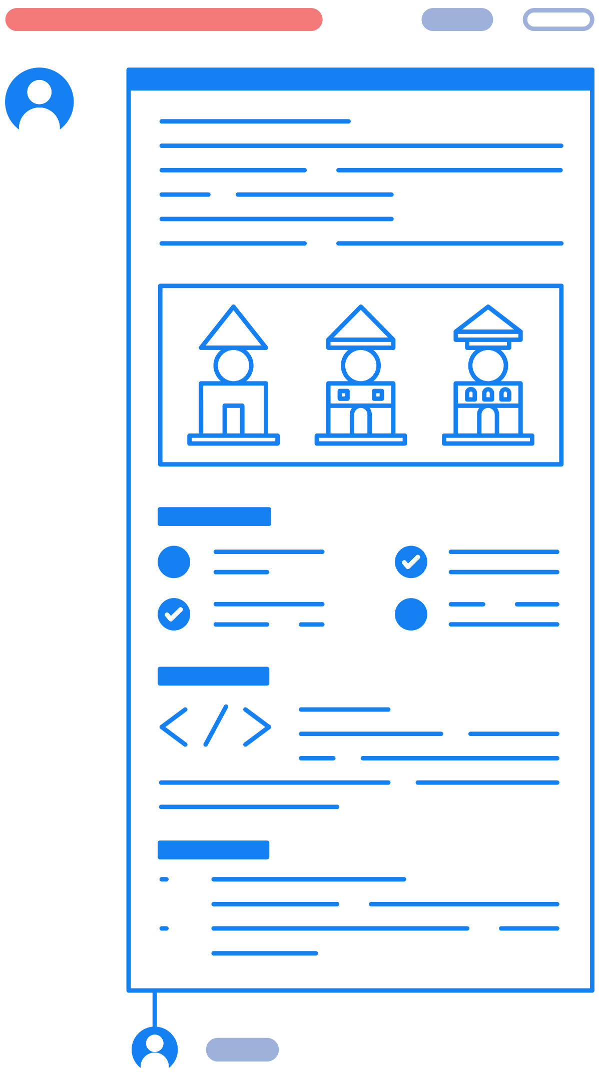
The preparation phase of an iteration always overlaps with the execution of the previous iteration - while one iteration is being executed, the next one is already being prepared since it will start once the current one ends. Iteration leads should typically plan a full iteration period for preparation so that when one iteration starts, the iteration lead of the following iteration starts their preparation. Preparing an iteration will of course typically not require the iteration lead's entire time though but can usually be done at the same time as also contributing to the current iteration.



The iteration lead is responsible for collecting input from all stakeholders, assessing it for consistency and availability of prerequisites and converting it to proper issues

Issues

Well-prepared issues are a key element of an effective development process. They provide guidance for the project team's work, allow external parties non-involved with the project to get an understanding of what is happening directly, and can serve as a future reference to understand what was done in a project, and for which reasons.



Good issues with complete and detailed information are key to a successful project

There is a plethora of tools available for maintaining and collaborating on issues and this process does not prescribe the usage of any particular one – all of the rules we present here are independent of the concrete tooling used in a project.

Good issues aim to:

- Describe what is to be done and why, potentially accompanied by screenshots, mockups/sketches or other visuals that help understand the desired outcome; it might also be beneficial to add a summary of the issue's history, covering previous related changes or alternative approaches that have been ruled out and also providing the reasons for those changes
- Include reproduction steps if the issue describes a bug; ideally, those are visualized with a screen recording or other media
- Detail concrete requirements that must be met and an overview of the changes to be made to complete the issue; in order to prepare this list, the iteration lead might need to partner with a team member more familiar with a particular part of the codebase or feature
- Include all necessary materials that are needed for the issue; this could be visual assets, links to online documentation for third party libraries or APIs or contact details for external parties involved in an issue etc.
- Bring up any open questions that need to be answered or risks that have been identified and might prevent the issue from being completed
- Be a discrete unit of work; issues should only contain related requirements and ideally not represent more than a few days of work - larger issues can often be broken down into multiple smaller ones, possibly even allowing for work to happen simultaneously

