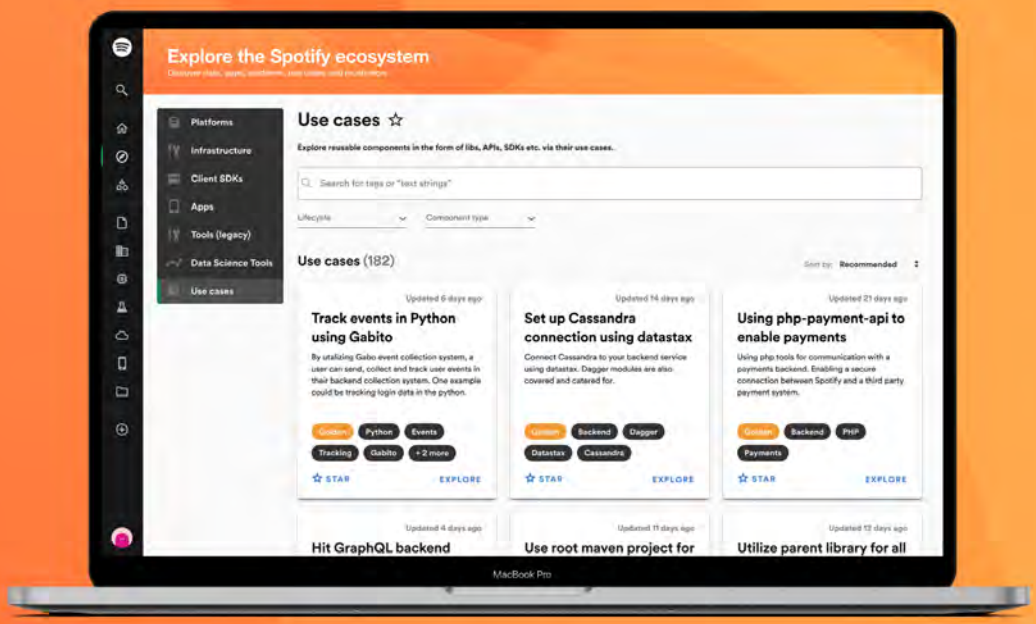
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Cracking the Bottleneck of Productivity by Enhancing Discoverability

Leveraging interaction design to enhance discoverability of reusable code components.

Master's thesis in Computer science and engineering

OSCAR FREDRIKSSON
ELIAS LIND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Cracking the Bottleneck of Productivity by Enhancing Discoverability

Leveraging interaction design to enhance discoverability of reusable code components.

OSCAR FREDRIKSSON
ELIAS LIND

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Cracking the Bottleneck of Productivity by Enhancing Discoverability
A master's thesis about leveraging interaction design to enhance discoverability of reusable code components.
OSCAR FREDRIKSSON
ELIAS LIND

Supervisor: Mafalda Samuelsson-Gamboa, Division of Interaction Design, Department of Computer Science and Engineering

Advisor: Lee Mills, Spotify

Examiner: Staffan Björk, Division of Interaction Design, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The Use Case Marketplace discovery page.

Typeset in LaTeX
Gothenburg, Sweden 2020

Cracking the Bottleneck of Productivity by Enhancing Discoverability
A master's thesis about leveraging interaction design to enhance discoverability of reusable code components.
OSCAR FREDRIKSSON
ELIAS LIND
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In the landscape of digital companies, speed, efficiency and productivity have become important tools to master in order to retain the customers of today and win the customers of tomorrow. However, previous research within the realm of productivity in software driven companies indicate that measuring and, thereby controlling, productivity in software development projects is difficult. Instead, many organizations have reverted to aiding developers in being as productive as possible by removing obstacles and productivity bottlenecks.
This master's thesis project explores how to leverage discoverability as a means to positively impact productivity by enhancing discovery and interpretability of reusable code.
The investigation of the topic is built upon studying previous academic literature as well as thorough qualitative research on productivity and discoverability problems within a large software driven enterprise. This research consisted mainly of user research in the form of interviews and focus groups that was used as fuel for the exploratory design work that followed, with the final goal of presenting a list of user experience factors to consider when designing for discoverability. Alongside the formulation of these factors, a design concept manifesting the factors whilst addressing found issues of code exploration and evaluation was developed in an iterative design thinking approach.
The result is a marketplace solution that surfaces reusable code components to developers via their intended use cases. Finally a set of 15 user experience factors were formulated, aiming to inform design decisions on discovery and interpretability in future developments of enterprise software.

Keywords: Interaction design, discoverability, productivity, user experience factors, enterprise software, reusables.

Acknowledgements

First of all, we would like to thank Mafalda Samuelsson-Gamboa for her tireless work, providing us guidance and support over the course of this thesis. Always on call and always ready to help. Thank you.

We would also like to dedicate a special thanks to everyone we have met at Spotify, especially the ones that participated in our interviews, workshops and user tests. Thanks for inviting us to be part of the band and sharing personal stories and ideas throughout the project. Special thanks to Lee, Stefan, Bill and the good people from Tools and Pulp Fiction. Thanks for all your help and enthusiasm!

In addition, we would like to express our sincerest gratitude to our families and friends who have stood by us, not only during this thesis project, but during our university studies over multiple years. Your love and friendship is what matters the most in the end.

Oscar Fredriksson and Elias Lind, Gothenburg, June 2020.

Contents

List of Figures

Acronyms

ACD Activity Centered Design. 18
API Application Program Interface. 58, 61, 62, 83, 92
DX Developer Experience. 19
FAQ Frequently Asked Questions. 67
GHE Github Enterprise. 6, 11, 58, 93, 106
HCD Human Centered Design. 18
HMI Human-machine Interaction. 21
HMW How Might We. 27, 51, 55, 56, 107
IDE Integrated Development Environment. 7, 9, 10, 60–62, 92
POV Point of View. 27, 51, 55
SDK Software Development Kit. 41, 54, 58, 61, 62, 83, 92
UCD User Centered Design. 17, 18
UX User Experience. 18, 107
UXF User Experience Factor. 18, 44, 52, 68, 74, 75, 77, 93, 104, 107–109, 111, 112
WFH Work From Home. 103, 104

Glossary

Application Program Interface A set of routines and protocols that defines interactions between multiple software items. xix

auto-complete A popular feature in search systems that displays a list as the user types with popular options matching the entered symbols and letters. 64

benchmarking The practice of comparing best practices from different companies. 63

business code Code that intrinsically has influences on the Spotify business goals. 6, 112

design system Design rules, constraints and principles to provide coherent, systemic order in systems. These can encompass anything from colors, fonts, figures, icons, spacing etc to how a table is supposed to be displayed or how specific types of information should be conveyed.. 81

individual contributor An engineer at Spotify who's daily work includes coding. 1, 62, 70, 103

Integrated Development Environment An application that combines common coding activities such as; editing source code, building executeables, and debugging, into one software. In other words, an umbrella-term for the software application used by developers to write, edit and test code. xix, 7, 61

library A collection of resources used in computer software programming. These may include configuration data, pre-written code and classes, values or type specifications. A library is also a collection of implemented behavior. This means that a developer who want to write a program can use a library to make system calls instead of implementing those system calls over and over again. 50, 54, 58, 61, 62, 69, 76, 92

markdown file A markdown file or .md is a text file with text written in a lightweight markup language with plain-text-formating syntax, such as *bold* for bold text. 12, 92, 93, 106

mission A mission is a group of tribes with a similar focus area. 10

platform mission Eight aligned tribes working with developing the internal platforms for Spotify's ever growing body of engineers and developers. 37, 42

reusable An existing artefact that could be utilized, and therefor reused, when building new software systems or features. 8, 48, 50, 51, 57, 58, 105

Software Development Kit Is a collection or a bundle or software development tools in one installable package. xix

squad A small, self-organizing cross-functional teams usually made up of less than 12 people. 6, 10, 48, 76, 110

tacit knowledge Tacit or implicit knowledge is knowledge that is not codified and therefor is hard to explain. It is most communally acquired through experience, see 2.2.4. 47, 58

tribe Consists of multiple squads, usually made up of a total of 30-150 people, with the collective ability to produce live product features end-to-end. 10

yaml file A human readable data file, commonly used for configuration files and in applications where data is being stored or transmitted. 93, 106

1

Introduction

In the landscape of digital companies, speed to market has become one of the most important tools to master in the race to win the customers and markets of tomorrow [62]. One important reason is the similarities in product value proposition amongst digital companies and organisations. The main features in services such as Spotify, Apple Music, Amazon Prime Music, Tencent Music etc. are essentially the same; to provide on-demand music streaming around the clock from a giant library of artists, bands, and genres. This forces each competitor to introduce new and innovative features that enriches the key functionalities to eventually differentiate themselves from the others whilst still adapting to the ever-changing customer preferences [47]. Being a global actor in an immensely competitive space, Spotify have to wield the tool of speed, efficiency, and productivity as accurate and precise as possible to retain the customers of today whilst winning the customers of tomorrow.

