Developing a software engineering team structure at a SaaS company

Gaurav Bhorkar

Haaga-Helia University of Applied Sciences
Degree Programme in Business Technologies
Entrepreneurial Business Management
Master Thesis
2023

Abstract

Author(s)
Gaurav Bhorkar

Degree
Master of Business Administration

Report/thesis title
Developing a software engineering team structure at a SaaS company

Number of pages and appendix pages
50 + 1

When software grows, companies need to create multiple high-performing engineering teams to keep up with software development and enable fast delivery of value to the customers. Often teams are unable to perform due to how they are structured. This thesis is a case study of establishing an effective software engineering team structure at a software as a service (SaaS) company. The product is an intranet service used by customers for internal communications and knowledge management. The main goal of this thesis was to study the company's current problems and attempt to solve them with a new team structure that enables high performance.

The study incorporated a broad literature review of modern software engineering practices, team dynamics, organisation design, and organisational factors that affect team performance such as dependencies, cognitive load, clarity, psychological safety, size, and software architecture and communication structures. The methodology used incorporated group interviews as a primary data collection method. Other secondary data sources such as company documentation, HR system data, among others were also explored. The study utilised the 5-step design thinking process by empathising with the case, defining the main problems, ideating solutions to solve the problems, prototyping a new team structure, and proposing evaluation for the new structure.

The results of the study are structured as the design thinking process progressed. The empathize and define steps reveal the strategic areas, current structure, focus areas of current teams and their complexity, and dependencies. The problems to solve included low autonomy, low motivation, dependencies, too large focus areas, slow delivery, common code ownership, etc. The ideate and prototype steps focus on refining communication, collaboration, and dependency resolution by defining teams that are focused on different value streams.

The author proposes five stream aligned teams to the primary value streams of the SaaS product namely, pages (content), channels (content), discovery (search and notifications), governance (identity and administration), and growth. For a solid foundation, a platform team with two sub-teams is proposed, one focusing on the cloud platform while another focusing on the software platform. To execute the strategic area of artificial intelligence (AI), a separate AI team is proposed to provide services to other teams to leverage AI and reduce their complexity. Lastly, a quality engineering team focusing on enabling all teams to incorporate quality in their work is proposed. This proposed structure solves the main problems by focusing on high team autonomy, well-defined focus areas, lowering cognitive load, and reducing silos.

The thesis adds to the knowledge of organisation design by demonstrating a process of analysing and structuring software engineering teams thus addressing one of the major challenges in the ever-evolving landscape of modern software development.

Keywords
Teams, Organisation, Structure, Performance, Software, Communication

Table of contents

1 Introduction 1
1.1 Structure of the thesis 2
1.2 Case company and the goal of this study 2
2 Literature review 5
2.1 Modern software engineering 5
2.2 What is a team? 6
2.3 Organisational factors affecting team performance 7
2.3.1 Dependencies 7
2.3.2 Cognitive load 7
2.3.3 Structure and clarity 8
2.3.4 Psychological safety 8
2.3.5 Staffing and size 9
2.3.6 Software architecture and communication structures 9
2.4 Organisation design 9
2.4.1 Team-based organisation structure 10
2.4.2 Functional and cross-functional teams 11
2.4.3 Identifying value streams for teams 12
2.4.4 Resolving dependencies 13
2.4.5 Preventing silos 14
3 Methodology 16
3.1 Case study 16
3.2 Group interviews 17
3.3 Design thinking 19
3.4 Process of this study 19
4 Developing the engineering team structure 21
4.1 Empathizing with the team and defining problems 21
4.1.1 Strategic areas for product and engineering 21
4.1.2 Current team structure 22
4.1.3 Current focus areas and cognitive load 23
4.1.4 Current problems with team communications and dependencies 24
4.1.5 Summary of problems faced by teams 26
4.2 Ideating solutions for the new team structure 27
4.2.1 Capabilities needed to execute strategy 27
4.2.2 Identifying value streams in the product 28
4.2.3 Platform functions needed for developing and running the service 29

4.2.4 Other services 30
4.3 Prototyping the new team structure 30
4.3.1 Stream-aligned teams 31
4.3.2 Enabling teams 34
4.3.3 Complicated sub-system team 34
4.3.4 Platform teams 35
4.3.5 Communication, collaboration and resolving dependencies 36
4.3.6 Reducing silos 40
4.4 Testing the proposed team structure 41
5 Discussion and conclusion 43
5.1 Evaluation of the study 43
5.2 Significance of the study 45
5.3 Limitations 46
5.4 Concluding remarks 46
5.5 Future research 47
References 48
Appendices 51
Appendix 1. Semi-structured group interview structure 51
Appendix 2. Interview results 51

1 Introduction

Modern software is large and developed by multiple teams. Early software used to be simple, developed, packaged, and then delivered to customers. Today, the paradigm has shifted to providing software as a service (SaaS), wherein, the software company develops, deploys it in the cloud, and provides the software as a service to its customers over the Internet. This has reduced deployment and maintenance costs for customers (Waters, 2005).

SaaS companies typically uses a subscription-based business model, which ensures a steady pace of income for companies. SaaS has grown in popularity since the early 2010s. Between 2012 and 2018 subscription-based companies' revenues grew 5 times faster than S&P 500 companies ('The Subscription Economy Grows More Than 300% In The Last Seven Years', 2019). The SaaS model builds on the emergence of cloud computing where business do rapid development of products and services utilizing third party cloud services. For example, nowadays companies do not need to manage and maintain their own databases, they can use a database as service (DBaaS) from a third-party provider and configure it for their use-case. The emergence of cloud computing has also reduced the start-up cost of software development, primarily due to availability of cheap and simple to integrate foundational software service (e.g., databases, on demand computing, AI services, etc.) to build the value chain (Qian *et al.*, 2009). As a result, a SaaS product can be rapidly prototyped and iterated over to test the product market fit and does not need a big team to get started with software development.

Companies expand their software engineering teams to keep up with the growing need to develop more features, new products, to grow their business and keep up with competition. Typically, there is a point where the software is too big to be developed and maintained by a single large software team. The engineering performance reduces, typically introducing issues such as low quality, high technical debt, decreased motivation, and cognitive overload (Skelton and Pais, 2019). Therefore, when the time comes to split a large software engineering team into smaller teams, product and engineering leadership at companies often debates on how to split the teams, and quite often there are disagreements as there are many ways to structure software engineering teams.

The case company for this thesis is a SaaS provider that is at the same stage described above. This thesis aims to study literature for structuring teams and apply it to case company's situation. The end goal is to design an engineering team structure for the case company so that each of the resulting teams are autonomous and set up for high performance. The author uses the design thinking process with 5 steps – empathize, define, ideate, prototype and test, to understand the problems faced by the company and find a way to solve them with a new team structure. The

design thinking process is a general process of problem solving which can also be used to solve the problem of structuring teams.

1.1 Structure of the thesis

This thesis is structured as follows. In the rest of the introduction chapter, the case company and the goal of this study is described. The second chapter aims to build a theoretical framework for the study, with a literature review of software engineering practices, organisation design, and team performance to find the main factors that need to be considered when developing a new team structure. The third chapter covers the methodology for the case study research as well as describes the design thinking process in detail. The fourth chapter describes the implementation of the methodology and the results of the case study. Finally, the fifth chapter discusses the findings, limitations, and concluding remarks of the study as well as mentions the potential for future work.

1.2 Case company and the goal of this study

The case company has experienced growth in the last few years. The engineering team has grown to ~30 individual contributors and 4 managers and has shown signs of further growth with promising hiring plans. As the number of engineers has grown in the engineering department, the current organisation structure needs restructuring. The expectation is that, with the new organisation structure, the resulting engineering teams are autonomous and high performing. The company has gone through motions of merging and de-merging teams in the past 1 year. There are many debates between product, design, and engineering leadership on how the new structure should look like. So far, there have been many different proposals to structure the primary engineering teams, such as user Persona based teams, user role-based teams, product domain-based teams, a hybrid of persona and product domain-based teams, project-based teams, discipline-based teams, and other hybrid styles including but not limited to virtual teams, guilds, for doing engineering work.

It is obvious that because of no consensus within the leadership there has been inefficient decision making regarding the team structure. The author hopes due to the data driven and iterative nature of the design thinking process, the resulting team structure will be based on ground reality and scientific research, thus resulting is less disagreement.

The company develops intranet software and does business with a SaaS model. The main problem the software solves is of internal communications and knowledge management, that is, communication between leadership and employees and vice versa, communication between different departments and groups of individuals, and employee directory, etc. Primary features include ability to create an intranets website easily and speedily with pages, channels for dynamic communication, a people directory, content search, notifications, and possibility to integrate with other

company tools and services. The software is valuable for customers as shown by the company's growing customer base.

The company has also adopted modern practices of agile software development in the recent past. A few major milestones are as follows:

- Continuous Delivery (Fowler, 2013) – being able to deliver software to customers continuously.
- Trunk Based Development (*DevOps tech: Trunk-based development*, no date) – developing small changes and continuously integrating to the main development branch (often called a trunk).

The whole engineering organisation has seen some improvements due to the above practices. These include reduction in work in progress and improved focus, reduction in cycle type, and an increase in the rate of delivery. However, on the other hand there are other issues, such as, high amount of context switching, many projects and features have been left unfinished (and unreleased to customers), long running and delayed projects, a growing defect backlog, a growing amount of technical debt, among others.

As observed by the author, the product's software, over time, has followed the following Lehman's laws of software evolution (Lehman, 1996):

1. Continuing change – the software needs to be adapted continuously to maintain satisfaction.
2. Increasing complexity – the software has increased in complexity over time and needs work to maintain and reduce the complexity.
3. Self-regulation – the software evolution has self-regulated according to the dynamics of the organisation such as product, engineering, sales, finance, etc. decisions.
4. Invariant work rate – the effect of adding more people to a product area has not led to a major impact in increasing the speed of development over a long term.
5. Continuing growth – the functional content of the software needs to be continuously adapted according to the feedback of users and market requirements to stay relevant.
6. Declining quality – the system quality declined over time due to increasing complexity and a changing operational environment.
7. Feedback system – to evolve successfully the system needs multi-loop and multi-level feedback system which inherently trigger growth and change in the software.

The case company is clearly on the stage where there is a need for change in the current state of team structure to "tame" the increasing complexity, efficiently manage the continuing change and growth, and increase the quality level of the product, by increasing team performance. The main

aim of the thesis is to design a new engineering team structure for the case company, backed by scientific literature. This thesis will answer the following question within the context of the case company:

How to achieve a team structure that helps in increasing team performance?

2 Literature review

In this chapter, a theoretical framework is developed for the study. First modern software engineering practices are described to give the reader a background on how software is developed in the industry and the industry best practices. Then, the chapter defines a team within the context of software engineering as well as a team's stages of effectiveness to build context for the following section regarding team performance. Thereafter, the chapter explains the organisational factors that affect a team's performance, which are then used in the study's methodology to structure the new teams which aim to improve their performance.

