Ruby Kolesky & Kevin Norris

joyfully

The Joyous Way of Working

joyous

Copyright © 2021 by Joyous Limited

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the authors, except in the case of brief quotations embodied in critical reviews and certain other non-commercial uses permitted by law.

For permission requests, please email pr@joyoushq.com

For more information about Joyous and the authors, please visit joyoushq.com

Table of Contents

About Joyous

Joyous is an open feedback solution for large agile enterprises. We are a venture funded software company - headquartered in Auckland, New Zealand.

Joyous is three years old. Our product and engineering team consists of around 20 people.

We believe employee feedback is a valuable (and finite) asset. One that is too valuable to waste on HR topics alone. So, we built a collaboration tool on steroids.

Joyous starts weekly conversations with employees - focused on specific topics. Organisations use Joyous to enable transformation, improve adoption of tools, or shift culture.

Because feedback in Joyous is open, team members can be a part of the solution. We are the antidote to the traditional anonymous survey.

Joyous is product led. Our product is a joy to use, achieves great outcomes, and leads the way for our organisation.

We are building our third major version of Joyous, a large project that touches every part of our product. This version sets us up to succeed at scale within very large organisations.

Introduction

Over the last year, we have come up with our own way of working at Joyous. We didn't set out to come up with something different. There simply isn't an existing way of working that feels good to us.

Some of us have used Waterfall and Scrum at previous companies. We also tried Kanban and Shape Up at Joyous.

The pros and cons of other methodologies

Waterfall offers a clear structure, defines the end goal early with clean handoffs. It also makes changes difficult, excludes the customer, and delays testing.

Scrum is more collaborative, delivers value often, and makes it easy to change. Longer term, sprints are exhausting, can lead to scope creep, and is challenging at scale.

Kanban is flexible, transparent, and boards help keep everyone on the same page. A lack of clear timeframes, out of date boards or complex boards can lead to poor outcomes.

Shape Up sets clear boundaries on appetite and scope. It allows small teams to deliver sizeable pieces of work without much admin. Low fidelity designs can lead to misunderstandings and it isn't suitable for projects with large unknowns or technical projects.

The major parts of Joyfully

Joyfully is divided into three major parts:

- Strategy and alignment
- Relationships and team structure
- Our process for defining, designing and building features

Joyfully is not perfect, nor finished, but it is working well for us. Where we have ended up is quite different from other ways of working.

Our hope is that you might include parts of what we do as you shape your own way of working. We want more product teams to codify and share their ways of working. Not only will this benefit your own team, but others too.

PART 1: STRATEGY AND ALIGNMENT

Our Joyful Principles

Partnerships over Silos

We have focused on building a strong partnership between product and engineering. One founded on kindness, honesty, empathy and trust. The notion of partnering is just as strong between individuals at Joyous.

Purpose-Driven over Idealism

Joyfully covers everything from strategic alignment through to how we deliver. We have total unambiguous clarity on what objectives are our priority. And, even more important, what objectives are not.

Broad Capabilities over Specialists

We all have a broad set of capabilities, and a compulsion to deliver excellent outcomes. We do whatever it takes, including our leaders. At Joyous leaders work alongside the rest of the team.

Sustainable Innovation over Continuous Sprints

While objectives come from the business, we have the freedom to do our jobs the way we know is best. We deliver large chunks of meaningful work in a way that feels good to be a part of - and doesn't exhaust us.

Collaboration over Ownership

We leave our egos at the door. We seek input early and often - and incorporate good ideas from others. This leads to better outcomes, faster. And allows us to make full use of each other's strengths to fill in the gaps.

Product Strategy

Few methodologies include the basics of forming a software product strategy. A good process for building software is quite useless if you are building the wrong thing.

> "I get asked at least once a week what exactly product does. Forming a product strategy and taking everyone on the journey is the most important part."

Why you need a product strategy

Let's be real. Your primary reason should be to increase revenue. Your goal is to achieve and maintain killer product market fit.

A good strategy ensures that your product is solving real and important challenges for your customers. It ensures your new and existing customers love your product, and won't churn.

Most importantly a good strategy ensures your product is unique. And hard (if not impossible) to copy. This avoids a go to market race in your category.

This is a big part of what it means to be product led. Building a great product is a huge challenge. But doing so makes everything else so much easier.

Having an agreed and documented strategy arms your whole team with the information they need to achieve their goals.

It will help marketing know how to position the product. It will help sales convert deals. It will help operations plan and resource the business. And it will help product and engineering to build the best product.

There are 10 important steps to forming a product strategy, which we will cover off next.

1 - Define your ideal customer profile

It can be tricky to form a clear product strategy if you have many different types of customers. The more different they are, the more they need different things from your product.

> “It's easy to get stuck trying to please two very different types of customers. Often, you end up pleasing neither.”

So, what is an **ideal customer profile** (ICP)? We are a B2B company, which means we sell to other businesses.

An ideal customer profile for us is a description of the type of company that we want as a customer. They are also the company that would get the most value from our product.

We define the ideal customer profile using firmographics, including:

- The average size of the company
- The average company revenue, or average deal size
- The ideal industry or industries
- The ideal locations

Our ideal customer has between 2,500 to 250,000 employees. Particularly large agile organisations in telco, banking, retail and infrastructure. Ideally located in North America, New Zealand or Australia.

In future we will extend into other industries and locations. Having a common view of our ICP now helps ensure we are all focused on the same customer type.

2 - Know your worth

As product leaders, you should know your worth - every which way.

Here are some questions you should be able to answer off the top of your head anytime:

- How many ICP customers do you have?
- How many licensed users do you have?
- What is your average deal size?
- How much revenue have you earned to date?
- How much recurring revenue are you generating?
- How much has it cost you so far?
- What is your burn rate (how much cash are you spending each month)?

If you don't know, find out. To the dollar if possible. And don't just find out today, stay on top of this one every day.

It may feel like some of this is the concern of other parts of the business, such as burn rate. If you are a product leader, you should know.

Do what you can to keep your gross margin as high as possible, 75% is a good target.

3 - Be across your product usage

It is critical that you know how many of your users are actually using your product. It doesn't matter how many licensed users you have if they are not engaged.

So, how do you measure this?

This will depend on your user and your product. Many SaaS companies use a metric called Monthly Active Users (MAU). This is the percentage of users who use your product at least once a month.

We use this metric and we also keep track of a few others that are important to us. Our product initiates weekly conversations with employees. We want employees to answer the question sent from Joyous. After that we want someone from the organisation to respond. We then track the opportunities for change that arise from conversations.

So, there are five metrics we care about:

- Monthly Active Users (MAU)
- Questions Sent
- Questions Answered
- Conversations (when someone else replies)
- Opportunities for Change

Once you can measure, then what?

Not only do we track these metrics, our customers also care about these numbers. They work alongside us to keep them healthy.

Of all the metrics, the most important one to us is Conversations. Why? Hearing back from someone is a key predictor of future engagement. A consistent drop in

conversations causes a drop in Questions Answered. And, longer term it also causes a drop in MAU.

From our data we know that if 60% of Questions Answered turn into Conversations we are in a good place. That is enough to maintain the current MAU. And 70% is enough to stimulate an increase in MAU. So, we will be proactive about hitting those numbers with each customer.

There are many tools you can use for standard product analytics, such as MixPanel. Because these numbers matter so much to us, we include them in our reporting to customers.

4 - Gather feedback

It's easy to assume you know what people think, and you know what's best, after all it's your product right? Trust in the fact that you don't, and ask the people who actually know.

The three big questions here are: who do you ask, what do you ask, and how do you ask them?

Let's look at **who you should ask.**

First ask the people on the ground. The team working on your product will have a raft of good ideas. They will also know the product's limits and weaknesses. For us this includes our product analysts, data analysts, data scientists and engineers.

Your company will have people looking after customer accounts. Your senior leaders are likely in regular contact with your most valuable customers.

They are all getting feedback from customers and even users about your product all the time. Make sure you create a forum and a habit for them to feed it back to you. If you ask them, they'll tell you the uncensored truth.

We recommend you get some internal feedback first. Then use that as fodder to spark conversations with your ICP customers.

It could be that only a few customers will engage with you, but they are also likely to be the movers and shakers. The ones who can see the potential and value of your product, and are your advocates.

Next, find out what your biggest prospects think is lacking in your product. Ask your sales team what barriers are preventing prospects from buying.

> "Prospects can provide insightful feedback. Sometimes as part of an RFP process, which is why your product team should play an active role in large deals."

Okay. So we know who to talk to, next **what do we ask** them?

There are two approaches to consider. The easiest approach is to keep it simple. Ask people what they think you should be doing. Leave it open ended and see what they come back with.

If they share ideas with you ask them to expand on the challenge they are trying to solve.

Make sure you understand:

- The importance of the challenge
- The impact it is having today
- How they are solving it at the moment
- What a good outcome would be. The outcome should be measurable, not simply a new feature.

The alternative to an open ended discussion is to create a series of targeted questions. Or present an initial early roadmap.

For the targeted questions:

- What's working well for them?
- What do they like about the product?
- What's not going so well for them?
- What challenges are they facing?
- How would they rate the usability, stability, and performance of the product?

For each question give them an opportunity to supply suggestions and specifics. When showing an early roadmap encourage them to share their ideas too.

Next it's time to think about **how you might ask.**

In person (or on a video call) will always yield the best results. Run interviews if you are doing a one-on-one, or run focus groups if you need to gather feedback from many people.

If you can watch people using your product you should. Especially if you can sit beside them and allow them to ask you questions as they go.

Another way to ask for feedback is using an online survey. Google forms is pretty easy to get up and running (plus it's free).

5 - Understand the competition

Find out what your competitors are doing. Do a gap analysis of functionality across five of your biggest competitors. If you don't have direct competitors then find comparable companies.

Don't just learn about their features, find out about their strengths and weaknesses. Read reviews about them, talk to their customers, talk to people in the industry.

Figure out what industries they are most successful in, and what their market share is. This might help you uncover a potential key differentiator or industry to target.