What began as a start-up in an apartment outside of Stockholm, has today, 14 years later, turned into a billion dollar company with offices all over the world and a growing fleet of 5800, and counting, employees. Out of these, over 1000 are considered software developers, also known as individual contributors, that everyday strive to improve or add features and functionality to the Spotify product portfolio. Growing at blitz scale is a sign of success, but the same growth also renders clouds on the horizon [61].

The byproduct of more users and an expanding feature portfolio is an ever-growing code stack. Modern repositories no longer contain 30 to 250 components, as claimed by Poulin in 1999 [63], but rather thousands of components. This is making it increasingly difficult for a single developer to keep track of all the code written at a contemporary global software company [85]. Knowing the whole code repository of Spotify today is next to impossible. So is staying up to date with the 250 development teams and what they are currently working on. Internal research suggest that the company's new found scale does not only affect the software development negatively, it affects discoverability of information in general. Documentation and information becomes fragmented and hard to find. This in turn, forces new hires to ask questions to close colleagues, or in different forums, in the pursuit of assistance and a chance to get a systematic overview. Add to that frequent organizational changes and therefore a hard time of keeping track of who owns and knows what, and the negative spiral is born. Whilst sounding like small problems in isolation at

first, over time, thousands of engineering hours will have been wasted as a result of bad discoverability.

1.1 Aim and Research Question

This thesis aims is to explore how principles of user experience and interaction design can be applied to understand, and eventually solve, the aforementioned discoverability issue and productivity bottleneck. This will be done by presenting a list of user experience factors to consider when designing features for a discoverability system in the context of an enterprise software. The list of factors will be backed by thorough user research and understanding of the current development situation at Spotify, as well as user testing after having applied and evaluated the factors in a final design concept. More specifically, the thesis aims to answer the following research question:

Which user experience factors should be considered when designing for discoverability in the context of enterprise software to enhance productivity?

1.2 Stakeholders

The following stakeholders have been identified for the project, either by being interested in, or affected by the implications of the results.

1.2.1 Spotify

The project is a collaboration with Spotify AB in Stockholm. A supervisor provided by the company is assigned to support the duo in their progression. Since time and efforts are allocated to the project from the company, there are expectations of deliverables by the end of this thesis, see 1.3.

1.2.2 Developers at Spotify

The target user group of the study, and furthermore the group to improve the everyday productivity of, are the members of the development teams at the company. These users are hence of greatest importance to understand and cater for.

1.2.3 Platform Developer Experience tribe at Spotify

The team working with internal systems to improve the working experience of developers is called the Platform Developer Experience tribe. This team is interested in the findings of this project to better plan for the future allocation of resources and prioritization of feature and solution enhancements to their platform.

1.2.4 Chalmers University of Technology

The school and especially the academic institution of Interaction Design and Technology has an interest in the thesis duo performing well to further spread the good name of the school and institution. The institution provides a supervisor whom will advice the thesis duo on the realm of Interaction Design as well as in the writing of the master's thesis. In the end, an examiner has the final word and will be the one deciding if the report, as well as presentation, holds sufficient academic height.

1.2.5 Thesis duo

Since this project is the final step of both students master's degree, there is an inherent willingness to both prove the knowledge learned over the course of the university studies as well as gaining new insights during this thesis semester. For potential future employment, conducting a thorough project to be proud of, at an admired company like Spotify, could be very valuable, hence, doing a good job is of essence.

1.3 Goals and Deliverables

Other than providing a deeper analysis of the bottlenecks of productivity and, more specifically, problems with discoverability in the context of software development at Spotify, the goal is to present a list of user experience factors and a design concept that manifests and exemplifies these. This design will furthermore be user tested and evaluated based on the user experience factors through a medium to high fidelity prototype.

A final document with links to research insights and a final project presentation will mark the end of the project at Spotify.

1.4 Delimitations

In this thesis, the research and inquiries will be limited to the context of software development at Spotify. The target users will be developers at the company and whilst the development environment may share processes and tools with other major tech companies, no two companies are exactly the same. The findings and results may, hence, be more or less applicable depending on how many attributes are shared with the investigated company.

Within the frame of this thesis, productivity and the way to enhance it has been tackled by researching and eliminating productivity blockers within existing tools and infrastructure. This means that productivity from an management or organizational aspect has been disregarded. What else has been left out is the examination, research and evaluation of similar, productivity enhancing enterprise software applications. The focus of this thesis has been on understanding the current problems of developers at Spotify and solving for those.

Furthermore, the thesis will only investigate discoverability of internal code and information, i.e. content produced by Spotify and its employees, not external content. Whilst the aim is to construct general guidelines for discoverability to enhance productivity, it will be outside the scope of the thesis to evaluate these guidelines in any other context than within Spotify's ecosystem. Furthermore, the user experience factors will mainly be based on insights from the user research conducted within this master's thesis.

Finally, the project will not include any development of the envisioned design concept, the focus will rather be to design and evaluate prototypes simulating the intended functionality. Regardless, the technical feasibility will still be taken into account by consistently involving developers and stakeholders throughout the process.

2

Background

This chapter aims to give the reader a better understanding of the Spotify domain, e.g. organisational constellation, current tools and infrastructure, developer productivity in general and discoverability in particular. Some of the previous related works within the subject will also be highlighted.

2.1 Productivity

This section handles productivity from a software standpoint and starts by referencing how productivity within software projects and organizations have been viewed in the past and how it is now defined and quantified at Spotify. Spotify's definition of productivity is also the definition that will be used in this master thesis project.

2.1.1 History of software developer productivity

The urge of measuring productivity amongst developers is not new. In the book *Rethinking productivity in Software Engineering* by Sadowsky and Zimmermann, [66] Stefan Wagner and Florain Deissenboeck argues that most software companies eventually reach a need of measuring productivity. This is a result of the fact that the time to implement software changes, largely determine the ability to adapt the business to ever-changing market situations and to quickly implement innovative products, features and services that might be essential for the survival of the company [80].

Measuring software productivity has proved to be way more difficult than measuring production processes in other industries. The reason for this is the fact that software companies typically produce new products with consistently evolving features and functionalities as opposed to fabricating the same products repeatedly. Furthermore, software development is a human-based activity with utmost uncertainties from the starting point of each project. All this together results in big difficulties in creating a reliable definition and universal metric [78].

In a literature study done by Trendowicz and Munch, called *Factors Influencing Software Development Productivity – State of the Art and Industrial experiences*, they categorized and annotated 142 references from state of the art research result-

ing in figure 2.1 presenting the most prevalent software development productivity factors stated within academic literature. The study claims that the success of software projects, in terms of productivity, relies upon humans. A well aligned team with experiences matching the problem at hand is really key to perform well. The third most commonly considered factor is tools and methods used. As stated by Trendowicz and Munch:

> *"However, even the best tool or method alone is not a silver bullet and cannot be a substitute for highly skilled people and effective work coordination... Tools and methods should therefore be considered as human aid that amplifies the positive impact of highly-skilled and well-coordinated teams on development productivity"*[78].

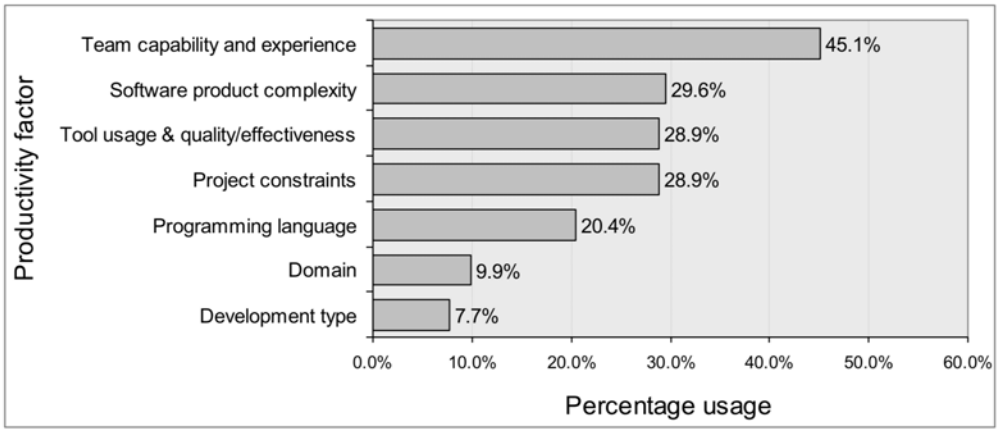


Figure 2.1: Most common productivity factors from the literature review. [78]

2.1.2 Productivity measurement at Spotify

Spotify has recently started to invest more time and efforts into removing their productivity blockers by research and product development. This initiative encompasses researching productivity on an individual and squad level as well as building and maintaining internal enterprise software and platforms for more efficient end-to-end development at scale.