2.1 Modern software engineering

Modern software, especially the one that is complex and is influenced by a real-world environment is typically collaboratively developed by multiple teams. More often, companies employ a variation of agile methodologies to develop their products. The manifesto for agile software development was declared by prominent industry experts in 2001 which stated the following principles (Beck *et al.*, 2001):

- Prioritising early and continuous delivery of software to satisfy the customer.
- Welcoming changing requirements even at later stages of development.
- Delivering software frequently, daily collaboration between developers and businesspeople.
- Keeping individuals motivated and trusting them.
- Considering face to face conversation as the best mode of delivering information
- Primary measure of progress is working software.
- Processes should promote sustainable development at a constant pace indefinitely.
- Continuously attending to technical excellence
- Focusing on simplicity – achieving the maximum with simple designs
- Self-organising teams will produce the best architectures, requirements, and designs.
- Regularly reflecting on how to become more effective.

There are many frameworks that implement the agile methodology, the most popular one being Scrum.

Developing software, unlike building a bridge is a rather complex process due to the nature of factors influencing it, predominantly changing requirements and a fast-changing technology landscape. This is according to Lehman's laws of software evolution that states that complexity of systems increases over time especially if the system mimics real world interactions (Lehman, 1996). As compared to building a complex bridge, building a complex software can be done in iterations as software, being non-physical, is easier to change (Farley, 2021).

Farley (2021) argues that for building modern software effectively, the stakeholders in the discipline of software engineering must focus on two core competencies:

- Learning: Learn about what the customers or users require from the software, or how to evolve a better solution for a problem, or how to apply the tools to solve a problem better, etc.
- Managing complexity: Manage how to not let the system grow complex at the technical as well as organisational level.

Farley (2021)'s arguments align well to counteract the adverse impact of Lehman's (1996) laws of software evolution. Learning is to gain insight from multi-loop and multi-level feedback and as the system grows more complex, the engineers must work towards simplifying it.

Learning is attained when teams work in an agile manner, delivering small increments of software, experimenting, and gaining feedback. Complexity is managed, when teams are continuously attending to technical excellence and applying best practices of software development such as modular development, separating concerns, high cohesion, and low coupling.

The teams that develop software must have the above competencies and should be enabled to follow agile principles of software development.

2.2 What is a team?

Most, if not all, software engineer job advertisements today mention teamwork as a required skill. The word "team" is used in variety of contexts. There are sport teams, sales teams, project teams, engineering teams, etc. A team is essentially a group of people working together to achieve a goal. Thompson (2008) defines a team as "a group of interdependent people with respect to information, knowledge, skills and resources, seeking to reach a common goal by combining their efforts". Team members need to have skills that complement each other to fulfil the team's jobs (Sage and Rouse, 2014).

For the purpose of this thesis, a team is defined within the context of agile software development. An agile team while confirming to Thompson's definition of a team, has additional abilities. An agile team is a group of people who have a common goal, trust, and respect each other, can intensely collaborate within and outside of the organisation, make fast decisions, and can deal with ambiguity (Cockburn and Highsmith, 2001).

A group of people, while may be called a team, may not be effective immediately when bunched together. A team goes through different stages to become effective. These typically include four stages as follows (Tuckman, 1965):

1. Forming – when the individuals start to get together and work out ways to solve the problems, albeit independently and may lack information. The team is not effective at this stage.
2. Storming – when the team members start to learn about each other, may have power struggles and disagreements. At this stage the team effectiveness is at the lowest.
3. Norming – when the team members resolve disagreements, accept each other, and start to make efforts to achieve the common goal. At this stage, team effectiveness starts to rise.
4. Performing – when the team members have full focus on the common goals, there is high competency, disagreements are resolved quickly. At this stage, team effectiveness is at its highest.

It is important to note that if a team reaches its performing stage, it may not remain at that stage. A team may be pushed to the storming stage, for example, due to changes in team structure, leadership, market, workload, etc. Therefore, teams should be structured in such a way so that it is easier for them to reach the performing stage.

2.3 Organisational factors affecting team performance

The key organisational factors that affect team performance as described in the following sections.

2.3.1 Dependencies

There are 3 categories of dependencies in agile software development – knowledge, process, and resource (Strode, 2016). These dependencies could be between teams or individuals. Knowledge dependencies include requirements, expertise, task allocation and historical knowledge. Process dependencies include activities or business process dependencies. Dependencies must be managed because they can block progress of work in a team (Strode, 2016).

For the context of this thesis, the study should focus on building a team structure that reduces dependencies between teams, for example, by breaking down the scope of each team in such a way that that there are less dependencies. As such, lowering the number of dependencies with other teams leads to autonomous teams as the progress of their work is not blocked by any other team.

2.3.2 Cognitive load

A team mental model is a shared understanding and a mental representation of key knowledge between all the individuals in the team (Klimoski and Mohammed, 1994). Agile development practices lead to a shared mental model which in turn enables team members to give more feedback and help each other (Schmidt *et al.*, 2014). However, human brain capacity is limited. A high cognitive

load affects team performance because teams are spread thin trying to address multiple domains and responsibilities (Skelton and Pais, 2019).

Therefore, for an effective engineering team organisation, each team must have only enough cognitive load that they can cope with.

2.3.3 Structure and clarity

Internal research at Google revealed that structure and clarity is one of the properties of effective teams (Tamiru, 2023). The research suggests that teams should have clear roles, goals, and plans so that the team members know the expectations, the aims and how to achieve them.

However, too many goals are difficult to achieve in the face of resource and time constraints which results in people juggling multiple goals, completing one task while neglecting others (Dalton and Spiller, 2012).

Therefore, for an effective engineering team organisation, each team should be able to have structure and clarify in their goals, role, plans, and should not have too many goals to work on simultaneously.

2.3.4 Psychological safety

Psychological safety allows team members to be comfortable of failure and thus allow risk taking (Tamiru, 2023). In psychologically safe teams, individuals feel accepted and respected. To foster learning and managing complexity, teams must strive for psychological safety.

To create psychological safety, organisations must accomplish shared expectations and meaning for work, foster confidence that voicing concerns are welcome, and pivot towards continuous learning (Edmondson, 2018). The leadership toolkit developed by Edmondson (2018, p. 159) recommends the following for leaders to build psychological safety in their organisations:

- Set the stage: Frame the work by setting expectations about failure, uncertainty and clarify the need for voice as they are interdependent. Emphasize the purpose by finding the things that are at stake, reasons why they matter, and for whom.
- Invite participation: Acknowledge the gaps thus to demonstrate humility in the situation. Ask good questions as well as listen intensively to inquire. Finally set up structures and processes, for example creating forums for input and guidelines for discussion. Thus, enabling learning and knowledge sharing.

- Respond productively: Express appreciation for raised concerns and destigmatise failure. When violations happen, for example when concerns are not voiced when required, flag them clearly and sanction them.

2.3.5 Staffing and size

The agile methodology advocates for teams to be self-managing or self-organising (Cockburn and Highsmith, 2001). To prevent dependencies, for example on specialists outside of the team or depending on another specialist team (e.g., a database team), it is important for teams to be autonomous (Stray, Moe and Hoda, 2018). Therefore, teams must be cross-functional, meaning, the team must be staffed with people whose collective skillsets enable the team to be autonomous. From a general software development perspective, the most common specialisations needed are frontend, backend, and database expertise. These skills could be gained from specialists or generalists.

The number of people in a software engineering team should be 5-8 to build high trust which then enables the team to experiment and innovate (Skelton and Pais, 2019).

From an organisational perspective, teams should be “full-stack”, that is cross-functional and about 5 to 8 individuals.

2.3.6 Software architecture and communication structures

There are different ways to design software architecture. Generally, software is broken down into different components (or modules) which may interact with each other to provide functionality to the user. The benefits of a modular architecture results in being able to develop modules separately with less needs for communication between modules (Parnas, 1972). If teams are working on a module together, there is a high need for constant coordination between teams to reduce the instability created by affecting each other’s work (Herbsleb and Grinter, 1999).

The impact of good design is lost if it results in temporal dependencies between teams as interfaces between components, functionality of the system, and commitments are continuously renegotiated to encounter instability (Herbsleb and Grinter, 1999).

Therefore, teams should work towards architecting the software and organising themselves such that it reduces the need for intense cross-team collaboration.

2.4 Organisation design

Organisation design is about making choices regarding how a company is set up, including its structures, methods, and systems that form the organisation (Nadler, Tushman and Nadler, 1997).

In this thesis, this definition is applied to the product and engineering departments of the case company.

Nadler and Tushman recommend both a top-down and a bottom-up approach to organisation design. The top-down approach considers designing to implement the strategy while the bottom-up approach considers designing to craft work processes, jobs, workflows, measures, and other operational aspects.

Kates and Galbraith (2010) note that the organisation strategy must influence the organisation design. A popular framework in organisation is the star model from Kates and Galbraith, states that organisations should use strategic direction to find the set of capabilities needed to achieve the strategic goals. Then the structure, people, rewards, and processes must be designed to build these capabilities. Kates and Galbraith note to achieve the organisation design, leaders must focus on reducing complexity and increasing alignment. Firstly, complexity is reduced by creating simple and easy to understand structure. Secondly, alignment must be needed between each of the structure, people, rewards, and processes to support the strategy.

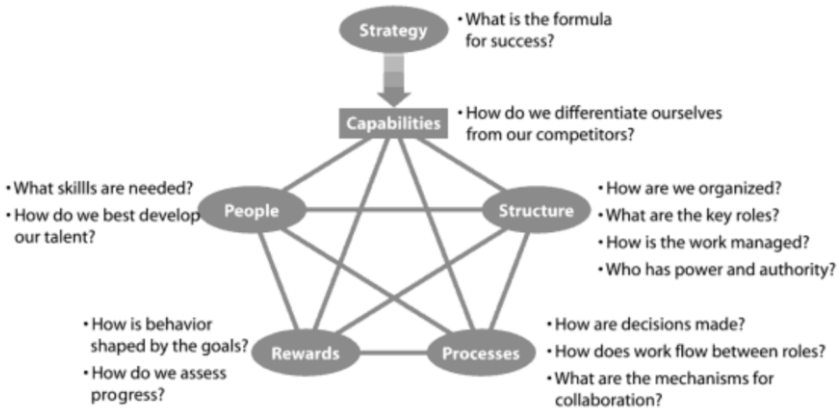

Figure: Star Model (Kates and Galbraith, 2010)

2.4.1 Team-based organisation structure

Harris and Beyerlein (2003) note that in a team-based organisation, a team is considered as a building block of the organisation where teams have accountability. The authors argue that teams can be organised around product, process, or customers. A team-based organisation also needs support systems such as goal setting, performance management, feedback, learning, and leadership that promotes flexibility, adjustment, and self-management.

In the book Team Topologies, Skelton and Pais (2019) make the following observations which can be considered when building a new team structure in this study:

1. Small, diverse, long-lived teams that works towards a goal.
2. There is a constant stream of work that flows to each team.
3. Team members have a team-first mindset, where they put the needs of the team before their own.
4. Teams have good boundaries which lowers their cognitive load.
5. Teams have restricted responsibilities, for example by limiting the types and number of domains that the team is responsible for.
6. Interactions between teams are facilitated to improve trust, awareness, and learning.