We are not a competitor to apps like Messenger, Whatsapp, MS Teams or Slack. Yet we are a conversational tool. So, we analysed this category too. This helps ensure our user experience meets the norms everyone has come to expect.

6 - Watch out for emerging trends

Once you start gaining traction it's all too easy to become complacent. You should never take your eyes off what's happening in your industry.

You should also stay close to what is happening in your customers' industry. Some lighthouse prospects might have created their own internal solution. Use your network to see if you can arrange a call to understand more about what they have done, and why.

Finally, keep across the technological advances of your platform. Occasionally there are big shifts that you cannot ignore, and it pays to stay ahead of them.

7 - Do a SWOT analysis

If you've made your way through steps one to six you should now be in possession of bucketloads of data. It's time to analyse it.

We see no need to reinvent the wheel, simply compile what you have into a SWOT analysis. SWOT stands for Strengths, Weaknesses, Opportunities and Threats. Arrange the items in each from highest to lowest impact.

You should have many correlating data points on some items. Play close attention to those and place them at the top.

You might discover at this point that a product strategy consists of more than just a set of features. Awareness, perception,

marketing, resourcing, and other factors may also be worth considering.

The SWOT informs and validates your product strategy.

Experienced product leaders and industry experts will also have excellent intuition. If the information is validating your intuition, then go with it.

8 - Understand your market potential

As mentioned earlier our ICP consists of large organisations. In time we will expand out to all industries, and locations.

This means our total addressable market (TAM) is enormous. We use the stats from https://stats.oecd.org/ to determine how many potential users we could have within large organisations in the OECD. Then we multiply that by our average price per user per year.

Consider how much of the market you currently have. Ensure your product strategy gives you a real shot at gaining a solid chunk of your TAM.

9 - Define your vision, goals and objectives

At this point you should be ready to articulate your product strategy. Next you need to get ready to present it.

If you are a product team worth your salt you should be good at doing presentations. If you are not, get someone to help you. Make something beautiful to back your vision.

A **product vision** should be a short and concise statement of what you are trying to achieve. This is the appropriate time to be ambitious rather than cautious.

The purpose of Joyous is to make life better for people at work.

The internal product vision for 2021 is to make life better for people at work - *at scale.*

Your **goals** should specific and measurable.

Avoid vague statements like:

- Increase revenue
- Get more customers
- Simplify data integrations

Instead be specific:

- Increase recurring revenue to $50 mil in 2021.
- Add 3 million new users by 2023.
- Reduce manual data integration effort by 80%.

You get the idea.

With goals like these everyone understands exactly what you are working towards. And you can measure and track your progress.

Now you need to get to grips with the **objectives** you need to undertake to achieve those goals.

Double back to the insight you gained in the preceding steps and summarise them at a high level.

Turns out there are three big things we need to do:

- Drive long term engagement through conversations.
- Support flexible conversations - ask any cohort anything, on any frequency.
- Make it easy to self-manage campaigns across many cohorts at scale.

The product vision, goals and objectives all go into your product strategy deck.

Include a few choice quotes, and trends of product usage over time. We also use high fidelity prototypes to help people see our vision.

Make the presentation as concise yet compelling as possible.

10 - Present your product strategy

It’s time to present the strategy. First present it to your team. They will give you good feedback. Update your strategy accordingly. Rinse and repeat this exercise until you've spoken to all internal stakeholders.

It's okay to be confident at this point. You have enough data to back up the plan. You are not asking for permission to make a success out of your product, it's your job to do so.

Next go back to those customers who gave you feedback and present it to them. They will give you good feedback, and guess what? You’ll adapt the strategy accordingly. Rinse and repeat until you’ve worked your way through all your key customers.

Connect with industry experts and present the vision to them. They gave us the most thought provoking feedback. And even at this late stage we continued to update our strategy.

By the end of this you should have a clear product vision - along with the buy in of your team and customers.

Never stop validating the strategy

Creating your initial strategy, while it's a huge undertaking, is only the first step. You should be re-evaluating and validating the strategy, at least every quarter. Never be afraid to change direction if you find a good reason.

If you are leading a product, then you simply must develop a product strategy. You are the brave game changers, the innovators, the voice of your customers. You are the funnel that can filter out the muck that isn't adding value. Be open to change and be bold in embracing it.

Strategic Alignment & The Negative Roadmap

> "Being 100% aligned enables us to completely focus on delivering the prioritised objectives."

Initially we struggled to get strategic alignment on objectives. The approach outlined below helps us to achieve total unambiguous clarity.

Align senior leaders

We formed a strategy squad. This includes our co-founders, and the leaders of product, engineering, operations, marketing and sales. Every three months this squad collaboratively agrees the objectives for the entire business. This includes the product and engineering roadmap.

At Joyous we don't sell features as part of the sales process. We also don't sell on the promise of our roadmap. We only sell the product we have. Because of this we don't need to rush work to meet deadlines set by sales conversations.

This allows us to focus on the things that will make the biggest impact to all our customers, and allows us to build quality code.

Positive & negative roadmaps

A concise bullet point describes an objective at this level, which we place on a *positive* roadmap.

We put important items that we are not doing onto a *negative* roadmap. This was a game changer for us. It means that there is no room for ambiguity, nor overloading.

We are either doing something, or we are not. We cannot stress enough how helpful this has been. If there's one thing you take away from this book, make it the *negative roadmap*.

> "Too often teams waste precious time and energy struggling with alignment. Using a negative roadmap has completely changed the game for us."

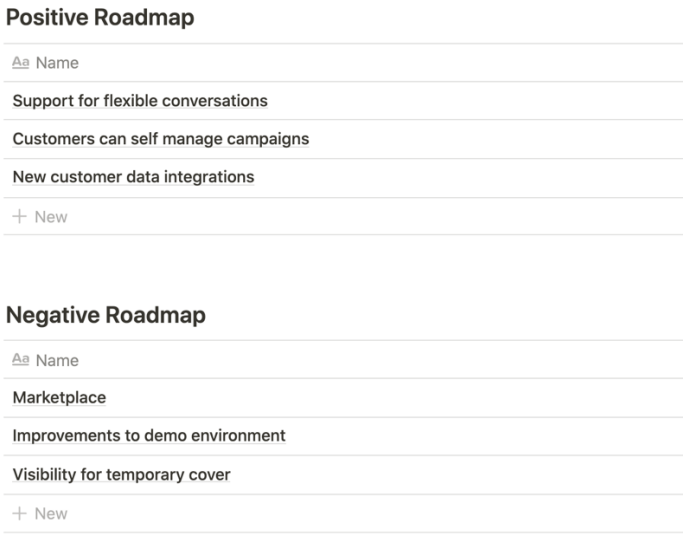


Figure 1 A simplified version of our strategic positive and negative roadmap for Q1 2021.

Stay focused and build quality code

If something big crops up - the strategy squad decides the priority together. Nothing slips in without consensus. It either goes onto the negative roadmap, or onto the positive roadmap. If it goes onto the positive roadmap, something else needs to come off.

We will respectfully say no to feature requests that are not aligned with our purpose.

We have a simple way to prioritise smaller items separately from the large objectives. Once a week we consider and prioritise a few items.

> “In the past I have seen many things cause a product to head in the wrong direction. I've seen products become terrifyingly complex, or appear stagnant to customers.”

There are no separate resources set aside for tech debt. As part of doing the feature work we strive to build the best solution possible. This *includes* fixing any tech debt. We don't always get this right, but we are always improving the code base.

Eat your own dog food

We use Joyous ourselves. We use it to ensure a great employee experience and to keep our team aligned. We especially developed our Team Performance conversation set for this purpose. This allows us to have asynchronous input into the objectives, and our way of working.

Unified Engineering

We have our product strategy. We have strategic alignment. We have a single prioritised roadmap.

Unified Engineering allows us to map our efforts directly to that roadmap.

At Joyous, all engineers are part of a single team. Our team has shared ownership of the whole codebase. Engineers work on roadmap items in priority order.

As items complete the freed engineers start on the next item or help with ongoing items.

> “Unified engineering is the opposite of the Spotify squad model. No silos, no dependencies, no arbitrary splits. One roadmap, one engineering team.”

Unified engineering consists of the following principles:

Shared ownership of the code base

All engineers are full stack and are able to contribute to all portions of the codebase. No individual or group of individuals has ownership over any section of code.

The code base is considered as a whole, not as parts.

A shared understanding of the code base

This is very important to enable all engineers to work on any part of the code base.

We achieve this by:

- Developing a code base that is homogeneous in nature. Moving from one section of the code base to another is as simple as possible.
- Ensuring there is awareness across the engineering team on how the code base is changing.
- Describing changes that are occurring in the code base. We do this in weekly catchups and presentations of how new features work under the hood.
- Ensuring that every piece of work is supported by an engineer that has a deep understanding of the area.

A shared set of tools, standards and practices

If we want a shared understanding the code base must be easy to understand.

We achieve this primarily through two means:

- Limiting the number of tools and technologies we use.
- Using a shared set of standards and practices across the code base.

For example at Joyous we use TypeScript for everything.

- Our front end is TypeScript (React)
- Our backend is TypeScript (NodeJS)
- Our scripting is TypeScript (NodeJS)
- Our CI/CD is Typescript (Custom Github actions)
- Our infrastructure is codified in TypeScript (Pulumi)

We have a single defined way of writing TypeScript. We enforce this standard through compiler options, tools and pull request reviews. We use Prettier (code formatting) and ESLint (static code analysis).

A focus on keeping the codebase simple

To keep the cognitive load of engineering low, we write as little code as we can. We try to only write code that deals with business logic. We try to keep business concerns simple and separated from one another. We don't force separation as this adds its own flavour of complexity.