For Spotify, similarly to the traditional way of measuring productivity in industrial production processes [78], productivity is defined as the ratio of outputs to inputs. For the development teams at Spotify, outputs is computed as the amount of changes in the customer's products, i.e. changes of the business code, whilst inputs are the hours of engineering time available within each team. To gain productivity, the goal is to deliver more changes per unit of time [73]. This is also the definition that will be used for this master thesis project.

As of now, Spotify measures productivity by mapping the number of code changes made to Github Enterprise [39]. The rationale is based on the belief that end user value created by developers at Spotify originates in code. Frequent change then becomes an indicator of productivity. By calculating the average size of a code

change and other differences in developer behaviour, Spotify hope to work around the inherent problems of the metric regarding measuring quality and weight of the code, mentioned above [73].

2.1.3 Research in software developer productivity enhancement

Previous works, many of which sponsored by influential tech companies, has focused on developers' activities in the Integrated Development Environment (IDE), i.e. the software application being used to write, debug and understand code; their testing and refactoring practices and the time they spend on understanding versus actually writing or editing code. Other research has gone into looking at the developers' workdays from a more holistic level, investigating how they make use of their time at work. Results suggest that developers spend anywhere from 9% to 61% of their time writing code [52]. In a research study from Microsoft, with over 5900 engineers participating, they showed that about 48% of work time was spent on developer activities where 15 percentage points of these were spent on writing actual code. When developers got to classify their days as a good or bad work day, 7 percentage points of coding activity differed between the two, indicating that a slight raise in amount of time a developer gets to code can change his or her outlook on a good, productive work day [42].

2.2 Discoverability

Discoverability concerns the quality of an item being easy to find. Additionally, according to Shanmugasundaram et al., the *discovery* of an item is only one of two cornerstones of discoverability. The other part is the *interpretability*, addressing the understanding of how to actually make use of the discovered information [71]. In this thesis, discoverability will be defined as the quality of both of these, discovery and interpretability of items or assets.

Software engineering is a knowledge-intensive activity, with developers using a plethora of diverse technologies and complex frameworks to guarantee high quality of their work and products whilst also maximizing their productivity. To do so, developers oftentimes require knowledge outside of their own to complete tasks at hand. This usually leads to help-seeking amongst co-workers, documentation, manuals and digital information resources. With a constantly growing body of knowledge, the ability to discover and interpret, i.e. to search and understand the knowledge, is key to software engineers' efficiency and success [49].

Relevant topics of discoverability will be described in the sections below.

2.2.1 Code search

Code search, like the name suggests, refers to the act of running searches on the actual code base as a means to get information, to learn and develop. In a case study

at Google, researchers argue that code search is becoming increasingly important for software development, as large code repositories become more of a norm. The study shows that searching for code is used to answer a multitude of questions about a wide range of topics such as *what* code does, *where* the code is located within the repository, *why* the code is behaving in a certain way, *who* is responsible for an edit, *when* an edit occurred and *how* to perform a specific task [65].

The main reason to why developers use code search is to find existing code examples of *how* others have implemented things in the past. These kind of searches that represents a need for help and understanding stands for more than 1/3, by far the greatest reason, of all the code searches being made at Google. See figure 2.2 for more details [5].

Category	Example	Count	Percent
Example Code Needed (How)		**87**	**33.5%**
API consumer needs help	"I want to know how a function should be called."	57	22%
Discover correct library for task	"Best way to convert output stream into a string of limited length."	14	5%
Example to build off of	"Just want to copy-and-paste some code I'm changing"	9	3.5%
How to do something	"How to write a hash function"	7	3%
Exploring or Reading Code (What)		**67**	**26%**
Check implementation details	"What a particular script does"	51	20%
Browsing	"Re-familiarizing myself with some code referenced in a CL under review."	11	4%
Check common style	"Where are friend classes usually declared?"	3	1%
Name completion	"I'm looking for an enum member that I think begins with a particular prefix."	2	1%
Code Localization (Where)		**41**	**16%**
Reachability	"Where a class is instantiated"	22	8.5%
Showing to someone else	"I'm trying to create a link to a known piece of code, to provide to someone else."	10	4%
Location in source control	"Where are all the boxed environment configurations?"	9	3.5%
Determine Impact (Why)		**42**	**16%**
Why is something failing	"I'm wondering why a CL I wrote earlier did not fix a currently-occurring production problem and am reading the code to diagnose."	26	10%
Understanding dependencies	"Looking for dependencies of a build file"	12	4.5%
Side effects of a proposed change	"Am I about to blow up production with my CL"	4	1.5%
Metadata (Who and When)		**22**	**8.5%**
Trace code history	"Who last touched some code."	13	5%
Responsibility	"Who is allowed to approve changelists for a particular file."	9	3.5%
Total		259	100.00%

Figure 2.2: Table showing answers to what question developers are trying to answer by engaging in code search [65].

Another report suggests code search as the main way developers retrieve and understand *reusables*, i.e. existing artefacts that could be utilized when building new software systems or features. This makes the act of code searching, and the tools that permit it, play a very important role since finding, understanding and using reusables allows the developer to produce at a higher pace with lower maintenance and higher code quality [49]. The reasons to making use of reusables are many. Companies have reported increased productivity, shortened time-to-market, improved software quality and consistency as results from enhancing code reusability. Furthermore, reusables produced by other developers and teams promotes availability of new functionality, higher consistency and regular bug fixes which are beneficial to developer productivity [5].

However, research also show that software developers are unable or unwilling to initiate the search for reusables if they do not have the sufficient knowledge about their existence or do not know how to locate them. Other research however, indicate that developers would put a great amount of effort into both locating and actively reusing components if they were aware of the existence of the component that could be reused [85].

2.2.2 Interpretability research

Looking into software developers' help seeking behavior, research has concluded that developers spend more time on selecting, integrating and verifying help information than actually searching for it, indicating interpretability and particularly evaluation of help information to be the biggest time drain [49]. According to the research, help seeking involves tasks, other than search, such as browsing thought multiple alternatives, comparing and assessing those and eventually selecting and integrating solutions for several sources in an iterative manner. The study suggests that an effective help seeking tool must take into account four proposed phases of help seeking, namely *search, select, integrate* and *verify* [49]. A tool could do this by meeting the information need of the developer and helping him or her to determine the relevance of returned results, especially when they are plentifully. One article suggests an approach to allow the developer to start with a rough query and then allow for gradual refinement and adjustment based on the system feedback [81].

2.2.3 Tools for discoverability assistance

Below are a three examples of products that have been developed through research to assist software developers in their search for help and guidance.

Multi-Faceted Interactive Explorer

Multi-Faceted Interactive Explorere or MFIE is a tool that can integrate with a developer's IDE to facilitate easier feature location. Its goal is to alleviate exploration and understanding of potentially relevant program elements that can be reused or used as examples for code understanding by developers. This is done by automatically extracting and mining multiple syntactic and semantic elements from a users software elements, allowing developers to search and then interactively group, sort and filter feature location results [81].

Strathcona Example Recommendation Tool

Strathcona is an example recommendation tool developed to help developers quickly access code examples by finding and presenting contextually relevant examples from the source code directly into the developer's IDE [40].

Codebroker

Codebroker is an autonomous search and delivery application that enables reuse of unknown software components by analyzing the code being written and proposing

potential reuse components for the task at hand. This is done by adding a plugin to the developer's IDE [85].

Expertise Browser

Finding relevant expertise is critical in software engineering. Finding needed knowledge is especially difficult in geographically distributed teams, where the distance, lack of face-to-face contact, potential cultural differences as well as time differences create a barriere for effective communication. Expertise Browser makes use of data from change management systems to locate people within the organisation with the desired expertise. By quantifying the size and amounts of code changes within a repository, the tool can assist developers in distinguishing how immersed a developer is in a specific project [53].

2.2.4 Tacit knowledge

Tacit or implicit knowledge is a type of non-codified knowledge that is hard to transfer to another person by means of writing down or verbalizing it. In knowledge management, tacit knowledge is known for being acquired mostly by experience and without some form of shared experience, it is extremely difficult to convey to another human being. Tacit knowledge is typically achieved from observations, imitations and practice [48].

2.3 Domain

Spotify has a unique organisational set-up made up of 250, and counting, autonomous end-to-end *squads*. Squads are small, self-organizing cross-functional teams, i.e. teams with different disciplinary expertise, usually made up of less than 12 people. Ideally, squads sit together and are grouped into different *tribes*. Tribes can be seen as overarching teams with different focus areas which in turn are part of a *mission* - a group of tribes with similar focus areas, tackling similar problems or going after similar opportunities [46].