2.4.2 Functional and cross-functional teams

Disadvantages of functional teams

In software development, many organisations create teams based on specialisation, for example, frontend, backend, database, operations, etc. Such teams, although highly specialised with knowledge, have dependencies on each other. Tune and Millett, (2017) mention the following negative impacts of functional teams:

1. Teams have bottlenecks. For example, the frontend team will depend on the backend team, which will depend on the database team to deliver their pieces.
2. Bottlenecks lead to lack of business agility even though teams may seem agile. A functional team of specialists may have agile routines such as continuous integration and continuous delivery, but when considered as a whole team of teams, value delivery to the customer is very slow. A frontend team may deliver a simple change to the UI, but to make it work, the backend team may take months because the database team need to update the database configuration, thus increasing the cycle time for the overall end to end value.
3. There is lack of ownership due to low autonomy and dependencies. Due to lack of ownership a blame culture may develop. People in a functional team tend to identify with their own type of experts than they identify with the main purpose of the organisation (Gray and Vander Wal, 2014).
4. Overall accountability is low because of lack of autonomy, ultimately making the user experience suffer.

For this study, it is safe to say that functional teams should not be considered because they have high number of dependencies.

Cross-functional teams

Cross functional teams are teams that have individuals with different specialisations, so that, the team has the full skillset needed to deliver value end-to-end. Cross functional teams have a higher

autonomy in terms of working closely with the customers to continuously discover their needs, as well as in terms of technology due to no dependencies between different technical functions (Tune and Millett, 2017). This means that a cross-functional team can learn and manage complexity continuously, the two main competencies that are needed to develop modern software as seen in section 2.1.

In the book Team Topologies, Skelton and Pais (2019) define the following types of teams:

1. *Stream-aligned team*: This type of team is responsible for a specific stream of work, ensuring that it is delivered efficiently from end to end.
2. *Complicated sub-system team*: These teams are experts in a complex or specialized area and provide a service to other teams for specific, complicated tasks.
3. *Enabling team*: Enabling teams focus on improving the tools, practices, and processes used by other teams to help them deliver more effectively.
4. *Platform team*: Platform teams create and maintain the foundational infrastructure and services that other teams rely on, such as infrastructure, tooling, and foundational frameworks.

Most of the teams in the engineering organisation should ideally be stream-aligned teams as these teams are directly interfaced with the customers provide the end value to them.

Gray and Vander Wal (2014) mention a similar concept for organising which they call *pods and platforms*. A pod is defined as a cross-functional autonomous team that is focused on a specific area of solving a customer problem, is always learning, and experiment fast. Pods also have boundaries for increased cohesion and autonomy, but they collaborate with other pods and the organisation to allow for control and consistency. A platform supports the pods to increase their effectiveness and allow for networking. A platform may prescribe standards, governance, or a common set of services for other pods.

As the concepts are very similar, this thesis considers the types of teams explained by Skelton and Pais (2019) as the basis for the study.

2.4.3 Identifying value streams for teams

Both Skelton and Pais, (2019) and Tune and Millett, 2017) advocate for business domains derived bounded contexts to be owned by teams (in this case stream-aligned teams).

The concept of Domain Driven Design (DDD) can be used to identify bounded contexts. Domain driven design is a concept introduced by Eric Evans whereby software and its internals are modelled after a deeply understood domain, thereby both the software and the domain following the same ubiquitous language (Evans, 2004).

A bounded context is a subdivision of a domain. Within a bounded context the terminology (or the language) is consistent, while outside of the context the same aspects may have a different meaning (Fowler, 2014). For example, a user in the context of telephone call is a caller and a receiver but the same user in the context of billing is a subscriber of services. Therefore, in a telephone call a user is simply used to identify who is calling whom, whereas, in a billing system the same user, called a subscriber, is charged bills, has addresses, credit card information, etc.

Tune and Millett (2017) explain multiple methods of identifying bounded contexts namely – jobs to be done framework, domain storytelling, domain use case diagrams, context maps, using language heuristics, etc. Explaining each of the methods is out of scope for this document.

Once domains and bounded contexts have been identified, they can be assigned to teams. Domains can be classified as simple, complicated, and complex based on their cognitive load. Therefore, care should be taken not to overload a team with multiple complex domains (Skelton and Pais, 2019).

2.4.4 Resolving dependencies

Teams driven by business domains will likely have dependencies with one another as naturally business domains have dependencies too. For example, after an order for books is placed (ordering domain), they must be shipped soon (shipping domain). Fortunately, having teams based on business domains also translate into software boundaries using domain driven design due to the application of Conway's law (Conway, 1968).

Dependencies can be resolved using communication, collaboration, and contracts. Skelton and Pais, (2019) define the following three interaction modes for teams:

Collaboration is when two teams work together closely on a shared goal, such as exploring new technologies or approaches. This mode is useful when a high degree of adaptability or discovery is needed.

X-as-a-Service is when one team provides something to another team as a service, such as an API, a tool, or a full software product. This mode is useful when there is a clear separation of concerns between the two teams, and when the consuming team does not need to be involved in the development or maintenance of the service, thus freeing up the consuming team to focus on their own work, thereby reducing the coupling between teams.

Facilitation is when one team helps another team to learn or adopt a new approach. For example, an enabling team helping a stream-aligned team gain expertise, thus reducing the amount of time and effort needed.

Skelton and Pais (2019) describe that the interaction modes have an impact on the how work is done by teams. While collaboration may allow fast discovery of work, it can create shared responsibilities and in turn dependency between teams. X-as a service can be difficult set up because interfaces need to be agreed and built in the software. To enable facilitation people with right expertise are needed. The advantages and disadvantages of the interaction modes are described in detail in Table 1.

Table 1. Team interaction mode advantages and disadvantages (Skelton and Pais, 2019)

Interaction mode	**Advantages**	**Disadvantages**
Collaboration	Rapid discovery of new things	Can slow things down, creates wide, shared responsibility for each team, with increased context and higher cognitive load.
X-as-a-Service	Increased speed of delivery, reduced coupling between teams	Can be difficult to set up and maintain, can lead to silos.
Facilitation	Reduced time and effort for stream-aligned teams, improved quality of work	Can be difficult to find enablers with the right expertise, can create a dependency on the enabling team.

2.4.5 Preventing silos

As teams get autonomous, they can form silos if they are disconnected from the rest of the organisation.

Tune and Millett (2017) strongly recommend conveying business context at all levels and repeat it often. A strong understanding of the business context and goals will help employees make strategic choices. The authors give an example where “without clear sense of purpose at an organisation engineers will likely focus on shiny new technologies, while product manager will simply copy others”. They recommend the use of cascading objectives where leadership defines the strategic ambitions objectives, executives, teams, and individuals will create objectives to align with that. An example of this in practice is the Objectives and Key Results (OKR) framework.

Learning can be promoted by reserving time for it or having a space for learning, for example, a team member doing an apprenticeship with another team (Gray and Vander Wal, 2014).

Another way to promote learning and spread knowledge is the use of guilds or communities of practice. A guild is an informal group of people to unite individuals in the company from who have

shared interests, leisure or engineering related (Smite *et al.*, 2019). Guilds are open to anyone who is interested. Smite et al. (2019) studied guilds at Spotify and recommend that guilds should have a mission, scope and expected value on the individual and organisational level.

A guild can help bridge the communication gap between not just software engineering team but between different parts and individuals of the organisation.

3 Methodology

This thesis is a case study with an aim to develop a team structure for the case company. The methodology used is a case study to understand the case company's situation with the current team structure and the problems associated with it. This information is then used to create a new team structure that addresses the current problems. The design thinking methodology is used as an inspiration for the executing the process of this case study. The design thinking process is a problem-solving process and can be used to solve the problem of structuring teams.

This chapter introduces the concept of a case study, data collection using group interviews, and the design thinking methodology. The chapter concludes with a detailed description of the process of this study.

3.1 Case study

A case study is an in-depth empirical study of a phenomenon within its real-world setting (Yin, 2014). It is important to note that a case study is done in a real word setting and not as a control environment experiment. In this study, the phenomenon is a company that develops software as a service.

Runeson et al. (2012) in their book, define a case study in software engineering, as an in-depth empirical study of one or a small number of instances of a software engineering phenomenon within its real-world context, by drawing information from multiple sources of evidence. The software engineering phenomenon studied here is the collaborative development of software by multiple teams, in a real-world context such as the one done in the case company. In book also defines the case study research process as a 5-step process – the design step where objectives are defined and the case study is planned, the data collection preparation step to define procedures and protocols to collect data, the evidence collection step where data is collected for evidence, the analysis step to analyse the collected evidence, and the reporting step where conclusions are reported. In this thesis, the design, data collection preparation steps are part of this chapter on methodology, while the data collection, analysis, and reporting steps are roughly part of Chapter 4 on implementation of the study.

Lethbridge, Sim and Singer, (2005) note that, to collect data for the study there are various kinds of data sources that can be leveraged for case studies in the software engineering context, primarily classified into first degree, second degree and third degree. In the first-degree category, there is an active involvement of the participant population (such as engineers) to collect the data. Examples of first-degree data collection include interviews, questionnaires, brainstorming, etc. In the second-

degree category, there is an indirect involvement of the participant population by having the researcher access their work environment. Examples of second-degree data collection includes observing participants' tools, metrics, logs, workplace, etc. Finally, in the third-degree category, the research studies work artifacts such as documentation, code, etc. This study has selected first-degree and third-degree data collection methodologies, mainly via group interviews and by analysing documentation. The third-degree category's data is not easily available to the researcher, and therefore is not considered for the study. The concept of group interviews as a data collection methodology for this study is explained in the section 3.2.

After the data is collected, the next step is to analyse it. Yin, (2009) describes the following techniques:

- Pattern matching to find patterns in the data collected within the case or with multiple cases.
- Explanation building to seek underlying explanations based on cause-effect relationships.
- Time series analysis to find out what happens in a case over a period.
- Logic model analysis to be carried based on logical models to match empirically observed sequence of events to theoretically predicted events. This analysis is similar to pattern matching.
- Cross case synthesis to compare different cases.

In this thesis, explanation building is primarily selected as the analysis technique. With the data collected from group interviews, an explanation is build based on cause-and-effect relations, and thus, resulting in a set of problems faced by the case company and their root causes.

3.2 Group interviews

Software development is predominantly a social activity with a high degree of collaboration in a team setting. Therefore, a group interview is an excellent choice to collect data for qualitative analysis within a social context. Moreover, group interviews are an efficient use of resources as more individuals can be interviewed simultaneously.

Frey and Fontana (1991) note in their paper that group interviews generally employ two strategies – observation, where the researcher observes the group interactions to collect data; and interviews, where the researcher actively asks questions based on observations or theory. A group interview may be used by researchers for many purposes summarized by the authors as follows:

- Exploratory – To explore, gather understanding, familiarize, etc.
- Pretest – To test a hypothesis, a do a pilot test, try ideas, etc.

- Triangulation – To validate using multiple methods, especially since a group setting provides a larger number of subjects.
- Phenomenological – To determine meanings beyond individual interviews and gain more insights. For example, by subtle pitting individuals against each other in the interview, the researcher can gain insights on group opinions, agreements, conflicts etc.