A few guidelines we follow:

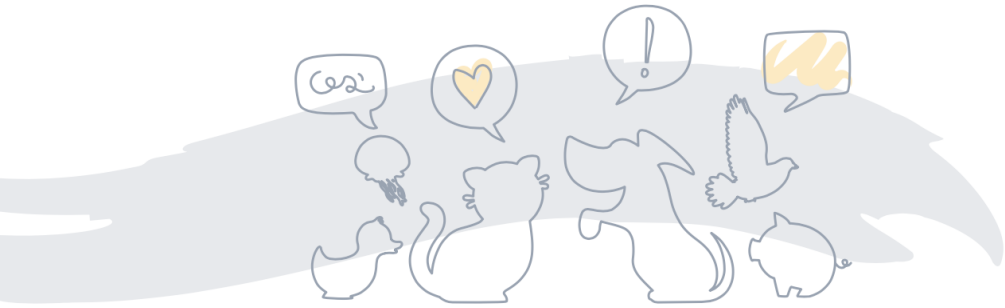
- Prefer managed solutions
- Prefer the simplest solution that meets the requirements

- Build for known use cases and known future requirements, not for possible futures
- Abstract and extract stand-alone application complexity to self-contained modules.

> “Unified engineering leads to needing fewer engineers. All engineers are focusing on the top business priorities at all times.”

With Unified Engineering we achieve our goals with a smaller and more efficient team. Engineering Crews describes how we scaled unified engineering.

PART 2: RELATIONSHIPS & TEAM STRUCTURE



Our Organisational Structure

At Joyous people learn more, faster, than they could elsewhere. This is true across our entire organisation.

Three key factors enable this: **collective leadership**, **partnerships**, and roles with **broad capabilities**.

This approach means we have fewer layers and slices in our people organisation. Not only are we streamlined, we are also vastly more productive. Why? Because this model builds trust, and reduces bureaucracy.

Collective leadership

At Joyous we don't arrange leaders in a hierarchy, one below each other. Instead we share roles at a senior level. Our CEO role is currently shared. We plan to extend this approach to other senior roles in future.

> “This is a hybrid approach between a co-role and a rotating role. Two to three people share the role. The operational part of a role rotates each quarter. Other aspects of the role are always shared.”

The benefits to a collective leadership approach:

- Detaching leadership roles from a single personality
- Encouraging teamwork at a leadership level
- Rapid learning for those new to the role
- Better outcomes and more innovation
- More energised leadership
- More support and less stress in challenging roles
- Greater work life balance for leaders
- Built-in succession planning

The more we experiment with the notion of collective leadership the more benefits we see. In future as we grow, we will share other leadership roles as well.

Partnerships

Strong partnerships exist across functional disciplines in our organisation.

Sales and Marketing have a strong partnership to achieve our revenue objectives. Sales and Product have a strong partnership to turn pilots into paying customers. Product and Engineering have a strong partnership to meet our product objectives. And so on.

We are all focused on the same strategic objectives, and working hard towards them. The shift in thinking is towards being collaborative and supportive.

> “We are often asked how we hold each other accountable. This question simply doesn't align with our partnership dynamic. We all agree our OKRs, and are collectively accountable.”

The notion of partnering is just as strong between individuals at Joyous. When a new feature is being designed, it is common for two product people to pair on it. When a feature is being coded, an engineer will never build it alone.

Broad capabilities

We all have a broad set of capabilities, and a compulsion to deliver excellent outcomes.

Initially we worked this way by necessity, because we were a start-up. Over time we made a decision to continue this way, rather than hiring specialists.

We waste less time passing the parcel between individuals with different specialist roles. This reduction in noise makes us both more efficient and versatile.

We also find that roles with broad capabilities is a great way to grow and stretch people who are eager. So, we hire people with this orientation from the outset.

Let's compare a traditional organisational structure to ours. We will focus on product, engineering, design and customer care.

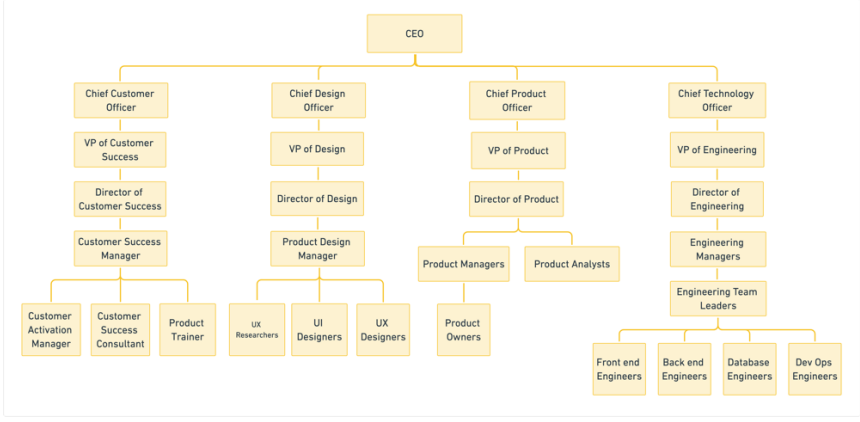


Figure 2 Traditional organisational chart, up to 7 layers and 12 slices.

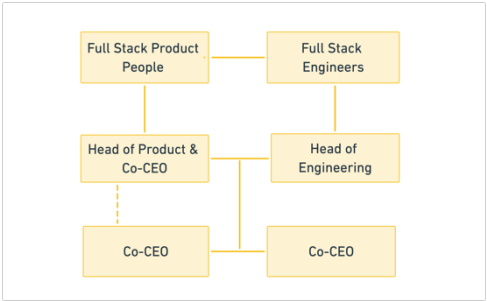


Figure 3 Current Joyous org chart, 3 layers, 2 slices.

Let's examine the key differences between these two org charts:

- **Orientation.** The traditional org chart has the most senior leaders at the top, representing a reporting line. Our org chart has our most senior leaders at the bottom, we think of these as supporting lines.
- **Connections.** There are no lines connecting roles across the same layer in a traditional org chart. Our org chart has connecting lines. This represents our collective leadership approach, and also partnerships.
- **Complexity.** The traditional org chart has up to 7 layers, and 12 slices. Some organisations may have fewer layers, and slices - depending on their size. Our org chart has 3 layers and 2 slices.

Now, you may wonder how we intend to scale our approach as we grow. We can't say for sure, but we aim to extend our model as shown in the next figure.

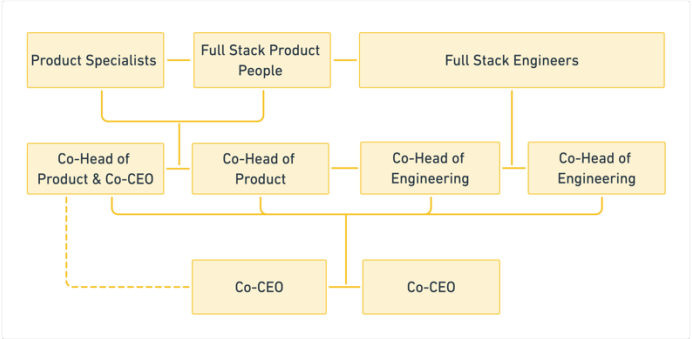


Figure 4 Future Joyous org chart, 3 layers, 3 slices.

The Product & Engineering Partnership

Having a meaningful partnership between Product & Engineering is critically important.

We are open, honest, empathetic and vulnerable. We bring our perspectives and expertise to the table and *we listen to each other*.

> "Our most important objective was to focus on building a partnership. It took six months to get to a point where it felt completely natural."

Everything we do, whether it is engineering or product oriented - we figure it out together. We invest time and care into the relationship.

For the most part we do not have separate meetings, product and engineering are usually catching up together. Our stand-ups are combined, and we do regular retrospectives together.

> "Spending quality time together outside of the process is another way we remain close."

There is no topic that is off limits between product and engineering. From scope, to ways of working, to hiring, to tooling, to just about anything you can think of.

Product isn't there to browbeat engineering into delivering on tight deadlines. Instead product is there to support engineering to deliver the right scope of work - and vice versa.

If an engineering challenge crops up then product helps. If a product challenge crops up, then engineering helps.

A strong bond and mutual trust between product and engineering teams is rare. Combine this with strong competency and a good product strategy - and you have a recipe for success.

Full Stack Product

At Joyous the product team also looks after our customers. *And* some of our product managers are also designers. We call this approach **full stack product**.

Using this approach we combine nine roles from product, design and customer care into one person.

We didn't set out with the intention of running this way, it simply evolved over time. Our first product people also had good design skills. So product and design has been part of the same role from the start.

Over time we noticed that having other people take care of our customers wasn't working well. So, we decided to experiment with product looking after customers.

If you think about it, the skills that product managers have are ideal for working with customers. They are experts in the current product, and they know what's coming. They are collaborative and organised. They are comfortable presenting to an audience. And, they prioritise work and set product direction.

> “We thought it might be more efficient, and result in better outcomes for customers. So, we decided to give full stack product a go.”

Unified product

In the same way we have Unified Engineering, so too do we have Unified Product.

For us this means:

- We share ownership of the product, rather than divide it into parts.
- We have a shared understanding of the product, and where it's headed.
- We have a shared understanding of our customers, and how they use our product.
- We have a focus on keeping the product simple.
- We have a shared set of tools, standards and practices. This extends across customer care, product management and design.

Stretching our capabilities

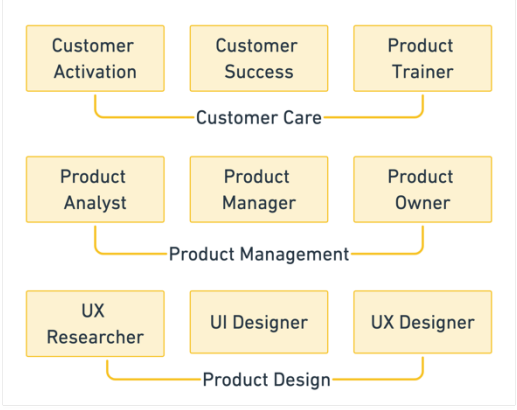


Figure 5 Roles combined in full stack product

It is rare to encounter a product person with all the skills and experience to be full stack. It can take six months to a year for someone to stretch across most of the stack.

> “One person started out focused on product management. Over the next six months they stretched their design capabilities. After that they stretched into customer care.”

People will need the time and space to stretch. They will also need support, and mentoring.

A shared set of tools, standards and practices is essential to make this work.