2.3.1 Autonomous team setups

Being autonomous essentially means squads can decide what to build, how to build it and how to work together. Another way of looking at it is saying they have many degrees of freedom in terms of what software to use, frameworks to build upon and methods to incorporate. Some boundaries exist, coming mainly from the predefined squad vision, the current product strategy as well as other short term goals [46].

The reasoning behind the autonomous setup is to ultimately make Spotify as fast and nimble as possible. By making decisions happen locally within the squads, Spotify are able scale without a lot of communication and organizational dependencies ultimately hindering the velocity. As mentioned earlier, speed in general is of vital

importance when going after the customers of tomorrow, proven by this quote from CEO Daniel Ek:

> *"If we figure out how to move faster, we can afford to be wrong more often, which gives us a better chance of getting it right in the long run."*

2.4 Tooling & Infrastructure

There are fundamental tools and infrastructure in place to enable squads to perform their everyday work. The nature of the tools are different, but they all share a common goal of making otherwise tedious processes more efficient, hence more productive, for developers at the company.

2.4.1 Slack

For communication, the main tool used is Slack. Slack not only allows instant messaging between colleagues but it also enables so called *channels* where anyone can start a *thread*, typically a question or post. Threads are transparent and open for everyone to see and contribute to. Each channel can be seen as a forum where the name is based on its theme or working area, e.g. each squad and tribe have their own channel, and hence acts as a way of categorizing the information [72].

2.4.2 Github Enterprise

Github is a gobal software company, providing hosting for software development version control using Git. Git tracks changes in source code during software development and lets developers see and track changes done by them selves or someone else. [69]. It is a tool designed mainly for coordinating and sharing work among programmers, introducing speed and data integrity in distributed, non-linear workflows [77]. Other than Git, Github also offers its own unique features such as access control and a multitude of collaboration features like feature requests, task management, bug tracking etc. [83].

Amongst developers at Spotify, Github Enterprise (GHE) is used for all the functionality mentioned above, as well as for its code search functionality, allowing users to search the full code repository of Spotify for examples of how specific libraries are used.

2.4.3 Stack Overflow Enterprise

Stack Overflow is an open, collaborative platform for programming questions and answers. A set of features are put into place to motivate contributors, such as awarding reputations points to users who contribute with valuable insights or knowledge [70]. Stack Overflow Enterprise has the same basic functionalities of Stack Overflow, it is just packaged to be used by a single company – allowing questions and answers

on private, company specific code, frameworks and practices to be accessible in one place [24].

The reason of introducing Stack Overflow at Spotify is to make information and knowledge accessible to a large audience.

2.4.4 Backstage

Backstage is Spotify's centralised hub where engineers and data scientists manage the individual software components they own and discover others. The ambition is to consolidate and unify the infrastructure tooling into one central place for developers to more easily discover what is already available, to increase reuse and move faster on product [3].

Backstage is an internal platform built to be an extensible ecosystem. Via a plugin, any squad can contribute with components for any infrastructure use cases. Backstage shows ownership of a component, how to technically call on and use it, what systems it belongs to and where one could find documentation about it. As an owner of a service, Backstage can be used to keep track of the status of the services, monitor tests etc. [4].

Another part of Backstage is the concept of *Golden paths*. A Golden path can be described as the best practice of a particular development domain at Spotify; back-end, web development , data engineering etc. It is the company gold standard and preferred way of setting up and building bigger standard projects within the Spotify ecosystem, for instance how to build a back-end service or a client feature, i.e. a feature of the Spotify app. As a newcomer to Spotify, ideally, you should go through at least your domain's Golden path to align on how things ideally are built. By following the golden path or blessed way of building a component, you are making sure that there is support for your frameworks and libraries, and can be sure your functionality will fit smoothly into the technology stack [56].

2.4.5 TechDocs

TechDocs is an internal documentation platform at Spotify making use of the docs-as-code framework where authoring and publishing documentation are integrated into the developer's tool chain and workflow. In the case of TechDocs, a folder is automatically created within the existing GHE code repository where the team that owns the code can generate or write the corresponding documentation as markdown files. From this folder, the documentation will be published automatically and at the same time becomes searchable in Backstage [35].

2.4.6 Spotify Code search

Codesearch is an internal tool at Spotify that started out as a side-project but later evolved to become a part of the current infrastructure. The idea of Codesearch is for developers to be able to look up code snippets across Spotify's code base.

Either to find an example of a code that does a similar thing to what you want to accomplish, or when you need to understand the production usages of a certain function or pattern.

The tool offers some basic filtering options to narrow the results as opposed to searching through the entire stack, including people's personal experiments that has not gone into production and hence might not be of interest [11].

2.4.7 Confluence

Confluence is an enterprise software for collaboration in bigger organizations developed and maintained by the software company Atlassian. By utalizing *spaces, pages* and *blogs*, content is created for the organization to have a single space for specific knowledge and information sharing [68].

Confluence serves as Spotify's internal portal for links, resources, platforms and company information. By gathering all this information at one central location, the idea is to avoid fragmentation. The core idea is that employees will be able to get things done more quickly when not having to spend time on trying to find the right things all over the place.

2.4.8 Google Suite

Spotify is using the Google Suite's fleet of tools to store files, create internal text documents, spreadsheets and presentations etc. A common use case is to create so called *Request For Comment (RFC)* documents, where the author basically is asking peers to contribute with feedback and comments on a specific matter. One member of the suite is Google Calendar, a common tool used for planning, booking of meetings and accessing colleague's calendars. Google Hangouts, Google's video conferencing tool, is another software used extensively for communication during meetings. Each meeting room at the company is set up with a TV and a web-camera, both connected to Hangouts. It is therefor possible to easily set up a video conference where people are able to join remotely.

3

Theory

The Theory section will cover the relevant frameworks and concepts used for the thesis project.

3.1 Wicked problems

Rather than tackling linear determinate problems that can be solved by applying conventional techniques for a finite time period, designers are mostly dealing with so called *wicked problems*. Coined by the mathematician and designer Horst Rittel, wicked problems are a *"class of social system problems which are ill-formulated, where the information is confusing, where there are many clients and decision makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing."* [64]

In contrast to other problems, wicked problems are inherently more complex and indeterminate. The indeterminacy is not to be confused with undetermined problems. As Buchanan stresses, the term indeterminacy simply suggest that there are no definitive conditions or limits to wicked design problems, which may make them seem as impossible to solve at first. Furthermore, wicked problems cannot be true or false, only good or bad. There is no complete list of suggested operations when solving wicked problems, neither is there a definitive test to the formulation or solution of them [12].

3.2 Research Frameworks

In this section, the different theoretical frameworks regarding research used for this research project is presented.

3.2.1 Ethnographic research approach

The ethnographic research approach stems from the field of social science and anthropology, where the main idea is to immerse oneself in the daily activities of the people one is trying to understand [21]. Malinowski, the first ever recorded ethnographer, developed the method by accident when, due to the outbreak of the First

World War, he was stranded and forced to live along side a group of native islanders he was studying. Afterwards, he famously advised his fellow researchers that: *"the fieldworker must spend at least a year in the field, use the local vernacular, live apart from his own kind, and above all, make the psychological transference whereby they becomes we"* [75]. The ethnographic approach is specific since it emphasises the natives' point-of-view, holism and the natural settings which provides a unique perspective on understanding users' actual activities [8].

In the chapter *Ethnographic Field Methods and Their Relation to Design* of the book Participatory Design: Principles and Practices by Schuler & Namikoa, the authors outline four main principles that ought to guide ethnographic work.

Natural Settings: The ethnographic method is grounded in field work and is committed to study people and their activities in their everyday settings. This then requires the research itself to be conducted in close contact with the research subjects in their natural environment. By doing so, the ethnographer would argue for specific leanings and observations that can only be found when experiencing it first hand.

Holism: The ethnographer takes the position that certain behaviours and actions only can be understood by the *context* they occur in.

Descriptive: By getting a first hand view of in field actions, the ethnographer develops a descriptive understanding of the research subjects which means a non-judgemental stance on actual behaviour. It lies in the nature of the ethnographic method to not judge one group's behaviour on the standards of some other faction [8].