Spikes

If a particular task is associated with too many open questions or uncertainties to be converted into a well-prepared issue, it is preferable to plan a spike first in order to resolve these open questions. Spikes should have:

- A description of the original requirement that will eventually be addressed in an issue, potentially accompanied by screenshots, mockups/sketches or other visuals that help understand the desired outcome
- A clear description of what the open questions are and how they are blocking an issue from being created by adding too much risk or uncertainty
- Guidance on potential solutions that should be evaluated or references to promising approaches
- A well-defined timebox, e.g. *"spend max 2 days"*

Iteration Planning

The result of the iteration preparation phase is a prioritized list of well-prepared issues and spikes. This list of issues will then be presented to the team during the planning meeting.

The iteration planning meeting is a joint meeting with the entire project team, the business experts and all other stakeholders involved in the project. During the meeting, the iteration lead presents each issue to the project team so that everyone acquires a coherent understanding of what each issue is about and gets a chance to ask questions.

One goal of the iteration preparation phase is to eradicate uncertainties and open questions around all issues. Instead, each issue should have been carefully examined for open questions and risks and a high-level strategy for completing it should have been defined. That way, the iteration planning meeting can be spent most efficiently and does not end up being a loose team discussion in which tasks are examined collaboratively which is a scenario many product teams are struggling with.

At the end of the planning meeting, the team collaboratively decides whether it can reasonably work on and complete all of the issues that have been presented in the meeting, plus past issues that are potentially moved over from the previous iteration after having been reviewed. If the team considers the presented issues to be too much work for the iteration, they collaboratively decide which ones are moved to a later iteration to be considered again in the future. If any of the issues are found not to be ready to be worked on (e.g. because dependencies of the issue are not ready), the issue is moved to a later iteration as well.

The iteration, once planned, is not a binding agreement. It is still possible for all project stakeholders to react to changes regarding features or priorities and the project team cannot guarantee all planned issues to be completed by the end of the iteration as new challenges might come up once work on an issue has started. The iteration plan is merely a snapshot of the feature requests and priorities at the time it is made as well as a best-effort estimate by the project team of which issues it thinks it can complete within the iteration. Ideally though, an iteration remains unchanged once it has been planned to enable smooth execution which also leads to increasingly predictable estimates as a project progresses.

Iteration Execution

After the iteration has been planned, execution starts and the planned issues are worked on based on their priority.

Once an issue has started to be worked on, the respective team member(s) will self-assign it (not all issue trackers allow assigning issues to more than one person at a time so if multiple engineers collaborate on an issue, they might have to choose one to assign it to). Issues are only assigned once work on them actually starts – pre-assigning issues during planning or afterward block these issues for everyone else to work on if the originally assigned team members are busy with other tasks and do not actually work on them. If an issue is being worked on by multiple team members sequentially (e.g. first the designer for preparing visual elements, then the engineer for implementing those), the latter team member will self-assign the issue once the former is done with their work. Once an issue has been resolved via a pull request or if it is blocked, the engineer(s) will self-assign another issue from the iteration backlog.

Although issues should be well-understood and well-prepared before they are even planned for a particular iteration, for more complex issues it is often beneficial to break them down into smaller, more concrete steps (which is often a great thing to do in a pairing session) before starting implementation.

If there are any alterations requested to the iteration after the planning meeting (e.g. due to unforeseeable changes to features or severe bugs popping up in production), all of these potential modifications to the iteration are triaged by the iteration lead. They might consult with the business experts or other project stakeholders to determine the validity and priority of the incoming requests. If an issue is considered necessary to be added to the iteration after the planning meeting, it can be added but another issue might have to be removed from the iteration in its stead.

If an issue is blocked and cannot progress, the iteration lead is responsible for trying to solve the impediment, potentially synchronizing with the business experts or other project stakeholders that can help resolve the situation.

Likewise, if all issues in an iteration are completed early and there is no more work left to do, the iteration lead will synchronize with the project stakeholders and the iteration lead of the following iteration to discuss which issues should be added. Oftentimes that will mean moving issues from the following iteration into the current one.