Frey and Fontana, (1991) mention the most significant dimensions of the group interview as – role of the interviewer, structure of questions, the purpose of the interview, and the setting of the interview. The interviewer's role and question structure, which are focussed more when planning the interviews, are explained below in more detail:

- The role of the interviewer: The role can be passive where the interviewer becomes a passive observer and only prompts to reinforce the discussion. A passive role is suitable for exploratory and phenomenological purposes. A passive role is typically used in an informal setting. Another role of an interviewer is active where the interviewer directs the interview flow, is an empathetic participant, and administers a structure and order to the interview and keeps the group on track. An active role is more suitable for formal settings.
- Question structure: For exploratory and phenomenological purposes, an open-ended question structure is generally suitable to permit more flexibility in responses and probes. For pretest purposes, a structured questionnaire is more suitable.

The main goal of the group interview in this thesis is to collect data and insights that are used to develop the new structure using the process describe in the next section. The main insights collected are as follows:

1. Problems faced by engineers, managers, and designers and how they relate to the current team structure.
2. Find out the domains of the product from the perspective of engineers, managers, and designers.
3. Find out the team structure desired by each team member to see if there is a common pattern of structuring teams.

As deduced from the above goal of the interview, the group interview's purposes are mainly exploratory, triangulation, and phenomenological. Therefore, the question structure chosen is quite open-ended and semi-structured, thus enabling flexibility in responses and allowing further probing. Although some questions are close ended, they are followed up with a quick "why?" thus enabling continuous discussion. The guiding questions for semi structured group interviews are listed in appendix 1. The role of the interviewer is chosen to be primarily active and then switching to passive in parts of the interview. The interviewer directs the structure of the interview actively, however

when the participants start discussing, the interviewer switches to passive mode to become an empathetic participant and only prompts when necessary. In the interview, participants collaboratively add notes on the digital whiteboard to answer the guiding questions, they facilitator prompts the group to clarify each note to identify the root causes of problems using the 5 why method (Pojasek, 2000). The setting is a formal, hybrid meeting, where some participants connect remotely, while some are in an office meeting room.

3.3 Design thinking

Design thinking is a human-centred approach to problem solving that focuses on understanding the needs of users and designing solutions that are desirable and feasible (Interaction Design Foundation - IxDF, 2016). The process is iterative.

The design thinking process is typically divided into five steps:

1. Empathize: Understand the needs and experiences of the users. This can be done through user research methods such as interviews, surveys, and observation.
2. Define: Once there is a good understanding of the users' experiences and needs, the problem that needs to be solved is defined in a clear, concise, and actionable manner.
3. Ideate: In this step, ideas for solutions to the problem are brainstormed.
4. Prototype: In this step, prototypes of the promising ideas are created so that they can be tested.
5. Test: The final step is to test the prototypes with users to get their feedback. This feedback can be used to further refine the prototypes.

Designing an organisation structure is a problem that can be solved with the design thinking process. Due to lack of time and to limit the scope of the study, only one iteration of the process is performed, that is, only one team structure is proposed.

3.4 Process of this study

Designing engineering teams for the case company is essentially a problem. To solve this problem this study will roughly follow the design thinking process.

In the empathize step, the author empathizes with different stakeholders as well as company documentation to understand the current situation. First step is to understand the strategic product and engineering areas. Thereafter group discussions are conducted, with the thesis author as an active-passive listener to direct the interview and take notes to understand the current problems faced by product and engineering stakeholders; current team structure; and the areas of focus handled by each team, classified by their complexity.

The group discussions are started with a set of guiding questions as described in appendix 1. The group discussion meetings are structured in the following manner:

1. Participants are shown the guiding questions on a digital whiteboard and are asked to write notes on the board to answer the questions.
2. The responses trigger discussions amongst the participants. The facilitator prompts the participants to understand the reasons behind the problems noted on the board. The method used in the 5 whys method to dig deeper into identifying root causes (Pojasek, 2000).
3. The participants are then prompted to write the list of domains and classify them as simple, complicated, and complex based on cognitive load. This is done to gather an understanding of the current domains in the product and their complexity.
4. The participants are prompted to distribute the domains into multiple teams individually and write the reasoning behind it. This is done to understand how participants view boundaries between domains and teams and to see if there emerges a pattern.

In the define step, the data collected from the group discussions as well as the company documentation is analysed to identify the following:

- To define the strategic product and engineering areas and the capabilities needed to fulfil the strategy.
- To define the problems that need to be solved with the new team structure.

In the ideate and prototype steps, there is a brainstorming of multiple ways to structure the teams that will likely solve the problems that were defined in the previous steps and the knowledge obtained from the literature review. Thereafter a potential team structure is derived and documented.

The testing step checks if the problems will be sufficiently solved by the new structure.

4 Developing the engineering team structure

This chapter describes the results from process of the study. The chapter is roughly divided into the design thinking steps – empathize, define, ideate, prototype and test.

First, the empathize and design steps results are described, namely identifying the strategic direction, current team structure, focus areas of development and their cognitive load, and the problems faced by current teams. This information is used to in the ideation step where ideas to fix the problems are described. The ideas are then used to create a team structure in the prototype step based on the knowledge from literature review. Finally, the chapter ends with proposal of ways of testing the team structure.

4.1 Empathizing with the team and defining problems

The empathize and define steps attempt to get an understanding of the current state of the teams, the current structure, and define the main problems faced by the stakeholders. These two steps primarily focus on gathering data for analysis using group interviews and studying documentation.

Group interviews were conducted for five groups comprised out of 19 participants. The composition of groups was as follows – two groups of engineers (4 participants each), one group of senior engineers (5 participants), one group of product managers and designers (4 participants), and one group of engineering managers (2 participants). The raw data gathered from the group interviews is in the form of digital boards and is described in appendix 2.

4.1.1 Strategic areas for product and engineering

Based on the star model from Kates and Galbraith (2010), the strategic direction of the company should be identified which can then be used to discuss the capabilities required to achieve the strategy. The author studied the documentation on the company intranet, discussions, and meetings with leadership in the product and engineering departments to understand the product strategy. Based on the strategy, the strategic areas were derived as follows. Note that only the product areas that are affected by the strategy are listed here instead of the actual strategy as it is considered confidential.

1. Product strategic areas
    a. Artificial intelligence
    b. Best content creation experience in the industry
    c. Analytics
    d. Content discovery
    e. Content engagement

f. Growing the user base
g. Security, Privacy, and governance
h. Reducing the time to gain value after buying the service

2. Engineering strategic areas
    a. Developer Experience
    b. Operational efficiency – monitoring, alerting, disaster recovery
    c. Cloud platform improvements

4.1.2 Current team structure

As noted in the methodology chapter, the author used the third-degree data sources to collect data for the case study, which includes company documentation (Lethbridge, Sim and Singer, 2005). The author analysed the HR system and documentation present on the internal company intranet pages to identify the current organisation structure. There are 4 engineering managers, 25 engineers, 3 designers, and 3 product managers.

There are following 4 teams developing the product currently namely Core, Mobile, Growth, and Platform. The current teams and their responsibilities are described in Table 2.

Table 2. Current teams and responsibilities

Team	**Team members**	**Team responsibilities**
Core	10 engineers, 1 quality engineer, 1 designer, 1 engineering manager, 1 product manager	Most of the features
Mobile	3 engineers, 1 designer, 1 engineering and product manager	Rewrite the new mobile app and maintain old mobile app
Growth	5 software engineers, 1 quality engineer, 1 designer, 1 engineering manager, 1 product manager	Driving user growth to the service, new pages functionality, and generally major new functionality that may drive growth in any product area
Platform	5 engineers, 1 engineering manager	Setting up and maintaining the underlying Cloud Platform, Developer Experience, Data Platform

It is also observed that there are two additional virtual teams namely LCM, and AI, that comprise of team members that are borrowed from other teams as well as permanent team members. The additional virtual teams are described in Table 3.

Table 3. Virtual teams and responsibilities

Team	Team members	Team responsibilities
LCM	2 engineers (borrowed from core team), 1 designer (permanent), 1 engineering manager (borrowed from Growth team), 1 product manager (borrowed from Growth team)	Temporary project team to fix a broken functionality. Some members are balancing time between their home team and this team.
AI	2 engineers (1 borrowed from Growth team, 1 borrowed from Platform team)	To build a platform for artificial intelligence. Members working full time.

4.1.3 Current focus areas and cognitive load

To find the cognitive load on each team, each group in the interviews was asked to document the areas of product development handled by their teams and classify the domains into simple, complicated, and complex (see appendix 1 for the interview questionnaire). The classification of domains is based on the recommendation by Skelton and Pais, (2019) which defines simple, complicated, and complex domains. A simple domain has work that has a clear path of action, a complicated domain has work that needs analysis and iterations, and a complex domain has work that needs experimentation and discovery (Skelton and Pais, 2019). The areas of development can be specific product functionality, a product domain, or a technology domain. The focus areas and their complexity that was deduced from the group interviews are described in Table 4.

Table 4. Current Focus areas and complexity

Focus Area	Type	Current Team	Complexity
Mobile App	Multiple Product Features	Mobile	Complicated
Pages (old & new versions)	Product Feature	Growth	Complex
Login	Product Feature	?	Simple
Main Web App	Multiple Product Features	?	Complex
People Directory	Product Feature	Core	Simple
Lifecycle Management	Product Feature	Core	Simple
Console and Support Tooling	Multiple Product Features	Core	Simple
App Marketplace	Product Feature	Core	Complicated
Custom Apps	Product Feature	Core	Simple
Search	Product Feature	Core	Complicated
Analytics	Product Feature	Core	Complicated
Google & Microsoft Integrations	Multiple Product Features	Core	Complicated
Page Widgets	Product Feature	Core	Simple

Notifications	Product Feature	Core	Simple
Admin	Multiple Product Features	Core	Complicated
Backend Core Library	Supporting technical artifact	?	Simple
Data Replication Services	Supporting technical artifact	?	Complicated
User and Group Management	Product Feature	Core	Simple
User and Group Provisioning	Product Feature	Core	Simple
Channels	Product Feature	Core	Complex
Cloud Platform	Technical Domain	Platform	Complex
Data Platform	Technical Domain	Platform	Complex
Developer Experience	Technical Domain	Platform	Simple
Email Notifications	Product Feature	Core	Simple
Page Templates	Product Feature	Growth	Simple
Bookmarks	Product Feature	Core	Simple
Launcher	Product Feature	Core	Simple
New Design System	Supporting technical artifact	?	Simple
UI Kit	Supporting technical artifact	?	Simple
Branding Styles and Typography	Product Feature	?	Simple
Navigation and Routing	Technical Domain	?	Simple
Custom Domains and Login	Product Feature	Core	Simple
Sign Up	Product Feature	Growth	Simple
Trial and Demo	Product Feature	Growth	Simple
AI Platform	Technical Domain	AI	Complex
API Gateway	Technical Domain	?	Simple
Third party API	Product Feature	?	Simple
Billing System	Product Feature	Growth	Simple
Roles & Permissions	Product Feature	?	Simple

As seen in the above table there are many areas for which there is no primary team responsible for maintenance (marked with a question mark).

4.1.4 Current problems with team communications and dependencies

To identify the current problems the group interview data was analysed for building explanations as mentioned by Yin, (2009) as well as by other data sources such as architecture documentation.
The current teams communicate on an ad hoc basis.