Members' Point-of-View: To keep a non-judgemental stance, the ethnographer strives to understand the world from the point-of-view of those studied. This way, the ethnographer can describe behaviour in meaningful ways that relates to the study participants [21]. In comparison with quantitative research methods, ethnography allows for developing a *thick description* of social behaviour where the researcher is faced with *"a multiplicity of complex conceptual structures, many of them superimposed upon or knitted into one another, which are at once strange, irregular, and in-explicit and which he must contrive somehow first to grasp and then to render."* [29]

During this thesis, the ethnographic approach will be used in combination with Design Thinking and its first two steps, empathize and define, to create a complete understanding of the problem space from the developers point of view. By situating within Spotify itself, it will provide a first hand experience of the subjects' *natural setting* and for the problems to be understood from the *members' point-of-view.* This will allow the thesis duo to take in the context by getting a high-level overview of developers' work life situation and the context in which their behavior happens - *holism.* By going into each encounter with an open, non-judgmental mind and by not originating from the space of software development, it will allow the duo to take

a *descriptive* research stance.

3.2.2 Ethnographic research methods

The essence of ethnographic research methods primarily lies in different levels of observations and involvement with the research subjects [21].

Participant observation comes in four levels of involvement: complete observer, the observer-as-participant, the participant-as-observer and the complete participant. [32] As a researcher, the kind of information that can be gathered at the different levels vary and usually involves a trade off between affecting the research subject and maximizing the depth of information that is obtained [21].

Interviews, both formal and informal, usually compliments observations to derive deeper meaning and understanding from the observed. As stated by Elliot, ethnographic interviews tend to focus on the use of non-directed questions [21]. These questions work as "triggers that simulate the interviewee into talking about a particular broad area" [36].

3.3 Design Frameworks

There are a multitude of frameworks that can be used by researchers and designers to create great applications, the ones utilized for this project is described below.

3.3.1 User Centered Design

User centered design (UCD), is a framework introduced by Norman during the 1980's that holds the user as a central figure for the design process [82]. Garrett argues in his book, The Elements of User Experience: User-Centered Design for the Web and Beyond, that the concept user centered design is very simple: *"take the user into account every step of the way as you develop your product"* [28]. Making use of a UCD process forces the designer to never let go of the users unique point-of-view resulting in a user experience based on conscious design decisions [28]. *"The goal is to develop products or services that match users' practices, needs and preferences"* [74]. The UCD process is split up into three phases: *design research, design* and *design evaluation*. During research, the designer assesses who the users are as well as their needs, see also the phases Empathize 4.1.1 and Define 4.1.2 from the Design Thinking framework. The design phase uses the research findings from phase one to design solutions to found problems and/or pain points, see also the Design Thinking phases Ideate 4.1.3 and Prototype 4.1.4. During the last phase, design evaluation, the designer assesses the design concepts with users and revises as needed based on the evaluation results, see also Test 4.1.5 [82].

3.3.2 Human Centered Design

Human centered design (HCD), is an alternative framework, derived from UCD, addressing aspects left out from UCD. HCD takes a socio-technical view, balancing requirements from two competing systems: the social system of human goals, needs and aspirations, human understanding and knowledge as well as business and cultural aspects and practices versus the technical, static, rule-based system managed by performance metrics. HCD argues that the narrow view of the human as a user becomes techno-centric and that this attitude neglects the human role outside the one of the end user [74].

3.3.3 Activity Centered Design

Activity centered design (ACD) is another framework with roots in activity theory that can be traced back to a Soviet-era Russian theory of psychology. In contrast to other frameworks, ACD is not concerned with the user *performing* tasks or activities, but instead on what tasks and activities are *enabled* by the system. Norman argues in his article Human-Centered Design Considered Harmful that ACD allows a designer to design for the activity instead of the user, permitting it to be used by a multitude of user groups agnostic of time and place. Norman argues *"One concern is that the focus upon individual people (or groups) might improve things for them at the cost of making it worse for others. The more something is tailored for the particular likes, dislikes, skills, and needs of a particular target population, the less likely it will be appropriate for others"* [57].

3.4 Design Theory

Below, relevant design theory and principles are addressed.

3.4.1 User experience

As proclaimed by the definition from the International Organization for Standardization (ISO), user experience is *"users' perceptions and responses that result from the use and/or anticipated use of a system, product or service"* [76]. User Experience (UX) therefore , arguably, concerns more subjective values. Whilst not as quantifiable as technical aspects, it is as important to cater for since it encompasses the whole user journey and addresses the accessibility of a design [44]. UX takes a broad approach to task performance where it is put into perspective with aspects such as the subjective meaning of an experience, emotional aspects as well as in which context the interaction occurs [19].

3.4.2 User Experience Factors

As stated in the research question, the goal for this thesis is to present a list of User Experience Factors (UXFs). More specifically, in the case of this thesis, a user experience factor will be defined as a guideline to bear in mind when designing

a discoverability system whilst simultaneously advocating for the user's emotions, comfort and perceptions. These are broad recommendations or rules of thumb on of how to achieve potent discovery as well as interpretability in a system to assist the user in discovering content and assets. This definition is the thesis duo's terminology and is not based on previous academic literature.

3.4.3 Developer Experience

Evolved from the concept of user experience and capturing how people feel about products, systems and services, Developer Experience (DX) similarly strives as a mean to capture how developers think and feel about their activities within their work situation, assuming that improvement of DX will lead to positive impact on team and project performance [22]. Just like the user experience is concerned with the use of a product or service, developer experience is concerned with the development of a product or service. DX consists of experiences relating to *development infrastructure*, e.g. development tools, platforms etc, *feelings about work* such as respect, recognition or team and *the value of one's own contribution*, e.g. plans, goals, motivation and commitment [22], see figure 3.1.

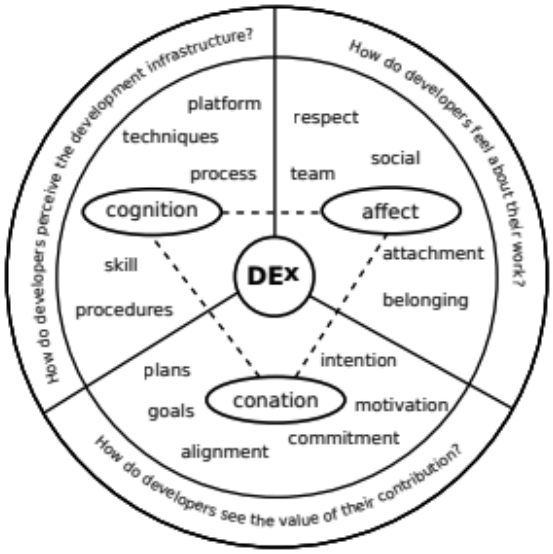


Figure 3.1: Developer Experience

3.4.4 Participatory design

Participatory design is a strategic design approach to bring the end user into the very heart of the design process, allowing for designing *with* the users instead of *for* them. Participatory design is about inviting the user to part take in more than just the early research and late evaluation phases, also in the idea generation and evaluation phase of the design process. By inviting the user to solve challenges they themselves face on a regular basis, designers can get insights in their actual experiences and let that focus their future design efforts. Ideas generated by the users can also act as great inspiration for solutions created [20].

3.4.5 Jordan's 10 principles of design

Senior human factor specialist Patrick W. Jordan introduces ten design principles that should be considered when developing new products and interfaces. Jordan argues that usable design should consist of, and have [45]:

- **Consistency** - Similar tasks should be solved in a similar way throughout the product.
- **Compatibility** - Tasks within the product should be solved in a conventional way compared to similar products or the outside world.
- **Consideration of User Resources** - consider the user's physical and mental limitations, for example by not presenting too much information at once.
- **Feedback** - Give the user response on their action, convey that they have been registered and provide meaningful information about the result.
- **Error Prevention and Recovery** - Minimize the likelihood of user error and make it simple and fast to recover from them if they do occur.
- **User Control** - The user should always feel like he or she has complete control of a systems actions or states.
- **Visual Clarity** - Information is presented in such a way that it is easily readable and understood without causing confusion.
- **Prioritization of Functionality** - Information and functions should be prioritized in such a way that the most important ones are the most accessible and presented in a logical fashion.
- **Appropriate Transfer of Technology** - Apply and take advantage of other technology from different areas to increase usability.
- **Explicitness** - Shape the product to give explicit clues of the product's purpose and use.

3.4.6 Norman's Fundamental Principles of Design

Some concepts regarding the cognitive psychology between a user and a product are used in user centered design to produce useful products. These concepts, as described in the book *"Design of everyday things"* by Donald A. Norman [58], are:

- **Discoverability** - Discoverability regards the possibility to determine what state the current device is in and what actions are possible within that state.
- **Feedback** - Feedback regards how well a system or product communicates the results of ones actions.
- **Conceptual Model** - Regards the way the product projects the information needed to create a good conceptual model of the product. Also, see Mental models, 3.4.8.
- **Affordance & Signifiers** - Affordance determine all actions that are possible

whilst signifiers are clues to how an action can and should take place.

- **Constraints** - How to limit a products use and interactions by its design. One example is the USB port that only allows a USB to be inserted in one specific way.
- **Mapping** - Regards the way different interface elements should be placed in a logical manner in relation to each other [58].