We often have one person lead something out. They will codify it and make it as efficient as possible. Then others will follow from the same shared starting point.

After that it will keep evolving, but someone leading helps us get there faster.

How to scale full stack product

Hiring people with the right potential is important. Try and hire people with experience in at least two of the disciplines.

Even more important, is someone's willingness to be full stack.

We currently have four people who are full stack or stretching.

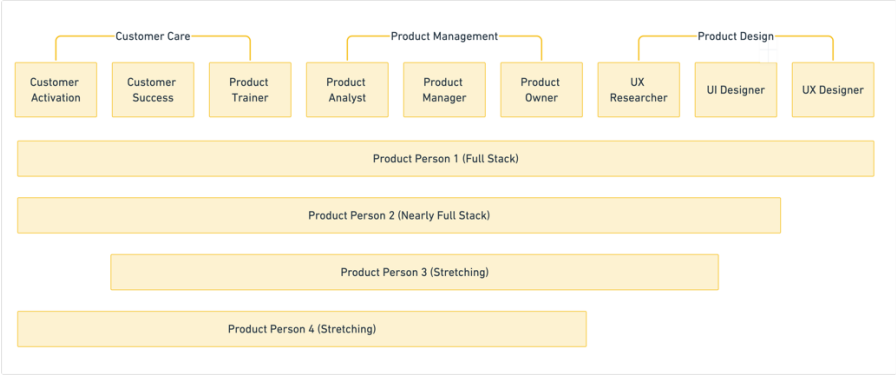


Figure 6 Stretching our product team

We also have a growing customer base. So, we intend to hire product specialists soon. Their focus will be on customer care. They will still be part of our product team and have a strong influence on product direction. While this is a deviation from our principle on broad capabilities over specialists, it is necessary. Our full stack product folks will still continue to care for customers. They will *not* become specialists in the other direction as a result. This is important as having *some* full stack and *some* specialists will allow us to scale.

Benefits of full stack product

Here are some reasons why we like this approach:

- **Better outcomes for customers.** When product people look after customers, they develop empathy for them. They also understand the challenges customers are trying to solve with our product first hand.
- **Faster outcomes for customers.** When product people cover the entire stack it becomes more efficient. This may sound counter intuitive, but for us it's true.
- **Less room for misunderstandings.** There is no longer a pass the parcel effect across the disciplines. Reducing

the likelihood that something gets misinterpreted between the customer and product.
- **Productization of customer process.** The relationship and process around the product is more likely to become part of the product. This is critical to our success. We want to scale our customers, without also having to scale our (people) organisation.

Engineering Crews

Over time, as the size of engineering grew, collaboration became more and more difficult. Adding more engineers to a single team lead to increased silos. Individuals were trying to keep themselves from excess cognitive load and context switching.

We developed the concept of Crews to maintain efficiency with more engineers. Crews are designed to maintain the benefits of unified engineering while scaling.

Crews developed organically from our experimentation with Shape Up's *Pitch Teams*. Crews allowed us to get back to efficient engineering group sizes. Group sizes where collaboration is simple and context switching is rare.

The basics of crews

Crews are still part of a single engineering team. Crew membership is fluid and shifts depending on the needs of the business.

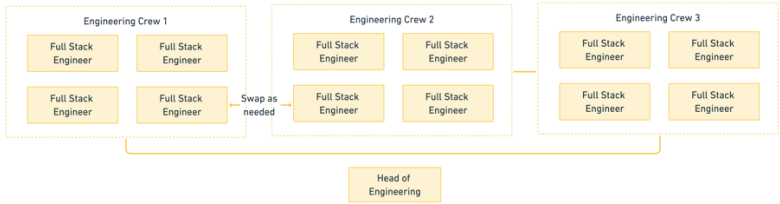


Figure 7 Engineering Crews within Unified Engineering

- Crews are semi-permanent groups of engineers. They re-form as needed or bi-annually. We aim for four engineers, this can grow / shrink as required but no less than two.
- Crews take on roadmap items in order of priority.
- Crews deliver end to end features.
- Crews work closely with, and are supported by product and design.

To showcase our fluidity, one of our crews started a feature that involved a lot of investigation and proof of concepts. A small crew is more efficient for this type of investigative iteration.

So, two people focused on this first. In the meantime, the rest of this crew joined another crew that was finishing up a feature. They helped get through the remaining bugs and got the release out faster.

The benefits of crews

- **Faster turnaround of high quality solutions.** Semi-permanence allows the crew to form as a team and get into a rhythm. With engineers in a crew working on the same features they deliver faster, and the quality of the solution benefits from all their input.
- **Rapid learning and growth.** Crews are an ideal environment for learning, as engineers are always working together on the same features. Because crews

are fluid, engineers are also able to work with many of their peers over time.
- **Connected but still autonomous.** Crews enable engineers to be autonomous within the bounds of unified engineering. Crews can have a high level of autonomy as they are working on an entire chunk of work.

The practices of crews

To ensure that crews don't result in siloed knowledge we have practices to keep the rest of the team in the loop. This includes regular catch-ups and presentations. You'll learn more about those and our other habits in Part 3.

PART 3: DEFINE, DESIGN & BUILD FEATURES

Overview

Developing Joyfully is an on-going and organic process. The process we describe in the book is the process we follow most often.

The stages of Joyfully

Stage 1: Product scopes, designs and refines a chunk of features.
Stage 2: Product and engineering align around the details.
Stage 3: Engineering crews build the features.

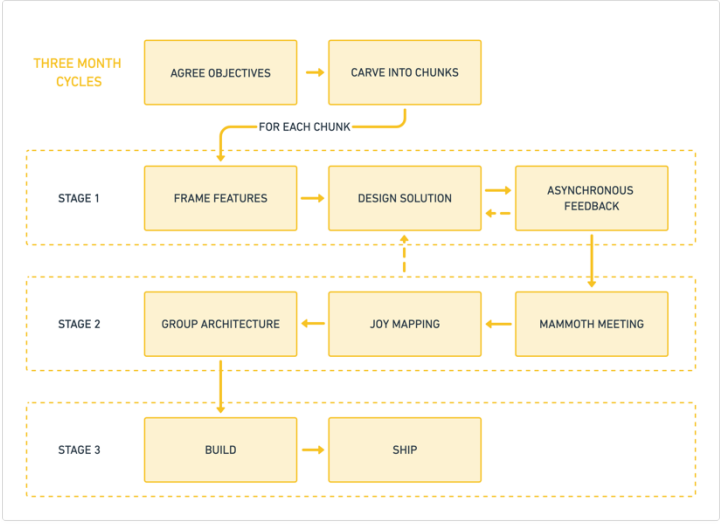


Figure 8 The Joyfully process

Multiple chunks are running through our process at any given time.

Due to the current size of our engineering team, we typically have three chunks being built at a time, each by an engineering crew. Soon this will expand to four.

Flexing the process

For all objectives we follow the path that makes the most sense for the work instead of adhering strictly to process.

Occasionally we will batch many chunks through one step at the same time. For instance: for our *Fully Flexible Conversations Objective* we defined and carved up all the features for four big chunks at once.

Our engineers also designed the Group Architecture for most of the work upfront. This happened at the same time as product began to design the features.

For engineering led features, such as codifying our multi-regional infrastructure, the process is once again flexed. In this case there was a lot of investigation and prototyping along with Group Architecture that occurred at the start. There was no requirement for Design and thereafter the process continued as usual.

Our joyful practices

A weekly organisation wide showcase. Once a week our entire organisation comes together to share our work with each other. This includes all 30 people in our organisation across all locations and functions. The idea is to share work early and often, rather than waiting until the end of a piece of work.

A weekly Product & Engineering catchup. Once a week we all discuss:

- Each item the engineers are working on at a task level, and demo any visual work.
- We will unpack any risks such as work taking longer than anticipated, or technical challenges.
- At a high level we gauge how each chunk of work is going, and update our forecasts.

- We will discuss and plan how to ship our releases.
- Product will share upcoming work, and share updates about the roadmap.

A weekly Product catch up. Product come together to discuss:

- Each ICP customer. We review any updates, tasks and metrics for each customer, and form a plan if needed.
- Product updates. We will check-in on each other and make sure we are on track, and offer support if needed.
- Other projects. Often product has other large projects on the go. For example: we recently launched a new Help Center for customers.

A weekly Engineering catch up. Engineers come together, and discuss:

- How the code base is shifting.
- New concepts entering the code base.
- New patterns or tools that are being introduced.

Engineers share regular presentations of new features under the hood. In enough detail to make the code approachable. The crew that completes the work presents it to the rest of engineering. This exercise is great for ensuring technical documentation is clear.

A bi-weekly retrospective. Product and Engineering will have a regular retrospective. There isn't a standard format for retrospectives, each week is different. Sometimes they are serious, and other times they are fun!

> "Once we did a retrospective where we each described the V3.0 project as if it were a movie. We gave it a title, a genre, a main character and a twist. It was heaps of fun."

Carve Into Chunks

Taking an objective and defining the solution is hard. It's also the place where many of us get it wrong.

Pre-2010 most software product teams were using a waterfall approach. They were designing large features, sometimes entire products upfront. And they were doing so well ahead of engineers building them.

Then gradually most of us shifted to agile. Features got broken into smaller and smaller units. Eventually they got so small that we lost sight of the big picture.

One thing we love about Shape Up is that small teams build decent chunks of work. And they are not confined to short sprints.

At Joyous we favour this approach too. Here are the steps we take to carve objectives into smaller chunks.