All discussions around an issue should happen on the particular issue's respective page in the project management tool of choice. It is of course at times convenient to have discussions in person or through online chat but even in those cases, a brief summary of the discussed points and the outcome should be posted on the issue. This is a necessity for distributed teams and allows everyone access to all of the context of a particular issue at any time. Even teams that are not distributed benefit from this practice as all information that is relevant to a particular issue is and remains available for everyone interested.

Communication

Communication is key for successful project teams - be they distributed or not. For communication to be beneficial for both the team culture as well as productivity, rather than becoming a liability or cause of constant stress, all team members need to keep some basic rules in mind:

- Be responsive: don't leave anyone hanging with unanswered questions or requests. It goes a long way in keeping working relations positive, and communication effective. Respond to online chat messages or mentions on issue pages etc. within a reasonable time, ensure you have notifications set up properly so you will see when somebody mentions you in a discussion or asks for your feedback
- Take your focus time: while some people can respond to any notification that reaches them immediately and still stay focused on the task they're working on, this is not everyone's most effective way of working. Feel free to take your focus time and switch off or ignore all notifications in order to focus on a particular task. Just make sure to check whether anything urgent came up a few times a day. On the flip side, when reaching out to a team member, be asynchronous as much as possible. Give people time to finish what they're focused on, and to respond properly. Very rarely is anything so urgent to warrant full interruption.
- Take advantage of rich media: screenshots, screen recordings, screen shares or even hand-drawn sketches can contribute to a better understanding of what you're trying to show or describe. A screen recording of a delivered feature is always a hit. During calls, switch on your camera so people can see you - talking face to face rather than with audio-only makes a big difference in communication style.

Engineering

Although Engineering and Design work together closely at all times, some of the techniques and practices differ simply because both professions deal with different subjects. We recommend following several established practices that have been adopted from the open source community for a smooth and effective engineering workflow.

Feature Branches and Pull Requests

All changes to a project's codebase are done in branches. No changes should ever be committed to the `master` branch (or whatever the project's main branch is) directly. There should generally be at least one branch per issue - for larger issues it often makes sense to split separate steps into separate branches and merge them one after another.

All changes in a branch should also be related to the same *"topic"* - e.g one branch should not address more than one issue or change entirely unrelated aspects of the application.

If the testing setup, hosting environment and potentially other requirements present for the delivered product should allow it, we recommend setting up continuous deployment so changes get deployed to production as the respective pull request gets merged. If that is not possible, we recommend setting up continuous deployment for a staging system at least so all project stakeholders can follow the project's progress.

Commits

Just as all changes in a branch should be related to the same *"topic"*, all changes within a single commit should be related to the same step for implementing that topic. Each commit should only do one *"thing"*, ideally not touching on too many different parts of the codebase. All commits should also have clear and concise commit messages that make clear what the particular commit does.

Pull Requests

Branches are not merged back to the `master` branch directly but via pull requests (or whatever similar mechanism the tools used in a particular project provide). Similarly to issues, pull requests should have all the information necessary for everyone to understand what they do, how they do it and why. In particular, good pull requests should have:

- A high-level summary of the changes that the pull request contains that provides the reader with a good overview without having to look at the actual code changes
- Guidance for testing the added or changed functionality; this is helpful for the reviewer, product or business experts looking at the pull request on a preview system and a Quality Assurance (QA) team if one exists
- Before and after screenshots or even screen recordings in case of a visual change
- A reference to the issue the changes in the pull request are referring to; if the pull request effectively closes an issue, most tools will automatically do that when the pull request is merged if it contains a comment like *"closes #"* in the description

As with issues, all discussions around a particular pull request should happen on the pull request's page. If discussions happen in person or via online chat, a summary should be posted to the pull request so all information and context is accessible to everyone interested at any time in one place.

It is perfectly fine to create pull requests early on while implementation is still ongoing and they are not yet ready to be reviewed or merged. Doing so is a good way to get early feedback and share the status of something with the rest of the team. Such pull requests should be marked as *"Work in progress"* though - some tools have dedicated mechanisms for doing so, in others that do not offer that, a good technique is prefixing the pull request's title with something like `[WIP]`. Some tools will even block *"Work in progress"* pull requests from being merged.