The core team being the biggest team has most of the communications internal. When there are dependencies during work, they are resolved on an ad-hoc basis, that is, by directly reaching out to the engineer who is the likely expert on the topic or their manager, thus breaking their flow state and leading to high cycle times and prioritisation conflicts. Sometimes teams form virtual teams to collaborate on features that span more than one team.

There are many dependencies between product teams. The most prominent ones are as follows:

1. Authentication and Authorization (Login and Permissions) – the product needs to validate whether a user is logged in or not before accessing any feature. Currently login code is shared by all teams and is changed on a need basis. However, the number of changes needed are not that high.
2. Mobile App – the mobile app is developed by the single mobile team. The team is a frontend team and does not have any backend developers, which means new feature in mobile app is dependent on that feature's backend being implemented by another team (mostly the core team).
3. The main web app is the foundation container for all the features. The features essentially depend on the main web app. This app does not have any team responsible, and teams collectively update the foundation code whenever needed (e.g., security updates).
4. The admin app and support tooling app are also similar to the main web app where it is a foundation container for all feature administration settings. This app also does not have any team responsible.
5. In the future many teams will depend on the AI platform to implement AI into product features.
6. Pages functionality is developed by both core and growth teams which leads to occasional conflicts with code and need meetings to resolve. Some page functionality is also used in channels which creates additional dependencies between core and growth teams.
7. The data replication service does not have any team responsible, and teams share the codebase and update it on a need-basis. The service is used to move data between different backend services and is very foundational. Every backend service which needs to receive or transfer data from another service depends on this service. Changes to this service are generally low, such as bug fixes, and minor improvements. However, has a high learning curve due to being developed in-house.
8. Core libraries are used by almost every service. These libraries contain common shared code. Changes to this library are often needed, to fix bugs, or add new common code, etc. This shared code does not have any responsible team and is updated on a need-basis.
9. Design System – these reusable technical components are used in almost all front-end code, thus creating a dependency with all teams. There is no team responsible for

maintenance and code is updated by teams on a need-basis. However, changes done to this component has a chance of cascading to all dependent codes thereby risking stability.

10. Notifications and Email is used by other product features thus creating a dependency between core and other teams. There is currently no well-defined interface between notifications and other areas thus needing collaboration every time changes are needed.

4.1.5 Summary of problems faced by teams

The problems and their root causes are identified based on the data collected in the group interviews of engineers, product managers, designers, and engineering managers. In this part as well the group interview data collected was analysed for building explanations as described by Yin, (2009). The problems and their root causes are described in Table 5.

Table 5. Problems and their root causes

Problems	**Root causes**
Lot of common ownership of code (therefore creating dependencies between teams)	Over reliance on reusability of software, creating large dependencies in software as well as between teams
Team is always on learning mode, experts in nothing, cannot keep track of what is going on Too long backlog for the team There is an encouragement to know everything Long meetings Too much work in progress Lack of focus on monitoring and observability	High cognitive load. Too large teams and too many domains Undefined focus areas for teams
Cannot deliver value fully (slow delivery, only partly, wastage of time) Lack of focus on product discovery, teams are rather output driven instead of outcome driven	Due to lack of expertise in all the areas that the team work on which is a result of high cognitive load
Internal tools and common services are rigidly enforced (e.g., in one case a service written in Node.js was rewritten in Java just because Java is the preferred language)	Lack of team autonomy, top-down workflow.
Autonomy, Mastery, Purpose is low or at medium level. Unclear which bug goes to what team	Unclear objectives and focus areas for teams
Mobile team cannot do backend work and needs to ask another team	Dependencies between teams due to teams based on function (i.e., mobile team has only mobile developers)

4.2 Ideating solutions for the new team structure

In this step, the author ideated various approaches to engineering team structures.

Teams based on each function such as frontend team, backend team, quality team, were rejected due to the creation of multiple dependencies between the teams. For example, frontend team depends on backend team for functionality, thus preventing delivery of full value in a certain part of the product by a single team. As ascertained in the literature review, cross functional teams are preferred.

The ideation step proceeded to first identify capabilities needed to execute the strategy. Thereafter, it was followed with identifying the value streams in the product as well as supporting services needed to develop, maintain, and operate the product and its value streams. Since there are more value streams than teams, some teams have more than one value streams. Value streams that are related to each are added to the same team to reduce cognitive load. If a team already has a very high cognitive load value stream, then no more streams are added to that team. When ideating teams, it was ensured that the capabilities needed to execute the strategy are developed.

4.2.1 Capabilities needed to execute strategy

As mentioned by Kates and Galbraith (2010) in the star method for designing organisations, the capabilities needed to execute the strategy are identified. The leadership team discussed together and found the capabilities needed for executing the strategy. The ways to achieve these categories are ideated based on the author's experience, suggestions by participants of the focus group interviews, as well as discussions with colleagues done during coffee breaks at the office. The ways to fulfil the capabilities also consider the mitigation for the root causes of the problems identified in the empathize and define steps, thus following the design thinking process. The capabilities and ways to fulfil them are explained in Table 6.

Table 6. Capabilities needed and ways to fulfil them.

Capability	**Ways to fulfil**
Artificial intelligence skills Ability to enable teams to build AI into features easily using a simple AI platform.	A separate team focussed on building an AI platform staffed with specialists. Training all employees on how to use AI and enabling them
Ability to execute strategic focus areas in parallel and deliver value fast (important areas – content management, discovery, growth, and governance).	Autonomous teams that can discover problems and implement solutions as independently as possible. Having clear focus areas, objectives, and code ownership for teams

	Reducing cognitive load on each team
Ability to enable teams to build their features on top of an easy-to-use cloud platform. Ability to monitor, alert, and recover to achieve operational efficiency	A team focused on platform engineering and reliability. Enabling all teams for operational efficiency with right training and establishing processes.
Ability to delivery software with high quality, security, and legally compliant manner	A team to support and enable quality engineering in all teams. Providing training and establishing quality processes.
Create a better developer experience by reducing the feedback loop for engineers when writing code.	Providing reusable artifacts and constant feedback and monitoring of factors that affect developer experience. A separate team may be helpful if the budget allows it.

4.2.2 Identifying value streams in the product

To find value streams in the product, product areas were clustered together by their similarity in terminology. As noted in the literature review (section 2.4.3), the value streams were identified by primarily considering each domain's bounded context as recommended by Skelton and Pais (2019). Each value stream was identified such that it can be independently developed and can have clearly defined dependencies between them. Following are the streams identified:

Content: An intranet's core value lies in creating content efficiently and consuming it efficiently. In the product there are two ways to create content, pages functionality for static content, and channels functionality for dynamic content. This is a core value stream and is expected to have many jobs to be done for the customers and is a strategic area.

Searching: Searching provides value to users by being able to find and discover the content in an intranet.

Notifications: Notifications provides value to users by being able to discover right content that is relevant to them and bring them back to the service, thus increasing engagement and consumption of the content.

People Directory: Customers of the intranet are companies, and their employees need to find each other. An intranet's value also lies in being able to find relevant people to facilitate collaboration.

Analytics: Analytics provide valuable insights about content creation and consumption to the users of the intranet so that they can make changes accordingly to boost creation and consumption.

Extensibility: Customers, especially in the enterprise segment have special requirements that are not met by the current product. However, providing a product that is extensible paves way for customers to be able to extract more value out of the service by customizing the product to their own needs. This includes features such as custom applications, app marketplace, etc.

Setup and Governance: When the intranet service is sold, it needs to be configured to the right parameters that suit the customers. Customers find it very valuable if the time to set and govern the service is lowest. This includes identity management, billing accounting, auditing, etc. There is a constant stream of requirements primarily coming from enterprise customers regarding administration.

Growth: Growth is a special value stream that does not deal directly with features, but rather with the growth of the product's user base. This includes user acquisition, activation (converting to paying users), and resurrection.

Support tooling, security, mobile app, etc. are not considered value streams because they do not deal with a part of the product but rather is an amalgamation of product features. For example, the mobile app and the website provide similar services, just that the mobile app is technologically different.

4.2.3 Platform functions needed for developing and running the service

The service needs to be run on top of a platform. There are a lot of common aspects that are reused by each value stream (or a group of value streams). These include the following areas which have a constant stream of work.

Cloud Platform: Cloud platform is the underlying services provided by the cloud service provider. These include, databases, compute platform to run services, etc. This requires special technical expertise to be able to setup, govern, and secure.

Software Platform: There is a software platform underlying the product. This includes data replication between services, shared libraries for services (e.g., database libraries, common way of writing code, etc.). The software platform also includes the facade of the application, that is, the container of all features – the mobile app container, the web app container, the admin app container, etc.). Note that the actual features that are contained inside the mobile app, web app or the admin app are not considered as platform components.

Artificial Intelligence Platform: Artificial intelligence (AI) is a strategic area and there is going to be a need for each value stream to evaluate the usage of AI to enhance the value proposition of

the value stream and the whole product. AI is a specialized technology and needs specialist skills in developing AI capabilities and making them usable via a well-defined interface. This value stream is expected to see a lot of requirements in the near and long term.

4.2.4 Other services

There are other cross-team services needed to keep the product functioning. These include as follows. These services apply to the whole product, similar to platform functions.

Developer Experience: A set of guidelines and expert consultations provided to keep the experience of developing the product high. This may include identifying and fixing bottlenecks in delivery each team's goals by the user of surveys, common instructions, or even shared code.

Quality Engineering: Expert consultations to improve the overall quality of the product. Such services enable each team to build quality in the engineering process itself. This also may include providing reusable artifacts to each team to reduce their cognitive load such that incorporating quality becomes easier.

Security Engineering: Expert consultations, best practices, and guidelines, to improve each team's overall product security posture.

Data Engineering: Experts in practices to enable analysis of data for discovering insights for growth.

4.3 Prototyping the new team structure

In this step, the author prototyped a team structure based on the value streams described in the ideate section. The team structure derived was as a result of discussions with leadership, patterns seen in the focus group interviews, and the knowledge gained from the literature review. Note that the prototyping and the ideation sections were worked simultaneously and influenced each other. To keep the study length within time limit, only one team structure was derived.

According to the team topologies, the new team structure involves stream-aligned teams, enabling team, complicated sub-system team, and platform teams (Skelton and Pais, 2019).

Each team in the new structure is presented with characteristics such as team name, value stream, objectives, goals, key metrics, etc. Some characteristics such as key metrics, goals, and objectives are expected to be refined and revaluated by each team.

4.3.1 Stream-aligned teams

There are five stream aligned teams considered – Pages, Channels, Discovery, and Governance, and Growth. The stream aligned teams are aligned with the value streams (Skelton and Pais, 2019).

The extensibility value stream was broken down and combined with the respective value stream that is intended to be extended. For example, extending content value stream, search and notifications, identity, etc. Since the team owning a value stream already has the knowledge, adding extensibility to that value stream will only be a minor additional effort as compared to a specialized team dedicated to extensibility which needs to gain knowledge of all the value streams.

Similarly, the analytics stream was also broken down and combined with respective value streams that needs analytics functionality. For example, content analytics must be owned by content teams while search analytics must be owned by the team responsible for search.