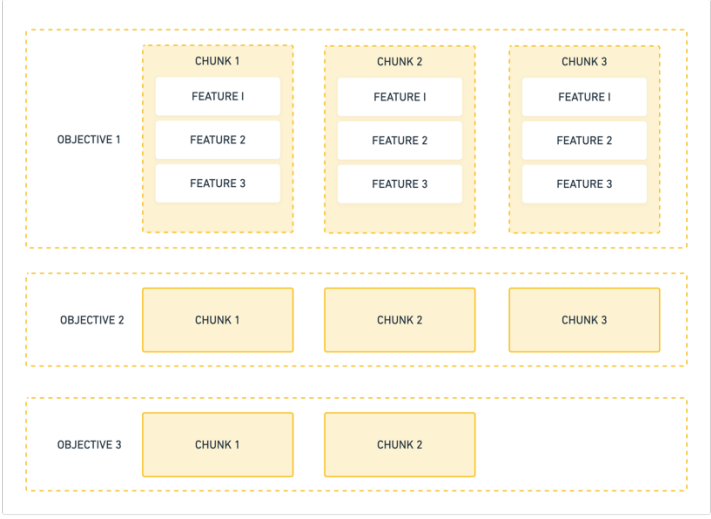


Figure 9 Breaking objectives into chunks of features

Carve objectives into chunks

- We take the objectives from the positive roadmap and carve them into smaller chunks. Each chunk is deployable, and consists of many features.
- It should be possible to build these chunks in parallel. So, there shouldn't be too much interdependency between them.
- Next, arrange them in the order it makes most sense to build them in.
- For each chunk:
    - Create a list of all the features that you'd like to include.
    - Divide the feature list into two parts. Those that must be in the first release, and those that can wait until after.

Let's take a look at the major chunks of our flexible conversations objective.

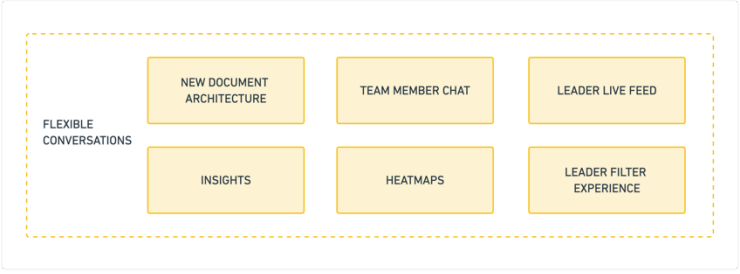


Figure 10 The six chunks of Flexible Conversations

> "Each feature you include in the first release extends the time to initial release. Try and set aside as many features as possible for later."

Next, let's look at the features for some of these chunks, divided into the first release, and those that can wait.

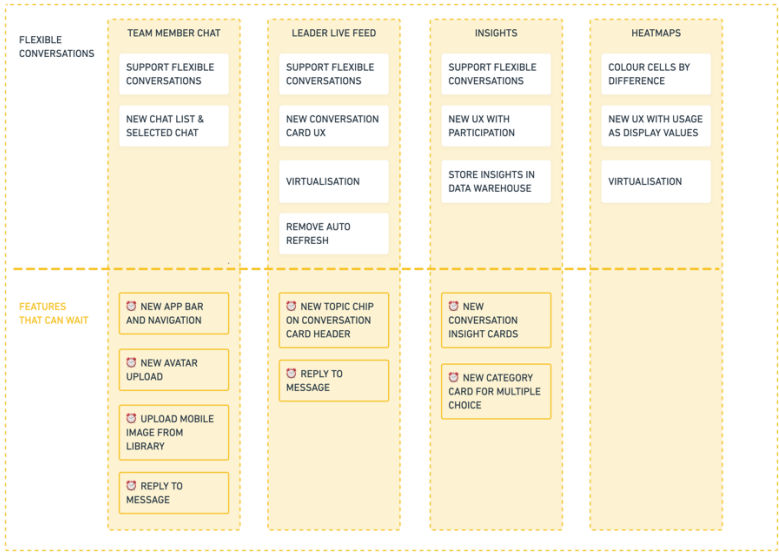


Figure 11 Four chunks with first release features (white), and features that can wait (yellow)

Features that can wait

Keep trying to set aside features. You'll be attached to including features in the first release that can likely wait. It's not to say that you won't build these features. Although separating them out leaves the door open.

The goal is to shorten the time until a customer first experiences an improvement. Once the first release is out then you can deliver the other features in a constant flurry thereafter.

Before going any further socialise your plan. This allows other stakeholders the opportunity to refine the plan further without wasting time designing the wrong features.

Important considerations when defining features

Start with the problem. We create a document called a Joy Map in which we map out the details of each chunk. Our first step is to describe the problem.

This includes:

- The challenges we are trying to solve.
- The short term impact of not solving the challenges.
- The long term impact of not solving the challenges.
- The high-level customer use cases that relate to this challenge.
- The success metrics we should consider as we design our solution.
- Links to related information that provide more background and context.

We continue to flesh out the details of the Joy Map later, after the solution has been designed in Figma.

Deliver tangible value. A product shouldn’t go too long without delivering value that a user can see and experience. Delivering changes only in the background doesn't feel like progress to customers.

Manage risk for significant changes. Three big risks for significant changes include:

- **Time to release.** Aim to deliver tangible value as soon as possible. And then continuously add more value thereafter.
- **Logical progression.** Don't shift the product so much that you can no longer sense check what has come out, or that users feel lost.

- **Future proofing on a hunch.** If you are not solving a real problem validated with data - from many correlated angles - then you are making stuff up. Don't waste time making stuff up.

Poorly written code is a kill joy. While you want to work as efficiently as possible, rushing significant changes out the door is a bad idea. If you need to get work out in a hurry, it's better to reduce functional scope over compromising on quality. No engineer wants to spend their life up against a code base built in haste. This will also cost you and your customers in the long run.

Be transparent, always. We forecast how long we think the work will take, in actual time - updating our forecasts weekly. Our Joy Board in our Notion workspace has two views. A Gantt chart, and a Kan Ban view. All large chunks of work are forecast and described here. This includes work across all our engineering crews and data science.

We share our forecasts with all our stakeholders, customers included. Towards the tail end of a long project we will set a hard release date. We commit as a collective, rather than having a date set on anyone's behalf.

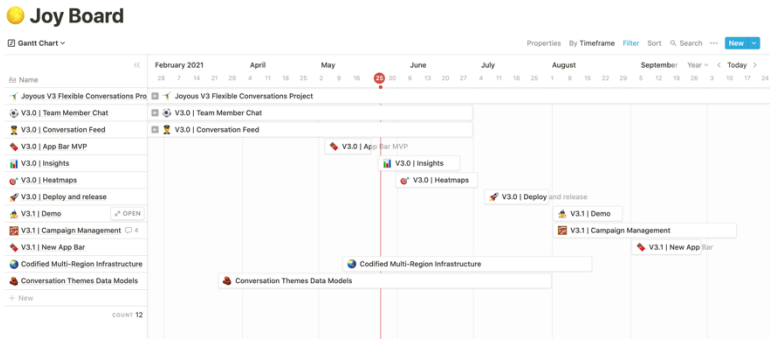


Figure 12 The Joy Board - A Gantt Chart in which we forecast all work

Frame Features

Going deep in design

We hit two challenges when using the low fidelity approach from Shape Up:

- **Too tight for deep design thinking.** In Shape Up designers create the final designs at the same time as the engineers build the solution. We prefer a lead time that allows us to iterate on the design with less engineering rework. For V3.0 of Joyous we also needed more time to create our new design language and design system.
- **Complexity isn't obvious.** Our engineers missed something big when doing the technical shaping from fat-marker sketches and breadboards. Yet the moment they saw the final designs and prototype, it was completely obvious to them. We landed in a situation where we were making big decisions mid build. We handled it well and made a good decision fast, but we wanted to avoid that in future.

So, now we go deep in design ahead of a build. And collaborate heaps during this process. We ensure there is a clear path for the work to be done as efficiently as possible at build time, without being held up by design.

When it comes to our design language we don't try to reinvent the wheel. Instead we use material design and customise it to our brand.

We use Figma as our design tool. Inside of Figma we have a defined structure for our design system, projects and files.

Our design system

We follow the Atomic design methodology. Before build time we ensure our designs are accessible. Our designs conform to Level AA of the Web Content Accessibility Guidelines (WCAG) 2.

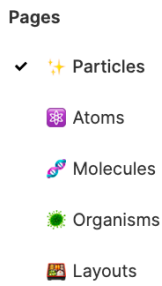


Figure 13 The Joyful Design System – Atomic Design Pages

This version of Joyous involves a full UI overhaul. So we have created a new design system as part of this project.

We build components with a 1-1 mapping to engineering components and use Figma variants for different styles and states.

Figure 14 Atoms with variants inside our design system.

Our Figma project structure

- We create a project in Figma (in this case for Joyous V3.0)
- We created a 1-1 mapping between the project Figma files and the project chunks.
- We use the same file names across Notion (on the Joy Board) and Figma.
- We include in the titles the status of the file [❤️ not started], [💛 in progress], [💚 ready]

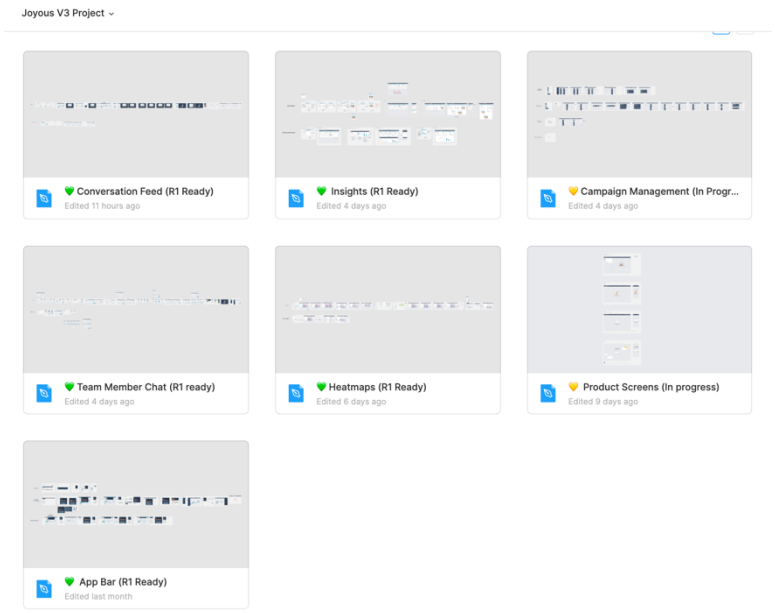


Figure 15 - Our V3.0 Project Structure

Our Figma file structure

For most of our chunks we will frame up the features in Figma. We mean this quite literally. We have a Figma File Design Template which is the starting point for each chunk.