3.4.7 Ethical Interaction Design

In the book About Face – The essentials of Interaction Design, Alan Cooper stresses the importance of considering the ethical aspects when designing a system that effects a user's life [13]. According to Cooper, there are two fundamental guidelines to consider, namely to *Do no harm* and *Improve human situations*.

3.4.8 Mental models

Cognitive theory in regards to HMI and the ways that humans tend to create mental models of what a system is and is capable of, is central in achieving optimal usage of an interface. A mental model of a system is an internal representation of a system that describes, more or less accurately, what a user believes a system to be capable of. In the words of Donald Norman, *"a mental model...provide predictive and explanatory power for understanding the interaction* [59]. These mental models can then be leveraged by designers to structure their design in a way that inherently will become easy to understand for the user since they can imagine how to navigate it. A mental model can be created spontaneously based on a users first impression of a system or thought spending a significant time within a system [7].

3.4.9 Material Design

Material Design is a design language developed by Google. Material Design is an adaptable system of guidelines, components, and tools that support the best practices of user interface design for Android, iOS and the web [10]. The design language is famous for introducing and heavily relying on *cards* and *grid views*. Other components used plentiful within Material Design are selection controls like *chips, buttons* and *check boxes* and elements such as *tabs* and *tool tips*. Material design will be used extensively during the design phase of this thesis [34].

4

Methodology

In this section, the methodology used for this thesis will be explained. The presented methods are picked with an ethnographic approach in mind, focusing on being present and observing in the actual field of action, i.e. the office sites of Spotify.

4.1 Design thinking

Design thinking is a design methodology, popularized at Stanford and IDEO, providing a solution-based approach to solving problems [17]. As a framework, it is being used by practitioners to resolve and frame wicked problems, as previously described, as well as co-evolving the problem space with the solution space [15]. It does this by (1) understanding the human needs involved, by (2) re-framing the problem, by (3) creating a plethora of ideas in creative sessions and testing these ideas and assumptions via (4) prototyping and (5) testing. This approach, brings together what is desirable from a human and user point of view with what is technologically feasible and economically viable, see figure 4.1. The ethnographic approach influences the types of methods chosen for each step by the need of being and observing within the field of actions. The five steps, and approaches for each, are described in detail below.

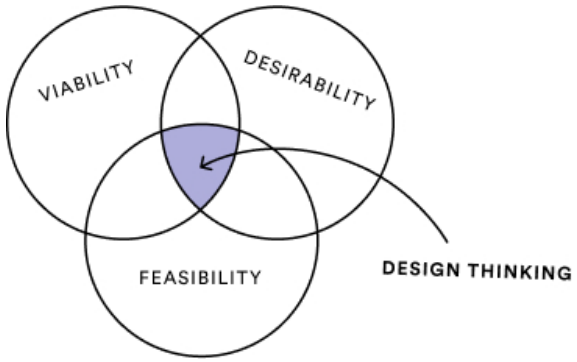


Figure 4.1: The intersection of innovation where design thinking lives.

4.1.1 Empathize

Empathizing with the user is the first stage of the design thinking process where the overarching understanding of the problem space is explored. This is mainly done by empathizing with the current users and their needs, experiences and motivations as well as immersing oneself in the physical environment to get a personal understanding of issues involved. Empathizing is a crucial step of human centered design processes since it allows the designer to leave assumptions about the world aside and gain insight into other peoples' needs [17].

Coming to the project with a very limited understanding of the problem space, extensive user research was necessary. Hence, numerous data collection methods are proposed below.

4.1.1.1 Contextual Inquiry

Contextual inquiry derives from ethnography, as described earlier. It is a method used to understand the users in their real working context, more specifically their needs, desires and approach to the work [6]. The main point is to conduct the inquiry in the users' natural environment, where they feel comfortable and are in the right mindset, rather than asking them to describe the way they work and recall events completely detached to the context where they usually occur.

The role of the contextual interviewer is not only to ask questions to the users but also to observe them as they work and notice their actions as they unfold to understand their inherent motivations [41]. What a user say they do and what they actually do are not necessarily the same thing. By both observing and asking questions, the risk for gaps in the interviewer's understanding is being lowered [8].

While observing users, it is, however, important to try to avoid the *Hawthorne effect* – the effect of observed users behaving differently because they are being observed [38]. In order to prevent this, it is important for the research duo to stress the overall goal of improving the situation for the users being observed, and how it is not, in any way, connected to performance reviews.

4.1.1.2 Interviews

Based on the office set-up at Spotify, with open landscapes and project managers and engineering managers sitting adjacent to the squad, more traditional interviews, i.e. interviews with a location different from the exact working spot, will have to be held too. Mainly to not disturb the rest of the squad working but also to prevent the interviewee from being dishonest because their manager might be overhearing the conversation.

The interviews will be semi-structured in the sense that they will combine predefined questions with open-ended, follow-up questions that comes up during the interview based on the stories told by the interviewee. As Wilson puts it, semi-structured interviews can be suitable for exploration of tasks and flows [84], an area of high

interest in the scope of this research. An interview guide outlining the areas of investigation and predefined questions will be formulated to act as the skeleton for the interviews.

Depending on how far into the empathize phase a project is in, the focus of the interviews tend to differ slightly. Especially if the problem at hand is a wicked problem with an ill-defined problem space. Cooper discusses three distinct phases, namely early interviews, middle interviews and late interviews [13].

- **Early interviews** are exploratory and the goal is to get a better understanding of the domain and problem from the user's point of view. Therefore, questions tend to be more open-ended.
- **Middle interviews** take place when the researchers are starting to see patterns and want to get richer data to connect the nodes.
- **Late interviews** involves more closed-ended questions to confirm previously observed patterns and to tie up any potential loose ends.

4.1.1.3 Focus groups

Another qualitative method for data collection that will support the contextual inquiries and interviews with single individuals are focus groups. A focus group is a type of group interview with several participants lead by a moderator, commonly used in research as well as participator design. They are used to study feelings, opinions and attitudes towards a product, service or brand [37]. One of the biggest strengths of focus groups lies in the group dynamic it induces. When guided properly, the participants will accept each other as peers and together share stories, perceptions and fantasies [37].

4.1.1.4 Questionnaire

Questionnaires, or surveys, are used for collecting self-reported information and data about people's thoughts, opinions or perceptions, typically in a written form or through Likert scale questions. By utilizing Likert scale questions, the participant will have the option to scale their response along a continuum, therefore indicating how strongly they agree or disagree towards a statement [37].

4.1.2 Define

The second stage of the Design thinking process is the *define* phase. This is when the data gathered from the user research is accumulated and analysed in order to define the core of the problem. The aim is to find a meaningful and actionable problem statement that will guide the ideation process in the third stage [27]. A great definition of the problem statement brings clarity to the table, to avoid having to work like a person stumbling in the dark of the design space, as Rikke Friis Dam & To Yu Siang puts it [27].

Below, the methods used to analyse the collected data are described.

4.1.2.1 Affinity diagramming

Affinity diagramming is a method used to externalize and cluster insights and observations from the user research [37]. Instead of having tacit knowledge stored in people's minds or hidden in interview transcripts, affinity diagrams allow for researchers to synthesize what has been captured in terms of research-backed insights, concerns, observations or requirements by putting it on individual sticky notes. Each insight can therefore be considered as a design implication on its own [37]. By clustering the sticky notes based on themes or affinity it is possible to see emerging trends or patterns in the data. Hanington & Martin [37] further emphasize that affinity diagramming is an inductive exercise; instead of grouping the notes based on predefined themes, the work is done in a bottom up manner by letting the themes emerge as notes of similar intent or issues are clustered together.

4.1.2.2 Thematic Analysis Coding

Thematic Analysis coding is a qualitative data analysis method for identifying, analysing and reporting patterns or themes within data [9]. Thematic analysis is characterized by the search for themes that emerge as being important to the research question or phenomenon studied. The identification of themes transpires through a process of carefully reading and re-reading the data and is a form of manual pattern recognition within the data where emerging themes become the foundation for further analysis [23].

The steps of Thematic Analysis Coding are as follows:

1. **Familiarization with the data:** Transcribing and reading the data, noting down initial ideas of codes that can describe the content. Familiarization with the data is about consuming and submerging oneself in the findings.

2. **Generating initial codes:** Systematically coding initial features of the data across relevant parts of the data set, collating data relevant to each code. A code is a brief description, not an interpretation, and is a way to meaningfully organize the data into groups. A section of the data can inherit multiple codes.

3. **Searching for themes:** Distill codes into potential themes, gathering all data relevant to each potential theme. Themes are broader than codes and involve active interpretation of the codes as well as the data. This is an iterative process where codes are moved around to form a multitude of different themes.