The mobile app must also be developed by all the value stream teams. Essentially, value streams within the mobile app should be owned by respective teams similar to the web app. The foundations of the mobile app were moved to the platform team that also owns the web app foundations.

Pages – Since Pages is a major part of the product, it needs a separate team to ensure a sharp focus. The characteristics of the pages team are described thoroughly in Table 7.

Table 7. Characteristics of the pages team

Team Name	Team Pages
Type	Stream aligned team
Objective	To provide an unparalleled static content creation and consumption experience
Value Streams	Content (Channels)
Features	Pages editing and viewing experience, page widgets, page widgets extensions, page lifecycle, page engagement analytics, etc.
Preliminary metrics	Number of active page editors per day/month Number of active page consumers per day/month
Members	1 Engineering Manager 5 Engineers (3 frontend, 2 backend) 1 Designer 1 Product Manager

Channels – Channels being another major part of the product needs a dedicated team also. The characteristics of the channels team are described thoroughly in Table 8.

Table 8. Characteristics of the channels team

Team Name	Team Channels
Type	Stream aligned team
Objective	To provide an unparalleled dynamic communications experience
Value Streams	Content (Channels)
Features	Channel management, posts creation and consumption, articles, channel integrations, channel extensibility, channel engagement analytics, etc.
Preliminary metrics	Number of posts created per day/month Number of comments created per day/month
Members	1 Engineering Manager 4 Engineers (2 frontend, 2 backend) 1 Designer 1 Product Manager

Discovery – The discovery team is a combination of search, notifications, and people directory streams as they primarily relate to discovery of either content or other people using the product. It could even be considered that discovery is a separate value stream. The characteristics of the discovery team are described thoroughly in Table 9.

Table 9. Characteristics of the discovery team

Team Name	Team Discovery
Type	Stream aligned team
Objective	To enable the fastest discovery of content, people, and tools in an organisation
Value Streams	Search, Notifications, People Directory
Features	Searching content, notifications (mobile, in app, desktop, email, etc.), search analytics, notifications analytics, engagement analytics.
Preliminary metrics	Search metrics – success rate, completeness, average result ranking, time to successful search Notifications metrics – relevance (ignored vs clicked), opt out rate, etc.
Members	1 Engineering Manager 4 Engineers (2 frontend, 2 backend) 1 Designer 1 Product Manager

Governance – The governance team has a broad scope of governance features. Although most of the governance features have a low complexity thus the cognitive load on the team will not be very high. The characteristics of the governance team are described thoroughly in Table 10.

Table 10. Characteristics of the governance team

Team Name	Team Governance
Type	Stream aligned team
Value Streams	Setup, Governance
Objective	To enable customers to set up the product fast and govern it easily
Features	Identity and access management, user provisioning, permissions and access control, custom domain, custom login, custom branding, custom emojis, security configuration, accounting, billing, reporting, primary integrations, etc.
Preliminary metrics	Average time to set up Admin NPS
Members	1 Engineering Manager 4 Engineers (1 frontend-full-stack, 2 backend) 1 Product Manager Designers can be borrowed from other teams on demand

Growth – The growth team will focus on the growth aspects of the product. This includes doing research on how to increase the user base of the product, acquire more users, activate users (make them paying users). The team will work towards user sign up, free trial experience. The characteristics of the growth team are described thoroughly in Table 11.

Table 11. Characteristics of the growth team

Team Name	Team Growth
Type	Stream aligned team
Value Streams	Growth
Objective	Propel the product expansion in terms of users, revenue, and market presence
Features	Trial experience, Onboarding users, Sign Up, Product led acquisition or upsell, user activation, etc.
Preliminary metrics	Number of new trials Churn rate Conversion rate A/B testing results, etc.
Members	1 Engineering Manager 2 Engineers 1 Product Manager Designers can be borrowed from other teams on demand (for example per project)

4.3.2 Enabling teams

Enabling teams are the ones that enable other teams to deliver efficiently.

Quality engineering – Quality has been given a high strategic importance by the company leadership. Have a separate enabling team for quality engineering enables a centralization of quality initiatives, processes, and technology. An enabling team's main work is to enable other teams to deliver fast. In this case the quality engineering team will enable other teams in the company to implement quality practices and processes. Quality engineers may temporarily become part of respective teams for a certain period to aid in quality implementation and transferring the quality engineering knowledge to each team. Once the teams are enabled, the quality engineering enabling team should function as continuous improver of quality initiatives across the organisation. The characteristics of the quality team are described thoroughly in Table 12.

Table 12. Characteristics of the quality team

Team Name	Team Quality
Type	Enabling team
Value Stream	Quality Engineering
Objective	To enable each team to deliver value to customers with highest quality
Features	This team will not own any features but help other teams to gain knowledge of how to deliver high quality in the features they develop.
Preliminary metrics	Overall test coverage, Release stability, Test runtime, etc.
Members	1 Head of quality (may also have additional responsibilities) 2 Quality Engineers

4.3.3 Complicated sub-system team

Complicated sub-system teams are the ones that are responsible for a part of the system that is complicated enough to deserve its own team. Thus, complicated sub-system teams reduce the cognitive load on other teams, thus enabling them to delivery more efficiently.

AI Platform – The AI Platform team is considered a complicated sub-system team because of the specialist skillsets needed for building an AI platform. This team is very autonomous given the complexity of artificial intelligence and the need for extensive experimentation to achieve a competitive lead. This team is staffed by old time senior engineers who can work autonomously. In the future this team is recommended to be expanded to a separate team out of platform team with an engineering manager to lead it. The characteristics of the AI team are described thoroughly in Table 13.

Table 13. Characteristics of the AI team

Team name	Team AI
Type	Complicated sub-system team
Objective	To build and easy to use APIs for AI enablement
Work areas	Machine learning models, vector databases, prompt engineering, large language models, etc.
Preliminary metrics	Response time, throughput, model performance metrics, etc.
Members	2 very Senior Engineers with AI expertise

4.3.4 Platform teams

The platform teams' main objective is to create a foundation such that it enables all the stream stream-aligned teams to deliver value to the customers easily and as fast as possible. Essentially, the platform teams reduce the cognitive load of stream-aligned teams so that they do not need to work on setting up the foundations. In the case company's context, three different streams within the platform team are needed.

All the streams within a platform team are recommended to be headed by one senior executive who has extensive experience in managing and setting up an engineering platform.

Cloud Platform – Developing a software as a service needs a good cloud platform for services such as compute, security, storage, content-delivery network, etc. The case company uses a popular cloud platform provider which needs specialists to manage and maintain the cloud infrastructure. The characteristics of the cloud sub-team within the platform team are described in Table 14.

Table 14. Characteristics of the cloud platform team

Team name	Team Cloud
Type	Platform Team
Objective	To build and easy to use and scalable cloud services platform for building the service
Work areas	Infrastructure provisioning and management, identity governance, security and compliance, developing a scalable architecture for high performance, platform tooling, monitoring and observability, container images, etc. Data engineering (to be considered splitting into an enabling team or a sub-team for focusing on the data platform, if budget is available to hire more data engineers)
Preliminary metrics	Uptime, Response times, resource utilization, running costs, etc.
Members (sub-team)	1 Head of Platform (shared with other platform sub-teams) 2 Platform Engineers

	1 Data Engineer 1 Product Manager (shared with other platform sub-teams)

Software Platform – The software platform sub-team is responsible to reducing the software development cognitive load on other teams by establishing common software foundations. The team must strive to keep the foundation simple, and low cost. The characteristics of the software platform sub-team within the platform team are described in Table 15.

Table 15. Characteristics of the software platform team

Team name	Team Software Foundations
Type	Platform Team
Objective	To build software foundations that enable fast development of features and excellent developer experience
Work areas	Design foundations Mobile app foundations Web app foundations Analytics foundations Backend and microservices foundations (for example, data replication, common libraries, etc.) Developer experience is considered an essential part
Preliminary metrics	Satisfaction from other teams, time to integrate, etc.
Members (sub-team)	1 Head of Platform (shared with other platform sub-teams) 4 Engineers (focused on frontend as well as backend) 1 Designer (focused on common design foundations, by collaborating with designers of other teams) 1 Product Manager (shared with other platform sub-teams)

4.3.5 Communication, collaboration and resolving dependencies

To resolve dependencies and foster communication and collaboration, team interaction modes of X-as-a-service, Collaboration, and Facilitation were chosen from the book Team Topologies by Skelton and Pais (2019). The book recommends choosing X-as-a-service as the interaction mode where there are clear boundaries in the bounded-contexts developed by teams. In cases where more discovery is needed for work that spans multiple teams, the Collaboration modes is chosen. In cases where one team needs to enable other team with knowledge or ways of working, the facilitation mode is chosen.

The following paragraphs explain the interaction modes for each team:

Platform Teams provide platform services to other teams (X-as-a-service)

Platform teams must publish a list of services they provide and the ways to access these services for other teams. Few examples include:

- Providing a way how to deploy services, for example, using deployment tools and enforcing a standard for deployment.
- Providing a standard way to create cloud infrastructure, for example, enforcing a standard for infrastructure as code.
- Software platform exposes interfaces regarding how to use the mobile app, web app foundations. Thus, it enforces architectural styles such as micro frontends to enable autonomous development for stream aligned teams.
- Software platform provides interfaces and ways for replicating data between services, so that teams do not need to collaboratively build data exchange mechanisms, but rather can just use the interface and mechanisms provided by the software platform.
- Provides services for observing, alerting, and monitoring of deployed software services for other teams.

Platform teams facilitate to enhance developer experience across

Platform team members can temporarily become part of other teams for a certain period to enable them to enhance their development experience. For example, helping each team set up monitoring and alerting, improving build run times, enhance security posture, etc. The goal with facilitation is not to lend more resources, but rather help gain knowledge, thus enabling the other teams themselves to be more capable, thus eventually reducing the dependency on platform teams.

All teams primarily use X-as-a-service interaction mode when possible

Since the stream aligned teams have their domains more aligned with bounded contexts, it will be now easier to provide services to each other if a dependency arises. With a well-defined interface, teams do not need to collaborate to build new features, but rather focus on their responsibilities of using the interfaces. That is, the provider of the interface focuses on providing the functionality while the consumer focus on using the functionality, where the interface is agreed beforehand. Thus, having an interface between teams decouples them. As seen from Conway's law, such interface between teams will eventually manifest itself as an interface between the software components produced the teams (Conway, 1968). Following are some examples of such services which teams can provide to each other:

- AI platform can provide an interface to interact with the AI services for all other teams.

- Discovery team can provide an interface to send notifications. Thus, if a page is created and a notification needs to be sent to a user, the pages-service (developed by the pages team) can simply use the notifications interface to deliver the notification to the notification service (developed by the discovery team). Thus, the teams do not need to collaborate because of the presence of a well-defined interface. The pages team can now focus on pages functionality, while the discovery team can focus on delivery of notifications, not just from pages but from any other team's services.
- Pages team and channels team can provide an interface to expose content so that it can be retrieved by the search service for indexing data for search (developed by the discovery team).
- Governance team can provide an interface to access identity information, so that any features other teams are building that need identity information can use this interface.
- Governance team can also provide an interface for allowing teams to add more administration configuration. This way, if a team is building a feature which needs some administration configuration, they can just use the interface provided to build the administration configuration themselves instead of depending on the Governance team to build it.
- Governance team can provide an interface for getting branding information for a customer. Thus, other teams can use branding artifacts using the interface. The governance team in this case can focus on building a feature for setting up and governing the branding artifacts, while other teams can only focus on using the branding for their features.