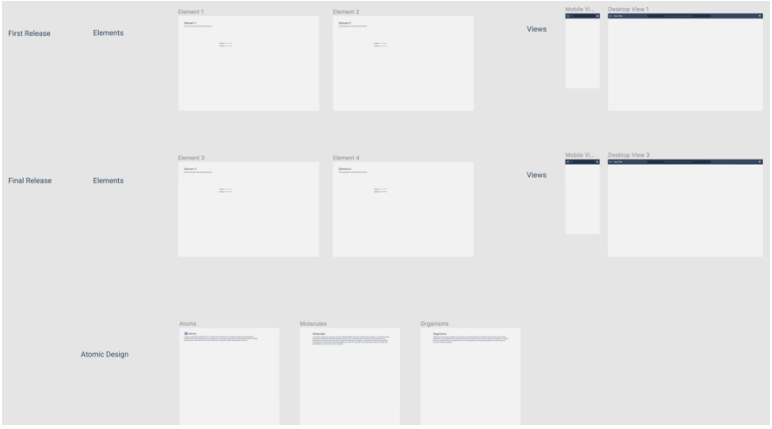


Figure 16 Our Design File Template

We create a Work in Progress (WIP) page in each file. Within the page there are two swim lanes. The top lane contains a series of frames defining the first release features. The bottom lane contains a series of frames defining the features that can wait.

Each frame has a feature name. We will map out all the frames before we begin to do the visual designs. This is what we mean by *framing the feature*.

At the end of each lane we will show final composed desktop and mobile views. We create another page in which we build a prototype demonstrating the UX of all the features.

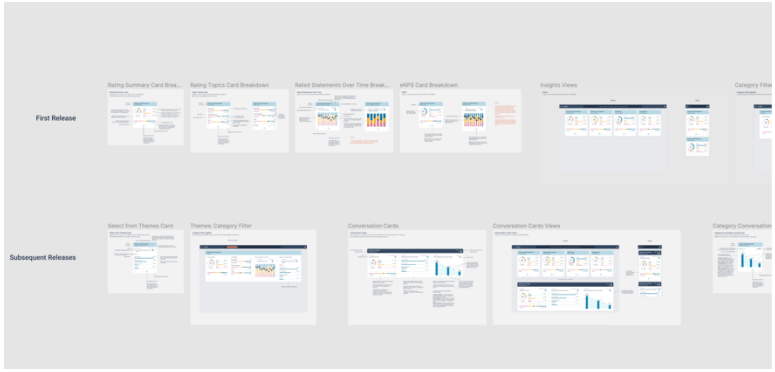


Figure 17 The lanes of the Insights chunk

For each major revision we will create a new WIP page, and rename the previous WIP page with a version number. Once we reach a point of readiness we will create a For Dev page into which we place the final frames ready for review.

Figure 18 Page structure in Figma File

Design solution

Once we have our features framed it's time to start designing our solution. Two product people will often map out the frames together. Then one will lead out the exploratory phase of the design.

Before the exploratory design is completed we will pass the feature to another product person. This occurs somewhere between 50% - 80% completion. The other product person will then wrap up the exploratory design and create any new components required.

Sometimes the features are split down the middle and designed at the same time by two product people in the same file.

Pairing and sharing the design works well for us. Involving two product people at the outset ensures the design is considered from multiple angles.

Below are the steps we take when designing a solution. We don't follow a strict pattern and often our process is messy in between, but we always end up at the same end point.

Make a prototype

Building prototypes in Figma is ridiculously easy.

Often building a quick prototype first is the best way to start for a feature that has a lot of interactions involved. It will help you uncover and solve the challenges quickly.

Even for a feature that isn't, having a prototype will clarify a lot for engineers, so we always build one at some point in Figma.

Finalise the views

We often start by composing a view. We love using layout grids, for these in Figma, it makes building adaptive designs so much easier. If we have existing components we will begin by dropping those, if we need to do something new we might just build something rough to start and turn it into a component at the end.

Flesh out the frames

Inside each frame we will showcase visual elements for that feature. These frames include annotations in a group. Grouping annotations makes them easy to hide when you just want to see the design.

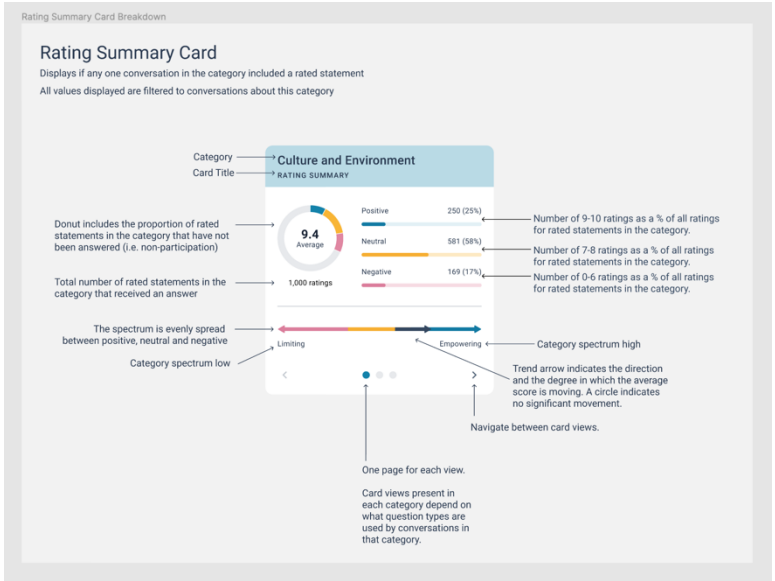


Figure 19 A framed feature with annotations

Build the new components

One of our product folks prefers to build components from the outset of designing a feature, the other prefers to do it at the end.

Either way is fine. Before the feature gets passed on for asynchronous feedback, any new components will be created in the design system. And the rough designs will be replaced with components.

This ensures any future changes will be completely seamless across all our projects when we update a component in the design system.

Be predictable

It may seem over the top to have a structure described as explicitly as we have. Having each feature structured, described and designed in the same way has created a predictable way in which each feature will be developed regardless of which product people are working on it.

Asynchronous Feedback

Asynchronous collaboration occurs in Figma using the comments functionality. Everyone at Joyous is at least a read-only user in Figma.

Our product folks will invite the entire company to review a chunk once the initial design is ready. We slack a link to the Figma file to everyone and invite them to make comments.

This is great because comments are made right in the place we need to consider them.

Our product folks either:

- Respond with an explanation,
- Discuss further,
- Or if it's obvious adjust the design, annotations or prototype immediately.

Comments remain open until the discussion closes or adjustments are made.

This asynchronous collaboration continues throughout the remaining stages of the process.

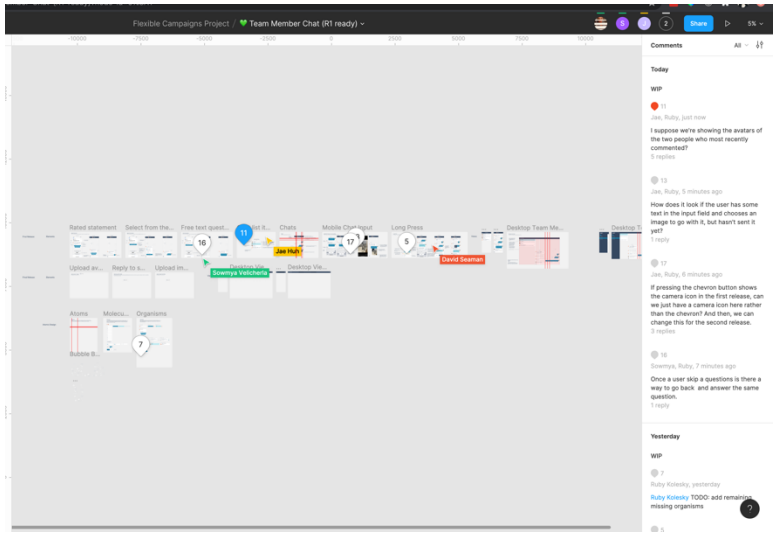


Figure 20 Many people commenting at once shortly after a design was shared for review

Mammoth meetings

We invite the entire organisation to join us for a one hour decision making meeting. We call it a Mammoth Meeting. This happens after we have processed all the feedback from the comments in Figma.

In this meeting people have the opportunity to raise any final items for discussion. We do an in-person meeting combined with a video call for people who are remote.

The practical challenge of a meeting like this is the high time cost. We are usually covering substantial ground with a large group.

We make this as time efficient as possible using a decision making framework.

It is more effective when the product person leading the solution doesn't run the meeting. Someone else acts as the facilitator, and someone from the team takes notes.

We have a big screen with the solution and Zoom call up on it, and we make notes on screen as we go.

The decision making framework

Ask everyone to list the items they want to discuss. At this point only note the item down without discussing anything in detail.

Work through each discussion item one at a time.

Step 1: Listen to the person who raised the item. Allow them to explain their concern and explore possible options. Other people can take part too. Avoid going into too much detail.

Step 2: Select two options. Avoid lobbying for one approach over the other. The idea is to identify what the best options are quickly.

Step 3: Go to an early vote. By show of hands count the votes for each option.

Step 4: Listen to the minority. For the option with the least votes, ask those who voted to explain why they favour this option**.** It's important to listen to the minority first.

Step 5: Listen to the majority. For the option with the most votes, do the same.

Step 6: Revote. Once everyone has had a chance to comment, go to a revote.

Highest votes win. It's as simple as that. Acting as a democracy, accept the vote and move on. If the vote results in a big shift from the original approach it should be validated with a design before finalised. The product folks do this and present an update at the next showcase. They may also lobby for an alternative approach if issues arose.

After you've voted on one item, move onto the next until you've worked your way through all the discussion items.

If you get good at this, you can save hours of time. And you can get excellent outcomes by giving everyone a voice.

> "It took 20 minutes to work through the very first item ever. Two people kept debating over options. After that we got the hang of it and moved through the remaining 8 items at around 2 minutes per item."