Preview Systems

In addition to setting up continuous deployment for deploying all changes that get merged into a project's main branch to production automatically, we recommend creating a mechanism that allows booting per-branch/pull request staging systems on demand which we call preview systems. Preview systems are production-like environments that run the entire application with a particular revision of the application's source code (that of a pull request's underlying branch) with real production or production-like data. These systems would ideally be automatically created for every new pull request (and destroyed once the pull request was merged). A link to the respective system would be added to the pull request automatically.

Preview systems are particularly helpful for letting non-technical stakeholders that cannot run the entire application themselves inspect features or changes. That way they can validate the respective features or changes and give feedback that engineers can then address before releasing to production. Preview systems also allow sharing status with external stakeholders that might not even have access to the application's sources at all.

Setting up a preview system mechanism can sometimes be challenging and might require a substantial amount of work. However, when taken into account early on in a project and in particular if the project's infrastructure is containerized anyway, it is often possible to set up preview systems with relatively little effort. Once the mechanism is set up, the benefits easily justify even a substantial effort.

In case of projects that have been running for some time already, have lots of dependencies, are not containerized and would thus be very hard to implement a preview system mechanism for, we recommend at least setting up a shared sandbox environment. That is not as valuable as a proper preview system mechanism as it will be shared among all stakeholders and will hold changes from multiple pull requests as well as be used by multiple stakeholders at the same time that might all be influencing each other. However, it is a good first step and often much easier to set up than automated preview systems.

Reviews

Pull requests are reviewed before they get merged back into the project's main branch; pull requests that have not been reviewed should usually not get merged. For a pull request to be ready for review though, it has to meet some pre-requisites:

- The branch has no conflicts with the target branch
- The changes in the branch are covered by appropriate tests and the Continuous Integration (CI) build is passing
- The pull request is not marked as *"work in progress"*
- The commit history of the pull request has been cleaned up, e.g. WIP commits have been squashed, debug commits have been removed, etc.

When a pull request is ready for review, its author should actively ask for another team member to review it - ideally via the tools used in the particular project if those support it or over online chat, etc. If not we generally recommend applying rotating reviewer assignments so that not all of the reviews depend on one or only a few people. This way, more team members participate in the review process and thus understand different parts of the codebase better. Reviews are a great tool for distributing knowledge about the codebase among the project team which prevents essential knowledge from being isolated to individual developers. Everyone asked for review should reply in a timely manner - even if it's to ask for someone else to be chosen if they do not have the time to do a thorough review.

Once the reviewer approved the changes and CI passes, the pull request can be merged by any team member including the pull request's author. If the original reviewer would like a second review by another team member, potentially someone more familiar with the aspects of the application that are being changed by the particular pull request, they will ask for it. In case anything comes up in the review that cannot be resolved between the reviewer and the author of the pull request, a third person should be brought in to resolve the deadlock.

Reviewing and potentially criticizing other people's work is a sensitive issue which is why we recommend a set of rules to follow:

- Be polite: you are reviewing another person's work that they put time and energy in - don't be dismissive and keep a friendly tone
- Be clear: don't be ambiguous but clearly address issues or modifications that need to be revisited and changed again
- Be positive: while the review's main purpose is to identify mistakes or bad patterns that are not caught by CI, reviews are also a great opportunity to give feedback on particularly good changes.
- Be helpful: if a pull request contains a particular change you don't think should be merged, give the author some guidance by introducing an alternative to the change that does not come with the same drawbacks

Testing

Testing is an integral part of all engineering work and a necessity for delivering high-quality results that also do not deteriorate over time. Untested changes should generally not be merged and not even be reviewed as one cannot know whether the code under review actually works for all relevant scenarios. In fact, if there are no early tests, the code will likely have to go through additional changes later on as soon as tests are being added and bugs are being discovered in the process.