4. **Reviewing themes:** Checking if the generated themes work in relation to the codes. The data is read through once more to make sure that they are adequately reflected by the themes. If themes contain code contradictions or become too broad, one should consider splitting the themes or moving the codes to find a better fit. This too is an iterative process where the researcher goes through the data, codes and themes again to make sure there are coherent themes representing the total data.

5. **Defining and naming themes:** Constant ongoing analysis to refine the specifications of each theme, successively generating coherent definitions and names for each theme. The theme should not be solely descriptive but also define the essence of each theme and why it is interesting. [9][54]

4.1.2.3 Point of View

Point of views (POV), also called user need statements, are actionable problem statements outlining clearly who a particular user is, that user's need and why that need is of importance to that user. A POV should define explicitly what it is someone is trying to solve in order to encapsulate and better communicate the designers perspective on the problem. A POV can also be used as a measuring for success throughout the design thinking process. A well written POV statement should entail *what* a design is attempting to solve, not *how* [31].

A POV is constructed out of three specific parts, *A user, a need* and *an insight*. These three parts are then combined into the following pattern;

> **A** [*user*], **needs** [*need*], **because** [*insight*].

An example POV could look something like this:

> A developer at Ericsson [*user*], needs a faster way to communicate with his/her colleagues [*need*], because his/her way of working requires a speedy way to ask questions and receive answers and their current communication path of using emails can not guarantee fast answers [*insight*]. [25]

4.1.2.4 How Might We questions

How Might We (HMW) questions are a design thinking method created to open up for ideas. It turns the problem statement into opportunities for design and suggests that there is a solution to the problem. If framed correctly, a HMW question should offer the possibility to be answered in a variety of ways. It should not propose a specific solution but set the stage for creative and innovative thinking [43].

As an example, if the problem statement was the same as the example in the Point Of View section above, see 4.1.2.3, the HMW question may be written as:

- How might we accommodate for a faster email response?
- How might we make the lessen the need of a fast response?
- How might we make it easier to find answers to questions without having to engage a colleague?

4.1.3 Ideate

Having a clear understanding of the problem, a solid knowledge about the users and their motivations and goals, it is time to enter the third phase of design thinking process. This is when the designers are ready to start forming ideas. Hartson & Pyla [38] stresses that this is not only supposed to be a fast and collaborative process, but ideation also calls for extensive iteration.

The planned ideation methods are as follows:

4.1.3.1 Sketching

Sketching is the visual medium used for creation of preliminary design ideas. It is a rapid process where freehand drawings are used to explore the design space and the potential solutions to the problems at hand. Focus is on concepts, not details [38]. Expressing ideas with simple drawings rather than sentences has several strengths; it is quicker, it can help designers think more openly and creatively, and visuals provokes further ideas [26]. Sketches are also easily shared and good fuel for discussions. It is important to note that a sketch is not just a way to document a designer's thoughts, it is way of thinking with pen and paper, *"designers invent while sketching"* as Hartson & Pyla puts it.

4.1.3.2 Brainstorming

Brainstorming originates from Alex Osborn and his famous brand agency, BBDO, and dates back to the 1930s [60]. The technique has traditionally been used as a group activity to generate a large a amount of ideas; quantity over quality. By leveraging the synergy of the group and without having to worry about judgment and criticism, previous ideas can inspire the creation of new ones [26]. *Welcome oddity* is another widely accepted rule of brainstorming that aims to create a safe forum where the creativity can flow freely [37].

4.1.3.3 Crazy 8's

Crazy 8's is a fast sketching exercise that challenges participants to sketch eight distinct ideas a specific time frame, usually a minute each. The goal of the method is to get past the inner criticising voice trying to evaluate the usefulness of ones idea and instead allowing creative impulses to flourish. This method builds on the idea that weird, impractical or impossible ideas can lead the way to truly inspired and innovative ones [33].

4.1.3.4 6-3-5 Brainwriting

6-3-5 Brainwriting has gotten its name from the setup of the method itself. 6 participants are encouraged to come up with 3 ideas each during a 5 minute period. When time is up, each participant switches his or her ideas for the ideas of someone else and ideates on top of them. By taking part of someone else's creations and building upon them the participants both stimulate the creative process and also

get a feeling of understanding and partial ownership of all the ideas being created. This switching of ideas continue until everyone has built upon each other's ideas, creating a total of 108 ideas during 30 minutes [51].

4.1.3.5 Dot voting

Dot voting, also called democratic voting, is a useful facilitation tool to help a group vote and narrow down ideas or other items. The group members each receive a fix amount of dots that they use as vote tokens to decide what idea has most potential. It is up to the facilitator to decide weather some specific selection criteria is in play, such as if one member is allowed to put more than one dot on one idea or if one can vote for his or her own idea [79].

4.1.3.6 Four Categories Method

The four categories method is used to divide ideas based on their level of abstraction, ranging from the most rational to the craziest idea [18]. Each idea is meant to be placed in one of the four categories: the rational choice, the most likely to delight, the darling and the long shot. By having ideas in each bucket, the designers make sure to cover all grounds and not only create simple and shallow ideas but also innovate and challenging ones.

4.1.3.7 Design Critique

A design critique refers to a meeting or group conversation where a design is analysed and feedback is provided regarding whether it meets its objectives or not. The idea is to analyze and provide feedback on the design itself and its underlying assumptions to ultimately improve it. Providing feedback and asking questions regarding a piece of work helps designers avoid mistakes, create ideas and get a fresh perspective [30].

4.1.4 Prototype

The fourth phase of the Design thinking process is all about creating solutions to the problems identified and letting your ideas come to life. It is an experimental stage where the design team produce cheap, scaled-down versions to test the core of the solutions to investigate if they address the problems sufficiently [17]. This physical and, oftentimes, tangible realization of product or interface concepts is critical to the design process since it enables testing with clients and potential users [37]. Rikke Friis Dam & Teo Yu Siang emphasize that after this phase, the aim is to have a better understanding of how real users would think, feel and behave while interacting with the end product.

Prototypes are categorized by their level of fidelity, ranging from low to high with with a continuum of variations in between [37]. The different kinds of prototype variations are described in more detail below.

4.1.4.1 Low-fidelity prototyping

Low-fidelity prototypes usually take the form of sketches or simple storyboards. In interface and software design, a common way of doing it is through paper prototyping where screens are sketched on pieces of paper [37]. Low-fidelity prototypes are ideal for early testing when details have not been finalized and things are likely to change. By having limited the efforts put into creating the prototype, the threshold of discarding an idea remains low [38]. Low-fidelity prototypes also feed nicely into the ideation stage where there is freedom to explore and generate new ideas and potential solutions [16]. It is, arguably, easier for peers to critique and give feedback to a prototype that is in an early phase. However, a bi-product that follows is that users might find it hard to take the prototype seriously. Being clear about its purpose is, hence, of great importance.

4.1.4.2 Medium-fidelity prototyping

Whilst a medium-fidelity prototype has limited functionality, it still has clickable areas which present an application's interactions and navigation possibilities. Compared to a low-fidelity prototype, prototypes at a medium-fidelity state can give user a better sense of what the final solution might look and feel like. They're great for refining the execution of solutions whilst still being cheap enough to provide room for complete changes in the direction of the solution [16].

4.1.4.3 High-fidelity prototyping

Compared to low-fidelity prototypes, high-fidelity prototypes are much more detailed representations of the design solution, i.e. they bear a closer resemblance in look and feel to the finished product [37]. They are used to refine and decide the final design decisions before they go into implementation. Despite being more expensive to invest in time wise, it is important to note they are still less expensive and faster than coding or producing the final product [38].

4.1.5 Test

The final step of the Design thinking cycle is the testing phase. Being part of an iterative process, the insights from rigorous tests are often used to redefine or add to the problem statements. During the testing phase, the designer learns more about the problem itself and the user experiencing the problem, which adds to the empathy towards the user that might alter the definition of the problem [17]. This may furthermore have an implication to previous design choices. Alterations and refinements are, hence, also done at this stage to rule out insufficient solutions [17].

The main method used for the last phase will be usability testing, which is explained below.

4.1.5.1 Usability testing

Usability testing is used to assess a product's usability. According to ISO, the definition of usability is the *"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."* [2]. The point is to validate the design choices and identify if part of the interactions are found to be frustrating or confusing, typically by letting a test subject go through the steps of a given task or set of tasks [37]. Scenarios, such as read aloud stories or video clips are often used to contextualize the tasks and put the test subject in the right mindset. It is not uncommon to use a Think-aloud protocol during a usability test, asking the test subject to verbally speak out their thoughts, in order to better understand the reasoning and conflicts in mental models while going through the tasks [37].