Joy Mapping

The purpose of Joy Mapping is to achieve clarity and alignment on what the crew will build. And, what unknowns remain for the crew to investigate during build time.

This process runs between a product person and the relevant crew. Together they map out the slices of work the crew must build to deliver the features of the given chunk.

A slice represents a full stack testable and demonstrable unit of work. Slicing and building chunks this way enables earlier sharing and feedback from others.

The output is an initial project board in GitHub, completing the Joy Map helps us get there. Once the GitHub board is ready we no longer update the slices in the Joy Map.

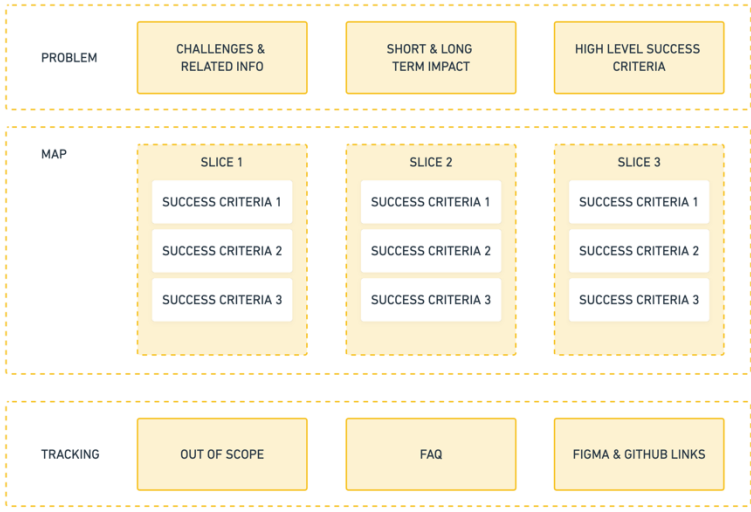


Figure 21 Elements of a Joy Map document in Notion

The process for Joy Mapping

Product creates an initial map of the slices. For each slice we describe the success criteria in the form of a list. For each item we describe the rationale. This process helps uncover gaps in the design. It also ensures that the chunk's challenges are completely solved.

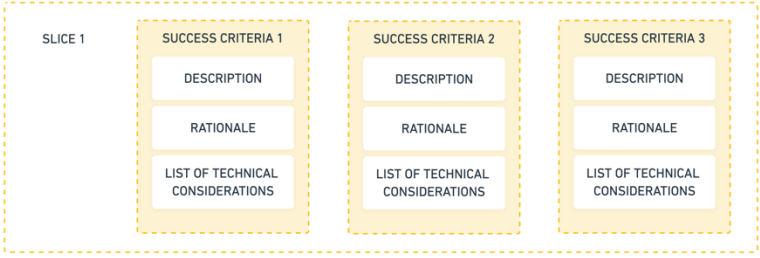


Figure 22 A slice containing success criteria

Engineering reviews the map asynchronously. Once product has created the initial map, they share it with the crew. They will

add comments, questions and any technical considerations. This promotes further asynchronous collaboration.

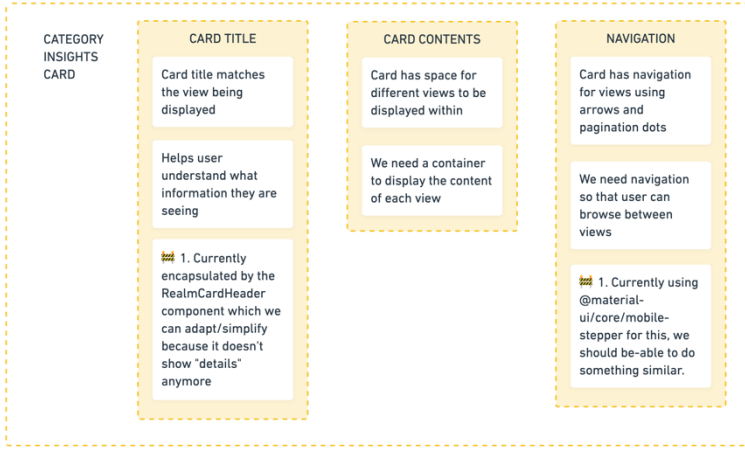


Figure 23 An initial slice with technical considerations

Product creates an initial GitHub project containing slices. Once the review finishes, product will create the initial slices in preparation for a kick-off meeting.

It is useful to have separate tickets for slices. This makes it easy for other stakeholders to see the broad progress while allowing engineers to focus on the details.

Product & Engineering complete the initial map in a kick-off meeting. There are several outcomes from this meeting:

- For the items which are clear, we create tickets labelled as 'initial work'. These factor in all the technical considerations.
- For the items which are not clear we create more tickets, with an additional label to 'investigate'. We prioritise these at the start. This ensures we surface the outcomes and prompt discussions as early as possible. We expect that we will create more tickets during build time from these.
- These tickets are linked to the related slice ticket using a number label. For example the first slice has a label ‘1’ and all the related tickets also have a label ‘1’.
- Each ticket contains a check list of the known tasks for it.
- We arrange the tickets in the order that makes the most sense. This will change as the work is in progress but gives us a good starting point.

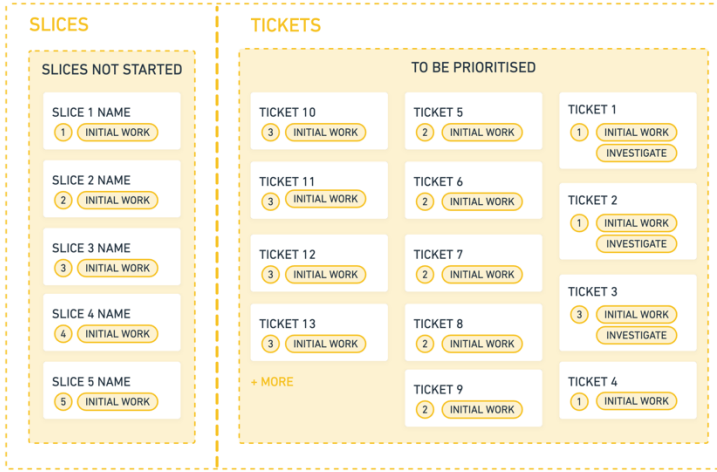


Figure 24 A template of a board with initial tickets

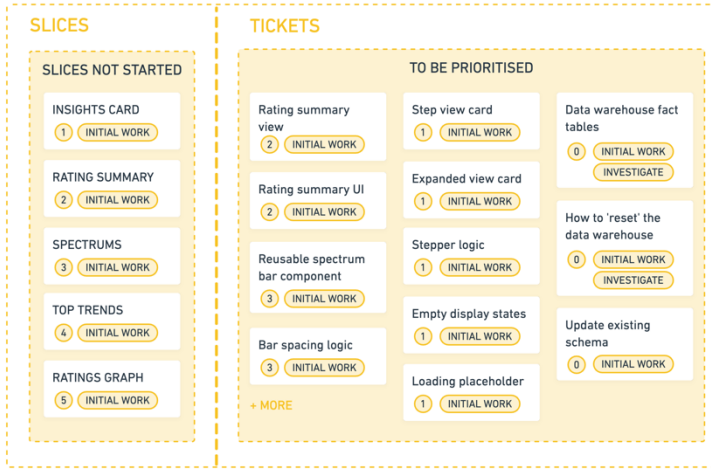


Figure 25 – Some of the initial work for the Insights chunk

The benefits of Joy Mapping

- **Revalidating the product approach.** The problem and solution is once more validated and described by a product person.
- **Upfront awareness of the unknowns.** The problem and solution is once more validated and annotated by an engineer. We highlight unknowns and prioritise investigating them at the beginning.
- **Clarity and alignment on the initial approach.** We start with a detailed description of problem and solution. From this crews can once again give asynchronous feedback and get more clarification.
- **Low effort to kick-off.** We don't need to think through everything up front. We expect there to be investigation and unknowns. We avoid a waterfall mindset.
- **Labels enable learnings.** As previously mentioned we label tickets as 'initial work', and if applicable 'investigate'. Tickets added later will not be labelled with 'initial work'. This means that we can clearly identify items that we missed initially for future

planning. It also makes it easy to identify why something took longer than we initially thought.

Group Architecture

As a scaling start-up we often hit hard engineering problems. Taking a collaborative approach to solving these problems has huge positive outcomes.

We have codified these outcomes as a set of technical and social requirements. We have also developed a process to ensure an architectural solution meets these requirements.

> “Group Architecture originated from an organic rhythm we got into when we were a small team.”

Technical requirements

- **Performant.** The solutions provides a good experience for users. Regardless of device, browser or network speed.
- **Scalable.** It will serve our known and imagined future scale and use cases. It is only scalable to what our use case feasibly requires. We don't add unnecessary complexity for cases that likely won't eventuate.
- **Simple.** We create as little infrastructure and technical complexity as possible. This means we don't need a lot of maintenance from engineering. This also ensures a low cognitive load on engineers.

Social requirements

- **Team alignment.** The team understands the architecture that they will be building. And, they feel a sense of shared ownership of the solution because they had a chance to contribute.
- **Architectural up-skilling.** The team up-skills in architecting solutions.

The Group Architecture process

Step 1: Describe the problem thoroughly without architectural bias in written form.

Step 2: Discuss the problem. Get the engineering team together and discuss the problem. Come to a common understanding of the problem. This may mean your description changes as new information and ideas come to light.

Step 3: Come up with alternatives. Encourage people to think about possible solutions to the problem asynchronously. Ensure there is enough time to plan possible approaches.

Step 4: Agree on an initial approach. Re-group and weight the options people have come up with. Deciding on a solution will often include a mix of approaches. Record each option, why it was not chosen. For the final approach, record why it won.

Step 5: Keep iterating the architecture. Have two engineers, one senior, iterate the architecture into a formal solution design. When the gist of the solution design is formed get together as a team once more.

Step 6: Re-validate the solution. Look for:

- Things that are missing.
- Things that won't quite work as expected.
- Areas for improvement.