While different languages and frameworks provide different testing mechanisms, a good approach generally is:

- Leveraging small and isolated tests (e.g. unit tests) for the majority of the test cases including edge cases
- Leveraging higher level tests (e.g. integration or acceptance tests) for ensuring the main features and flows of an application work as expected

Continuous integration must mandatorily be set up for a project to be successful. While it should always be possible to run tests locally, they also need to be run automatically for every pull request and after each merge to the project's main branch.

Refactoring

Refactoring is an essential part of any software project. As requirements change and frameworks and languages progress, code written in the past will eventually not be ideal anymore in the present and future. Constant refactoring ensures the codebase does not become stale and improves productivity overall by keeping technical debt at a minimum and avoiding big, painful and risky rewrites that otherwise often become necessary down the line.

When working on the codebase, all engineers should keep an eye open for parts that need to be refactored and either do so immediately in case of simple changes, or bring them up as individual issues for one of the next iterations.

Pairing

Pairing is a great way of spreading knowledge throughout the team, on-boarding new team members or resolving blockers. We encourage project teams to make pairing an integral part of their daily workflow, even across focus areas with e.g. engineers and designers pairing when working on UI changes.

Design

The design workflow greatly benefits from adopting established practices from software engineering. A structured and organized approach fosters communication and quality and is a necessary foundation for close collaboration between designers and engineers.

Design Source Files

Design source files should generally be managed like code, maintaining an authoritative mainline and making changes in change sets that are only applied back to the mainline once complete and reviewed. Source files can be maintained using a version control system just like for the project's source code or in a separate system, depending on the available tools and infrastructure.

Change Sets/Branches

Just like changes to a project's source code, no changes to design source files should ever be applied directly to the mainline. Only after a set of changes has been reviewed and deemed satisfactory, is it applied back as one discrete change. All individual changes that are applied back together should be related to the same UI element(s) or aspect of the UI (e.g. changing the color scheme once for all UI elements). Once changes have been applied back to the mainline, that fact should be noted in the issue describing the work that was done in the change. The change should also be referenced in the issue if the tools allow that, otherwise before/after images of the change should be attached to the issue.

Reviews

When a design change is ready for review, its author should actively ask another team member to review it - ideally via the tools used in the particular project if those support it or over online chat etc. if not. Everyone asked for review should reply in a timely manner - even if it's to ask for someone else to be chosen if they do not have the time to do a proper review.

Once the reviewer has approved the changes, they can be applied by any team member including the author of the change. If the original reviewer would like a second review by another team member, potentially one more familiar with the aspects of the design that are changed, they should ask for it. In case anything comes up in the review that cannot be resolved between the reviewer and the author of the pull request, a third person should be brought in to resolve the deadlock.

Changes to design sources need to be reviewed for:

- Completeness: does the change contain and/or cover all of the necessary and/or affected elements and states?
- Consistency: does the change fit in with and leverage the project's design system and overall visual direction?
- Feasibility: is the change possible to implement with the technology of choice with a reasonable effort?

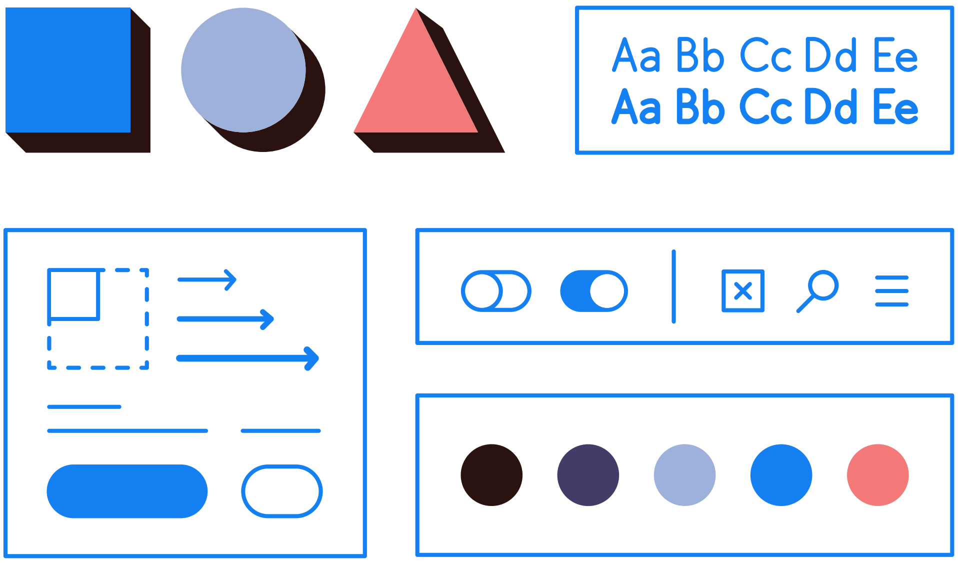
When reviewing design changes we recommend following the same set of rules as when reviewing code changes.