The above list does not document all the interactions due to the vast and complex nature of the product. However, it gives the reader a gist of how the X-as-a-service interaction mode can be used by teams to effectively reduce dependencies between them.

Another case where X-as-a-service is useful is for enabling code contributions from one team to other team's code. Each team should document their rules of accepting code contributions (similar to how open-source projects do). This document essentially becomes an interface between teams. It is then possible for one team to write code themselves even if the concerned area of the system is owned by another team as long as they are able to follow the other team's document for contributing code. This will reduce waiting times.

Using facilitation for enablement

The primary mode for the quality engineering team should be facilitation. This is because the main goal of the quality engineering team is to enable other teams to be better at delivering their work with high quality. In such case facilitation could be moving a quality engineering team member to a concerned team for enablement. For example, a quality engineer can help impart knowledge on

what quality is, help build test plans, help build a testing framework, help create test cases, etc. After the target team is sufficiently enabled, the quality engineer can continue as a consultant, and move their focus to another team's enablement.

The facilitation mode is not just for enabling teams. It could be used by other teams also. For example, if a team develops an interface and there is a need to increase adoption of that interface, the team developing that interface could facilitate the adoption for each team. A clear example of such strategy can be seen in security interfaces such as migrating to a token-based authentication from a cookie based one. While a cookie-based authentication system may work, but moving to a token-based system is more secure, and thus would need each team to adopt it. However, since each team may lack knowledge about implementation, the team that implement the token-based system (likely governance team), can facilitate the implementation by lending their team member to other teams to help transfer knowledge and increase adoption.

Teams use collaboration when required

The main reason to not recommend collaboration as an interaction mode is because it blurs boundaries between teams, may result in more coupled teams, and increase cognitive load on teams (Skelton and Pais, 2019). However, collaboration is suitable when there is a need for fast discovery and experimentation (Skelton and Pais, 2019).

In the case company's scenario, the AI team, which is a complicated-subsystem team building the AI Platform, can use collaboration with other stream-aligned teams to incorporate AI into their features.

The platform team can also use the collaboration mode with a stream-aligned team when building a new platform team. For example, if the current platform lacks capabilities for developing an event driven system, the software platform sub-team can work together with the stream-aligned team that needs such a system to build one together.

After the intended results are achieved upon collaboration, the interaction mode can switch to X-as-a-service. The same approach can be used for example for adding more features to the shared core library, where a team could write code for extending the core library and collaborate with the software platform sub-team to review and incorporate it. Collaboration is recommended to be only when needed.

4.3.6 Reducing silos

While separate teams allow for the parallel development of multiple focus areas of the product, it may also create teams that are so autonomous that they do not have anything common amongst them, thus creating silos. The case company is recommended three ways to reduce silos – by creating alignment across all teams and leadership, enabling movement of people, and establishing guilds.

Creating alignment

It is recommended to cascade company objectives at all levels of the company, for example via the objective and key results (OKR) methodology (Tune and Millett, 2017). The objectives and key results decided at the company level should be cascaded to each team. Each team must create a set of their own objects and determine key results to achieve those objectives. This will create alignment. Team leaders, namely engineering manager and the product manager must communicate the vision and objectives very clearly and periodically and ensure that the work that the team choses does is relevant to the objectives. This way, all teams will be working towards the same company vision and objectives.

To keep track of progress of achieving objectives and align with each other, it is recommended to create a periodic alignment meeting for product and engineering leaders (all team leads and executive leadership). The meeting should discuss the current progress of OKRs and ways to mitigate any threats that may derail progress.

At a technical level, to create alignment between teams, it is recommended to create a periodic alignment meeting between technical leads of each team (for example, all lead engineers). In this meeting the leads should discuss the technical capabilities of the end-to-end system including architecture, technical processes, developer experience, deployment processes, etc. This will help share knowledge and establish common technical ground for all teams. However, care must be taken to ensure autonomy for each team so that teams can achieve a fast flow of delivering value.

Team apprenticeships

It should be possible for people from one team to expand their knowledge and areas of expertise. One way is to enable movement of people to become part of another team for a certain period and increase their knowledge, similar to an apprenticeship (Gray and Vander Wal, 2014). If a team member is more motivated to move to a certain team after an apprenticeship, it should be allowed as motivated individuals will lead to better team performance (Skelton and Pais, 2019).

However, it should also be noted that while movement of people is okay, it should be not very frequent as it disrupts the teams. Teams should be long-lived (Skelton and Pais, 2019). Therefore, managers should balance team stability and movement of members between teams.

Guilds

Informal groups of people with shared interest, also known as guilds, help with knowledge sharing across the organisation (Smite *et al.*, 2019). People sharing knowledge without team boundaries inevitably reduce silos. It is recommended to the case company to enable creation of the following guilds:

- A backend guild to enable knowledge sharing between individuals who are interested in working on backend technologies, databases, microservices architectures, etc.
- A frontend guild to enable knowledge sharing between individuals who are interested in working on frontend technologies, user experience, frontend frameworks, etc.

A guild is voluntary and should not have any engineering responsibilities and should be used to predominantly share knowledge. The group should agree on a mission and responsibilities of individuals.

While the above two guilds are suggested, individuals should be free to form new guilds and grow them. This is expected to foster a culture of knowledge sharing and learning continuously.

4.4 Testing the proposed team structure

In this step, the prototype of team structure is tested. As testing the structure takes a long time, it is considered outside the scope of this thesis. However, this section suggests ways to test the team structure for feasibility.

The prototype team structure can be tested by gathering feedback from all the team members. Qualitative feedback can be obtained by comparing the current team structure and the proposed team structure and evaluating if they would solve the problems faced due to the current team structure. More detailed explanation of problems solved is presented in the conclusion chapter from an evaluation perspective.

The new team structure must be tested for financial feasibility. The new team structure needs 6 Engineering Managers, 30 Engineers, 4 Designers and 6 Product Managers. This means the case company needs to recruit 2 new Engineering Managers, 5 new Engineers, 1 new Designer, and 3 new Product Managers. This financial undertaking may disrupt the budget and profitability of the case company. If the budget does not allow for hiring a full set, then the structure must be revisited

and evaluated again. For example, it may be feasible for a single senior engineering manager and a single product manager to handle both pages and channels teams, or the platform team to be able to function for a short period without a product manager. However, the impact must be assessed carefully.

Finally, it is recommended to evaluate how the new team structure will have an impact on software architecture. As Conway (1968) suggested that an organisation is likely to produce software with a structure that mimics its communication structures, it is important to note that autonomous teams will likely result in software components that have boundaries and interfaces that are a copy of the interfaces between the teams developing them. In the proposed team structure, a likely result would be that the software will have 4 major components – for identity and governance, pages, channels, and discovery and engagement developed by respective teams. There will likely be a need to implement technologies that promote development of interfaces between these components, namely, message and event passing systems, event registry, application programming interfaces, etc.

5 Discussion and conclusion

The main results of the study, which is the proposed team structure and the communication structure, are already described in the prototyping step (section 4.3). In this chapter the author reflects on the study. First the main goal of the study is recapped and the results from the study are evaluated, and the significance of the study is discussed. Then, the study's limitations are described. Finally, the chapter ends by drawing concluding remarks.

5.1 Evaluation of the study

The main goal of this study was to answer the following question for the case company:
How to achieve a team structure that helps in increasing team performance?

The new team structure involves 5 stream-aligned teams (pages, channels, discovery, governance, and growth) to work on independent value streams, 2 platform teams (cloud, and software) to build the foundations, 1 enabling team (quality engineering) to enable other teams to deliver with high quality, and 1 complicated sub-system team (artificial intelligence) to reduce the cognitive load on other teams and give focus to an important strategic area. The new team structure is developed to mitigate the main organisational factors that affect team performance which are described in section 2.3 as – dependencies, cognitive load, structure and clarity, psychological safety, staffing and size, software architecture and communications. By mitigating each of these factors, the resulting teams should be set up to achieve high performance due to the following reasons:

- The structure reduces dependencies between assigning teams focus areas that have a clear bounded context as recommended by Tune and Millett (2017). Lowering the dependencies will lead to less blocking of work and waiting times for teams (Strode, 2016).
- Smaller focus areas reduce the cognitive load on teams as the newer teams will not be spread thin trying to work on multiple areas (Skelton and Pais, 2019). With a lower cognitive load, it will become easier to create a shared mental model for team members which will allow team members to give more feedback and help each other (Schmidt *et al.*, 2014).
- Smaller focus areas also bring structure and clarity to the teams by focusing on singular goals which leads to clear plans to achieve them. A small number of goals will bring clarity, reduce resource and time constraints, and reduce juggling of multiple tasks (Dalton and Spiller, 2012)
- While team structure is not the only factor to improve psychological safety, with smaller team with focused objectives and goals, it will be easier to bring meaning to work. By following agile practices and reducing silos by employing apprenticeships to move people

between teams, establishing guilds, it will foster continuous learning. This partly fulfils Edmondson's (2018, p. 159) leadership toolkit thus helping to establish psychological safety.

- Smaller teams with 4-7 members in each team will help build high trust and enable teams to experiment and innovate (Skelton and Pais, 2019).
- When the new team structure comes into effect, the boundaries between teams will likely not correspond some boundaries between software components. However, over time, by prioritising the acceleration the effect of Conway's (1968) law, the software should be modified such that their boundaries between different domains or components should correspond to the interfaces between teams. This will evolve the software architecture with components that are more autonomous and can evolve independently, thus allowing the teams developing them to deliver value fast and independently.

The specific problems identified during the empathize and define steps are described along with how the new team structure solves them in Table 16.

Table 16. How the new team structure solves problems?

Problems	**How the new structure solves them?**
Lot of common ownership of code (therefore creating dependencies between teams)	Different teams would own their part of the code. There will be common code ownership initially, however teams are likely separate ownership of the code because of the application of Conway's law where the new communication structure should evolve the code structure (Conway, 1968).
Team is always on learning mode, experts in nothing, cannot keep track of what is going on. Too long backlog for the team There is an encouragement to know everything. Long meetings Too much work in progress Lack of focus on monitoring and observability Cannot deliver value fully (slow delivery, only partly, wastage of time) Autonomy, Mastery, Purpose is low or at medium level. Unclear which bug goes to what team. Internal tools and common services are rigidly enforced (e.g., in one case a service written in Node.js was rewritten in Java just because Java is the preferred language)	Each team in the proposed structure has a smaller focus area than the current teams. Thus, cognitive load is lowered. This should shorten the backlog for each team, reduce meeting times (e.g., for planning work), team members only need to focus on their team's domain and achieve structure and clarity (Dalton and Spiller, 2012; Tamiru, 2023). This should also eventually enable the team to focus on other quality aspects such as monitoring and observability (Skelton and Pais, 2019). Small teams which are highly focused make it possible to establish trust which enables experimentation and innovation (Skelton and Pais, 2019). As teams experiment and figure out ways to optimize their value delivery, they essentially are constantly learning. Thus, with the new structure, the teams are optimized for learning as suggested by Farley (2021). Providing a clear structure, having forums to share knowledge will aid continuous learning

	and help in building psychological safety in teams by partly fulfilling Edmondson's (2018, p. 159) leadership toolkit.
Lack of focus on product discovery, teams are rather output driven instead of outcome driven	The new teams will have their own domain to focus on independently as they have their own bounded contexts thus reducing dependences between teams and increasing autonomy, which should enable teams to discover new customer problems to solve and deliver continuously (Tune and Millett, 2017)
Mobile team cannot do backend work and needs to ask another team	Each new team is now staffed with "full stack" skillset, that is with frontend, backend, and mobile developers. This reduces dependencies with other teams enhancing autonomy (Stray, Moe and Hoda, 2018).