Record all improvement vectors.

The small group continues to flesh out the solution design, incorporating the feedback.

Step 7: Rinse and repeat steps 5 and 6 until the architecture solution design finishes.

The hard bits here are when to share

Some guidelines are:

- When you solve a previous unknown.
- When you hit something in the solution that doesn't quite work out as intended and you make a change.
- When you fill out areas that restrict the options of the remainder of the solution.

Build

We've reached the point where a chunk of work is ready to build. Before we dive in it's important to understand our view on tech debt and quality.

Considerations for tech debt

To most, tech debt is limited to the result of prioritizing speedy delivery over perfect code. To us it’s anything engineering related that affects the team’s ability to deliver customer value.

Our definition of tech debt shapes how we deal with it. We have a more broad view of tech debt than most. We also have a specific approach to fixing tech debt.

If tech debt exists in the code we are actively developing, it will be fixed as part of development. Thus, at Joyous, fixing tech debt actually *becomes part of* feature development.

There are many cases that we don’t consider tech debt in this context. Here are two examples. The first is legacy code that meets near and medium term scale expectations. The second is code that follows old patterns. Particularly if there are no extensions on the near term roadmap for that part of the code.

Although, if it's near code that we are about to extend, then it may be tech debt. Why? Because it could pollute future patterns and standards.

The examples below help clarify what tech debt means to us.

- Code that is hard to build new features on top of.
- Code that won't scale to the features or user growth we are expecting in the next one to two years.
- Processes that slow down progress. For example, manual deployments, and a slow development environment.

- Lack of test coverage causing bugs to go unnoticed while developing.
- Complexity that we can mitigate in engineering. Including architecture or coding practices.
- Anything that extends the onboarding period of new engineers.
- Anything that increases the cost of future feature development.
- Engineers with cargo cult mentalities tied to technological ideology over results.

We have a few conditions that we aim to meet for new features:

1. All features that we build must be scalable and extendable for the work coming up in future.
2. No feature we build should increase our time to develop the next features.

> “Much like a bank loan, there may be times when taking on some debt makes sense at the time. That's okay as long as you are comfortable with the cost.”

Our approach to quality

We share ownership of quality across product and engineering. We don't have a team of QA analysts, instead we have just one. A senior analyst who helps champion quality across our entire team.

Different stakeholders incline towards some types of quality more than others. But we all view quality as a team effort, and we care about building a high quality product. We are happy to do anything to support each other in achieving this result.

The various types of quality that we focus on:

1. **Functionality.** It does what we expect, as defined by the success criteria.
2. **User Interface.** The visual design matches the designs created in Figma.
3. **Accessibility.** It conforms to Level AA of the Web Content Accessibility Guidelines (WCAG) 2.
4. **Usability.** Users know it's there, how to use it, and it's easy to use.
5. **Performance.** It provides a good experience for users. Regardless of device, browser or network speed.
6. **Code.** It conforms to current patterns, is simple and maintainable.
7. **Regression.** It doesn't break existing functionality.

The stages of building

Once a crew begins work on a chunk, they will start to have a daily stand up. The product person, quality person also attend. And our stand ups are open to anyone. It is common for our Head of Engineering to attend all stand ups.

When we begin, the crew starts with the tickets labelled 'investigate'. As investigations complete the crew and product person will get together again. They will decide how to proceed given the constraints, and what a good solution looks like. This is often done as part of stand-ups.

The product person or crew will then create a series of new tickets to reflect the decision.

The front of the GitHub board includes two columns for Slices. The first column is a holding area for slices that have not started. When the first ticket for a slice is picked up, then the related slice ticket will be shifted to Slices in progress.

Figure 26 - The other build columns on our boards

There are seven stages that tickets will pass through during build time. Each is represented as a column on our project board in GitHub.

1. **To be prioritised.** Tickets will sit in this holding area in a roughly prioritised order. The crew will talk about which tickets to start as they free up.

2. **Figuring it out.** We like this stage from Shape Up. During this stage engineers have picked up the ticket, and are still figuring out the best way to go about it.

3. **Getting it done.** We also like this stage from Shape Up. When engineers are getting it done, it means they have a clear path forward and are making good progress. The steps involved are usually logged as check-list task items on the ticket.

   An engineer will conduct thorough manual tests and write coded tests before shifting the ticket to the next stage. In some cases it might make more sense to test later, in which case they create new tickets for those activities.

4. **Code review.** Another engineer reviews the code before we deploy it to a dev environment. As part of the review they will ensure it follows current patterns. They will

also check for simplicity, and maintainability. Finally, they will do a sanity test to ensure the work functions as intended. They pass on any feedback to the coding engineer to address.

5. **Ready for testing.** Once a code review is complete a ticket will shift to ready for testing. This is a holding area that signals to our QA analyst that they can test the related work.

6. **Testing.** Our QA analyst will test the work against our dev environment. If an issue comes up during testing they add comments to the ticket. The ticket is then shifted back to *Getting it done*.

7. **Ready to deploy.** Once a ticket has completed testing it shifts to ready to deploy.

Testing entire chunks

Of course end to end testing still needs to occur. Particularly for a large project such as V3.0. Here we use a combination of manual testing as well as automated front end testing using Cypress.

For entire chunks we invite our whole organisation to take part. One of our product folks and our QA analyst will work together to co-ordinate a series of tests.

We then use a channel on Slack in which everyone can report issues or concerns. Everyone can comment on issues here.

Our QA analyst will reproduce and log valid issues as new tickets. We will add a label matching the type of issue, to help with future learnings.

Benefits of involving everyone in testing include:

- Organisation wide awareness of the coming changes.
- Improved outcomes thanks to broad feedback.
- Higher quality as a result of more testers.

Don't be afraid to reset

At any point in the build stages we can and do revive previous stages. While there has been a lot of thinking up to this point nothing is final.

> "When the rubber meets the road and we hit a pot hole we drive to the conditions."

If we see a potential improvement we will investigate it. We may change the design, or the pre-discussed solution. We agree the change between the relevant product and engineering folks. Then we feed the result back into build.

Ship

Shipping a large feature, or major release involves many important considerations.

Creating a release plan

Ahead of the release we create a technical release plan. This release is far bigger than usual. Joyous V3.0 comes with a major architectural re-write. This means we cannot release our chunks separately.

Here are the basic elements for the plan:

- **Eventually you have to set a hard release date.** We started with a two week release window, and then a month out, we announced the hard release date.
- **Determine the precise timing.** We are a cloud based SaaS product. So a release as major as this one requires some thinking. We chose a Saturday afternoon in NZT to deploy the release. This is the day with the least activity across the globe for Joyous. It also gives us a Sunday to roll back the release in a worst case scenario.
- **Carve out a window for end to end testing.** We were able to keep this window narrow by testing things as we finished building them.
- **Run a test deployment.** We test the deployment, so we understand the potential downtime. This also helps ensure the real deployment runs smoothly.
- **Document the plan.** We have a document that outlines our release steps, timing, and roll-back plan.

Communicating to key stakeholders

With a release as big as Joyous 3.0 we have been communicating the coming changes months in advance. We created a What's Coming page. The plan and other updates are included in our monthly product updates.

We also shared working prototypes with customers and users throughout the process. This helped us to validate and improve the outcome. It also helped us take many key stakeholders on the journey.

Future improvements may not be immediately available to everyone. Our V.30 changes are not hidden behind feature flags. This release will be live for everyone from day one.

As we got nearer to a final release, we updated our customers on the final release plan.

Messaging users

With a release this major we decided that it would be important to message users ahead of time. So, one week ahead of the release we send a notification inside of the tool. This is a short message that lets users know a change is coming, along with a link to what's coming. We have a different message for different types of users.

After the release we send another notification inside of the tool. This includes a link to showcase the changes we have made, and how they benefit the users. This message also encourages users to ask for help and send their feedback to us.

Our customers are large enterprise organisations. So we created release comms for them, rather than sending out mass emails on top of our internal app messages.

Help

We have a pretty comprehensive Help Center at Joyous. This includes detailed articles on how to use Joyous. It also includes a library of videos.

We plan ahead and ensure that our Help Center content is updated at the same time as the release. This falls under the responsibilities of our product folks, so we made sure to set aside the time and plan for it.

Product marketing

As we near the end of this release our product folks team up with our marketing folks. This release comes with some game changing functionality. It is a huge opportunity to market the items that differentiate us even more from the market.

Conclusion

We are not yet done. In fact we are quite sure we never will be. For now we intend to keep going - using Joyfully as our way of working.

We will continue to be open, experiment and refine our way of working as we grow.

If you try to work Joyfully (whether completely or in part) then we would love to hear from you.

If you want to join our team, well - then we would love to hear from you too 😉

You can find us at joyoushq.com

All the best,

Ruby, Kevin and the team at Joyous

References and thanks

Shape-up by Basecamp - not sure we would have gotten to this point without first trying Shape-Up and keeping a few concepts from it. Thanks Ryan!

Material.io by Google - for open sourcing their design approach and components so the rest of us don't have to keep reinventing the design wheel.

Figma.com - for making the most awesome design collaboration workspace out there.

Atomic Design by Brad Frost - for creating a simple methodology for interface design systems that our product and engineering folks could get aligned around.

Notion.so - for creating wiki software that made it ridiculously easy for us to design our processes and ways of working (and even write this book).

Play Contemporary Leadership CoLab - who taught Ruby the decision making framework we use in our mammoth meetings.

Thomas Otter - who mentioned the idea of a negative roadmap to Mike. Ruby thought he was crazy, googled it and found nothing. But turns out he was right. Having a negative roadmap is the single most important thing for a product organisation to have.

Lisa Cunningham - our Head of Engineering, who is on parental leave at the time we write this book, and has played a significant role in forming our partnership and ways of working.

Chris Holdaway - one of our Product Managers who helped create and define our design system, and general ways of working continuously throughout our journey.

Aylon Hebert - who contributed to the Joy Map structure and process.

Mike Carden & Phil Carden - our founders. For starting us off with the best objective ever, and trusting us to get things done. They continue to be fiercely supportive and contribute feedback and input often.