Design Systems

All design work should eventually result in a design system evolving for the respective project, independently of its nature or scope. The design system is a structured, multi-level UI framework where each layer builds on top of the elements defined in the previous one, going from simple to complex, e.g.:

- Basic rules around font choices and color schemes build the foundation for the design system
- Atomic elements like buttons, labels and inputs are the building blocks for all higher-level elements
- Components are built on top of atomic elements and/or other components and resemble groups of lower-level elements that function as a unit like search forms, headers, etc.
- Components are arranged in particular ways to compose actual pages or screens for an application

A design system will ensure consistency across all of the UI of an application as well as minimize the work necessary for extending the UI over time as new elements will build on top of existing ones.



A design system is the visual foundation for a digital product

With every change that is applied, the design system is incrementally built up and extended over time. New UI elements that were not previously covered will reuse lower-level elements where applicable and only introduce new concepts and patterns where no such reusable elements are available. In cases where existing UI elements turn out to not be suited for previously uncovered cases, these elements might need to be changed, potentially making changes in other, higher-level elements necessary as well. These cascading changes ensure consistency across all of an application's UI.

Productivity Benefits

Design systems do not only support a consistent user interface but also directly improve productivity. Since the application's UI is based on a common set of rules, elements and components, engineers can build new pages without needing additional guidance from designers, simply by following the rules of and using the elements and components of the design system. In most cases, elements and components defined in the design system will even have direct counterparts in the application's source code. That way, designers do not have to maintain designs for every individual page or screen of an application. They only need to ensure the design system provides the rules and elements to compose all of the application's UI.

Deliverables

Ideally, no dedicated design deliverables should be necessary and engineers should work with the design source files directly and extract the information and assets they need from those. Therefore, the design sources files should be managed in a way that makes extraction of assets and necessary artwork easy for the engineers and designers who will use them. This should include preparing and marking elements for export, including proper offsets and intended appropriate file formats to reduce the necessity for producing any special deliverables beyond the files themselves.

If any particular deliverables are required that the engineers can not extract from the design source files directly, those deliverables should be attached to the respective issue in the format best suited for the particular use case.

In order to demonstrate complex behavior or animation, narrow prototypes or videos should be favored over text descriptions when possible to eliminate ambiguities and misunderstandings.

CLOSING NOTES

We hope that this book served as an inspiration for addressing some of the problems you and your team might be facing and rethinking the techniques you are practicing or might be missing.

An evolving Document

This book is the result of experience we made over the course of many years working on many different projects with clients around the world. We did not know everything we present here up-front but only discovered and formalized the techniques as we went through problems that we needed solutions for to ensure things would go smoother the next time. That learning process will continue and as that happens will we update this book. We might also add content on completely new topics that we are not covering at all in the moment like hiring or effective and enjoyable communication.

Keep an eye on https://redpenguin.tech/playbook for updates.

About the Authors

RedPenguin is a digital product agency that crafts products for clients around the globe. We have our roots in the open source community and create digital products for clients that value code quality, robustness, dependability and honesty. This book was written collaboratively by our team of experts in the fields of product strategy, product design as well as backend and frontend engineering.

Feedback

If you have any feedback regarding the book and the techniques we present, we would love to hear it. Are particular parts of the book not clear? Did we miss anything that is critical to you and your team? Did you run into any problems while adopting our process or had great success that you would share with us?

Send us an email at hello@redpenguin.tech.

License