It is important to note that while the new team structure helps to build high performance teams, it is not a silver bullet. Performance is also affected by other aspects such as organisation culture, quality of tools used to develop software, motivation, knowledge, etc. among other things. Therefore, the case company should holistically address all issues. It is recommended to consider adopting modern software engineering practices described in section 2.1 as well as focus on developing the two core competencies described by Farley (2021) – continuous learning, managing complexity. Once the new teams are formed, special care must be taken so that the teams go beyond the storming phase described by Tuckman (1965).

5.2 Significance of the study

This study was a case study for a SaaS company. As each organisation has its own challenges and ways of developing software, it is not recommended to blindly adopt the same structure for other SaaS companies. However, the literature review regarding factors affecting software development team performance, ideas for splitting teams, common problems ascertained from the group interviews, and the questions from the group interviews will help to develop a similar study.

This thesis utilized the design thinking process to develop the new team structure which can be utilized to similarly develop team structures for other companies or departments within companies. Developing a team structure is essentially a problem and can be solved like any other problem using the design thinking process. This thesis successfully demonstrates the usage of the design thinking process in the field of organisation design.

5.3 Limitations

Researcher bias

The author holds a managerial position in the case company and there might have been inherent biases which may have impacted the interpretation of formulation of group interview, data collected from the interview, and interpretation of the findings whether the new team structure fulfils the goal of the study. Therefore, the findings in this study must be read by being mindful of the researcher's bias. As noted by Yin (2014, p. 73), the author was aware of being biased and therefore made attempts to be sensitive to contrary evidence and be a good listener in the interviews to avoid being trapped into preconceptions.

Data collection and interpretation

One limitation of group interviews is that the opinions of one member can influence others which affect group dynamics and may not result in valid data to be collected (Frey and Fontana, 1991). To counter this, the author was sensitive to group dynamics as suggested by Frey and Fontana (1991) and intervened when such a thing happened. The size of the participants in the group interview may also influence the quality of data collected. Out of total 35 members of the engineering and product departments, a total of 19 participants were interviewed.

To be able to verify the reliability and validity of the findings in the future, the author has made the digital whiteboards available in the appendix 2 as evidence for the analysis.

Time and scope constraints

There were also time and scope constraints on this thesis due to which the author could not run a real testing of the proposed team structure. The testing and implementation steps are another project in itself and hence were considered out of scope of this study.

5.4 Concluding remarks

The thesis described a study conducted to find problems faced by the case company that affect software engineering team performance and ways to solve them using a new structure to create high performing teams. The theoretical framework for the study was built by doing a comprehensive literature review of modern software engineering, organisational factors that affect team performance, as well as organisation design concepts. The case study methodology used group interviews as primary data collection mechanism. The study employed the 5-step design thinking process with empathize, define, ideate, prototype, and test steps to study the current team structure, strategy, find problems with the structure, and then ideated and prototyped a new team structure.

Testing recommendations were also provided. The proposed team structure solves the major problems faced by the case company especially by reducing dependencies, lowering cognitive load, adding structure and clarity for each resulting team.

The case company is at a foundational stage where setting a right structure for teams and the resulting software is crucial for long term success. The author hopes that the new proposed structure helps the case company achieve this feat.

The author is an engineering manager at the case company and this study has been beneficial in gaining knowledge about team performance, organisation design and modern software engineering practices, data collection for solving organisational problems, etc. which will help the author in his future career.

5.5 Future research

The author was unable to test the proposed team structure and get feedback duo to time constraints. It would be interesting to study how to test a proposed software engineering team structure to gather feedback for the next iteration. Another area for research is to study efficient ways to roll out a new team structure with effective change management.

References

Beck, K. *et al.* (2001) 'Manifesto for Agile Software Development', *Manifesto for Agile Software Development* [Preprint]. Available at: http://www.agilemanifesto.org/.

Cockburn, A. and Highsmith, J. (2001) 'Agile software development, the people factor', *Computer*, 34(11), pp. 131–133.

Conway, M.E. (1968) 'How do committees invent', *Datamation*, 14(4), pp. 28–31.

Dalton, A.N. and Spiller, S.A. (2012) 'Too much of a good thing: The benefits of implementation intentions depend on the number of goals', *Journal of Consumer Research*, 39(3), pp. 600–614.

DevOps tech: Trunk-based development (no date) *Google Cloud: Cloud Architecture Center*. Available at: https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development (Accessed: 7 October 2023).

Edmondson, A.C. (2018) *The Fearless Organization: Creating Psychological Safety in the Workplace for Learning, Innovation, and Growth*. Wiley.

Evans, E. (2004) *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Farley, D. (2021) *Modern Software Engineering: Doing What Works to Build Better Software Faster*. Addison-Wesley Professional.

Fowler, M. (2013) *ContinuosDelivery*, *martinfowler.com*. Available at: https://martinfowler.com/bliki/ContinuousDelivery.html (Accessed: 7 October 2023).

Fowler, M. (2014) *BoundedContext*, *martinfowler.com*. Available at: https://martinfowler.com/bliki/BoundedContext.html (Accessed: 20 October 2023).

Frey, J.H. and Fontana, A. (1991) 'The group interview in social research', *The social science journal*, 28(2), pp. 175–187.

Gray, D. and Vander Wal, T. (2014) *The connected company*. ' O'Reilly Media, Inc.'

Harris, C.L. and Beyerlein, M.M. (2003) 'Teambased organization', *International handbook of organizational teamwork and cooperative working*, 187.

Herbsleb, J.D. and Grinter, R.E. (1999) 'Architectures, coordination, and distance: Conway's law and beyond', *IEEE software*, 16(5), pp. 63–70.

Interaction Design Foundation - IxDF (2016) *What is Design Thinking?*, *Interaction Design Foundation - IxDF*. Available at: https://www.interaction-design.org/literature/topics/design-thinking (Accessed: 22 October 2023).

Kates, A. and Galbraith, J.R. (2010) *Designing your organization: Using the STAR model to solve 5 critical design challenges*. John Wiley & Sons.

Klimoski, R. and Mohammed, S. (1994) 'Team mental model: Construct or metaphor?', *Journal of management*, 20(2), pp. 403–437.

Lehman, M.M. (1996) 'Laws of software evolution revisited', in *European workshop on software process technology*, pp. 108–124.

Lethbridge, T.C., Sim, S.E. and Singer, J. (2005) 'Studying software engineers: Data collection techniques for software field studies', *Empirical software engineering*, 10, pp. 311–341.

Nadler, D., Tushman, M. and Nadler, M.B. (1997) *Competing by design: The power of organizational architecture*. Oxford University Press.

Parnas, D.L. (1972) 'On the criteria to be used in decomposing systems into modules', *Communications of the ACM*, 15(12), pp. 1053–1058.

Pojasek, R.B. (2000) 'Asking"Why?" five times', *Environmental quality management*, 10(1), pp. 79–84.

Qian, L. *et al.* (2009) 'Cloud computing: An overview', in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*, pp. 626–631.

Runeson, P. *et al.* (2012) *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

Sage, A.P. and Rouse, W.B. (2014) *Handbook of Systems Engineering and Management*. Wiley (Wiley series in systems engineering and management). Available at: https://books.google.fi/books?id=eFRwQuzPnEcC.

Schmidt, C. *et al.* (2014) 'How agile practices influence the performance of software development teams: The role of shared mental models and backup'.

Skelton, M. and Pais, M. (2019) *Team topologies: organizing business and technology teams for fast flow*. It Revolution.

Smite, D. *et al.* (2019) ‘Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations’, *Ieee Software*, 36(2), pp. 51–57.

Stray, V., Moe, N.B. and Hoda, R. (2018) ‘Autonomous agile teams: challenges and future directions for research’, in *Proceedings of the 19th international conference on agile software development: companion*, pp. 1–5.

Strode, D.E. (2016) ‘A dependency taxonomy for agile software development projects’, *Information Systems Frontiers*, 18(1), pp. 23–46.

Tamiru, N. (2023) *Team dynamics: Five keys to building effective teams*. Available at: https://www.thinkwithgoogle.com/intl/en-gb/consumer-insights/consumer-trends/five-dynamics-effective-team/ (Accessed: 16 October 2023).

‘The Subscription Economy Grows More Than 300% In The Last Seven Years’ (2019) *Businesswire* [Preprint]. Businesswire.com.

Thompson, L.L. (2008) *Making the Team: A Guide for Managers*. Pearson/Prentice Hall (Pearson education international). Available at: https://books.google.fi/books?id=GpIiAQAAMAAJ.

Tuckman, B.W. (1965) ‘Developmental sequence in small groups.’, *Psychological bulletin*, 63(6), p. 384.

Tune, N. and Millett, S. (2017) *Designing Autonomous Teams and Services*. O’Reilly Media, Incorporated.

Waters, B. (2005) ‘Software as a service: A look at the customer benefits’, *Journal of digital asset management*, 1, pp. 32–39.

Yin, R.K. (2014) *Case study research: Design and methods*. 5th edn. Sage.

Appendices

Appendix 1. Semi-structured group interview structure

Following guiding questions were presented in the group interview, which was structured in two parts.

Part 1

1. As a member of a team, what are the main issues have you faced during development (in product or design or engineering)?
2. How many domains does your (or our) development team(s) have to deal with?
3. Any specific problems when trying to deliver value in the domains?
4. Are members able to work on any new task in the team (based on their expertise), efficiently?
5. Are the teams able to manage technical debt and make improvements to code and architecture?
6. How do you feel about the autonomy, mastery, and purpose as a part of your current team?

Part 2

7. What domains do you think are there in the product? Please classify these domains into three categories – simple, complicated, and complex.
8. How would you structure teams from your perspective?

Appendix 2. Interview results

Digital boards created during focus group interviews can be downloaded from the following link: https://drive.google.com/drive/folders/1b0pJJXGKliGRYlY70Jtd7-_UCkvGLBk6?usp=sharing

The digital board files names reflect the results obtained during each focus group interview:

Focus group 1 (engineers).pdf
Focus group 2 (senior engineers).pdf
Focus group 3 (engineers).pdf
Focus group 4 (engineering managers).pdf
Focus group 5 (product and design).pdf
all focus group interviews.pdf – this file only combines all the above files for easy reading.

Note: The digital boards are in PDF format, and it is recommended to download the files and view them in in a PDF viewer so that it is possible to zoom easily.