4.1.6 Tools

During the thesis project, a multitude of tools will be used to assist in the user research, facilitation of focus groups and workshops, production of prototypes and the hosting of usability tests. These tools are briefly described below.

4.1.6.1 Airtable

Airtable is a spreadsheet-database hybrid, mixing the features of a database with the format of a spreadsheet. Much like other spreadsheet tools such as Excel or Google Sheets, any type of data can be added to the document but the data can also be linked together, like the functionality of a database [67].

4.1.6.2 Figma

Figma is a cloud based design and prototyping tool. Much like other prototyping tools, it lets its user create vector-based digital mock-ups and wireframes of websites, apps or other user interface components. One of its core features is the ability to work collaboratively on the same project in real time, allowing for crowd-sourced idea generation and creation as well as a simple way of sharing and showcasing your project to stakeholders and users. Figma also pockets simple, yet powerful, prototyping capabilities, allowing for the creation of interactive high-fidelity prototypes with both the look and feel of a real product [14].

4.1.6.3 Lookback

Lookback is a user testing platform tool that act as a single solution for running remote user tests. The tool not only allow researchers to set up a test environment with a link to the prototype, but it also sets up a video call between the participant and the moderator and, with the participant's consent, automatically share their screen. Once running a test, the moderator can invite team mates to observe the test in real time. Furthermore, the tool allows any team member to add notes and comments that are time stamped to the exact moment they were added. Afterwords,

the test is packaged for the team to share or re-watch with all notes and comments available [50].

4.1.6.4 Mural

Mural is a digital work-space for visual communication and collaboration. Making use of an giant digital, shared whiteboard, users and invited participants can create digital post-it notes, symbols and text boxes to collectively ideate, communicate and create [55].

4.1.6.5 Reduct

Reduct offer a transcription service for audio or video files. A user sends in their file and within eight hours, a transcriber from Reduct's end transcribes the file with time stamps and sends it back to the user [1].

5

Process & Execution

In this chapter, the process of this Master's thesis project is explained in chronological order. The project followed an iterative design thinking approach, described in detail in section 4.1, in order to answer the research question and to extract the user experience factors.

The initial entry point for the project was a wicked problem, see 3.1, that involved looking into blockers of productivity in general. The first two steps of the design thinking model, namely *Empathize* and *Define*, therefore had to be iterated twice in order to find a more suitable scope of a problem to address. After the first iteration, the problem of discoverability surfaced as one of the major sub-problems of productivity and was hence, in close discussions with stakeholders, decided to move forward with. The second iteration therefore involved mapping out the full problem space of discoverability whilst in parallel letting the user experience factors be constructed and formulated. These factors were furthermore manifested, user tested and evaluated in a medium fidelity prototype, before making their way to the final list, presented under chapter 6. After the testing, some final changes were made to the concept before transferring it to a high fidelity prototype. A graphical representation of the project's phases can be seen in the figure below, see 5.1.

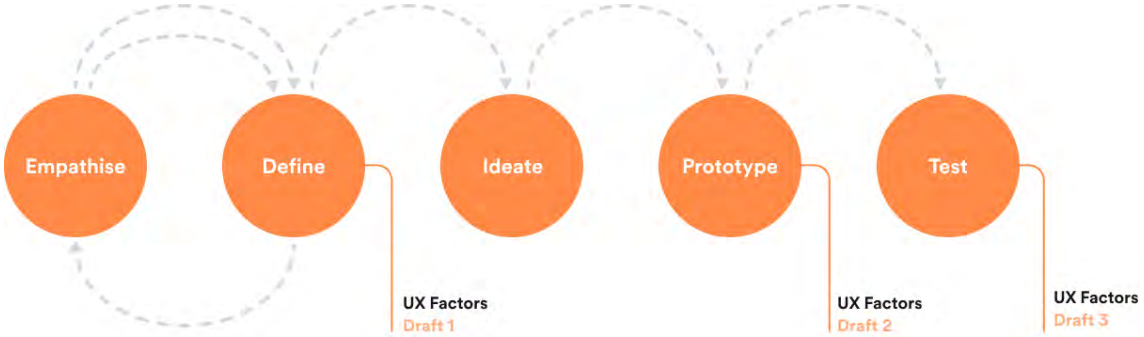


Figure 5.1: A graphical representation of the thesis project's phases.

5.1 Project Preparation

The first step of the thesis was to plan out all the phases of the project. To get a better first understanding of the problem and the activities to be undertaken, interviews with internal and external stakeholders were conducted during the first weeks.

5.1.1 Planning

This thesis project was carried out over 20 weeks during the spring of 2020. Over the course of this period, supervisions with the academic supervisor were hosted on a weekly basis to check in on the project and to discuss any potential hassles hindering the momentum of progress. Similarly, supervisions with the project mentor from Spotify were scheduled for each week to ensure all the necessary support from the company side to move forward as planned.

The plan was to conduct at least one full design thinking cycle with its five inherent phases; define, empathise, ideate, prototype, and test, with allowance for iterations within each step. The user experience factors were supposed to start evolving during the first empathise phase and then be updated and iterated as more knowledge and understanding of the solution space was gained. Alongside the project, the final thesis report had to be written continuously. Originally, the idea was to spend one full day a week dedicated to report writing and have a buffer in the end to allow for larger portions of the report to be written once every phase of the design thinking cycle had been touched upon. An simplified overview of the plan for the project can be seen in figure 5.2 below, and a detailed Gantt chart can be found in the appendix, see H.

Figure 5.2: The simplified overview of the plan for the project.

5.1.2 Interviews with Internal and External Stakeholders

To understand the topic of productivity within the Spotify context as well as to get a hold of tacit knowledge regarding the organisation's previous work within productivity, three semi-structured interviews with internal, as well as external, stakeholders were conducted. The internal interviews were held with one tribe lead and two product managers. The external contacts consisted of an interview with two academic researchers who were running research on team productivity at Spotify and other software companies in parallel with this project.

Questions that were asked to the internal stakeholders included questions such as:

- How would you define productivity and how is it measured across the organization today?
- Why is this of an interest to Spotify at large?
- What is your personal picture and understanding of the problem?
- What topics have you heard from others when it comes to productivity problems within the organization?
- To your knowledge, what has been done in the past to tackle potential problems?
- Who else would you recommend us to talk to?

The interview with the external productivity researchers included questions such as:

- How have you previously measured productivity and what are your recommendations when it comes to understanding it at a software company?
- In terms or research, what are the methods you have used to gather data?
- When conducting research within Spotify, what are best practice when it comes to set up and conduct interviews, focus groups and workshops?
- What other literature would you recommend on the topic?

5.1.3 Takeaways

From interviewing both internal and external stakeholders, a number of considerations were brought forward regarding the direction of the project. What previously has been done within the context of productivity was also discussed, as well as thoughts and recommendations on potential productivity blockers to start look into.

From the interviews, it was also made clear that productivity amongst developers can not be equated to their time coding in front of a computer screen, in complete agreement with the academic understanding, see 2.1.3. Planning, aligning with team mates, discussing the road map, and teaching new joiners the ropes are all part of making sure the company can be as productive as possible, even if it might be perceived as productivity blockers on an individual level.

The interviews with the external researchers provided insights on ongoing productivity research within the space of software technology companies as well as on how they have conducted research at Spotify in the past. They advised to not try to create metrics or measurements for productivity or performance, since it is very difficult to capture it holistically, in agreement with the literature study, see 2.1.1. Instead, they recommended trying to look for and understanding how to enhance the capability of engineers to be useful as a proxy for productiveness.

5.2 Empathise – Productivity

To better understand the bottlenecks of productivity amongst developers at Spotify, the initial entry point to the project, there was a need for both a more theoretical understanding of productivity in general but also how the productivity problems were manifested in the context of Spotify. It was hence necessary to both digest related literature touching upon the subject, as well as to talk to numerous developers across Spotify to understand their perspectives.

5.2.1 Productivity Literature

To get a better picture of the current body of knowledge when it comes to developer productivity and what previously have been developed to aid in it, several research articles where digested. The articles generally encompassed theoretical knowledge on productivity amongst software development teams but also the many problems of trying to measure it, see 2.1.1 and 2.1.3. Internal documents on how Spotify is currently measuring and defining productivity was also read to understand the context and overall company mindset on these questions better, see 2.1.2.

5.2.2 Previous Company Research

Productivity has previously been investigated and researched at Spotify to some extent, albeit with a different take and arguably with more focus on the measurement part. One major survey addressing developer's perceived productivity blockers was sent out internally at the company back in 2018 and received answers from 348 different developers. Reading through those answers gave an initial understanding of the problem space and what could be expected to surface during the interviews with developers.

5.2.3 Interviews on Productivity

The foundation of the semi-structured interviews was the interview script, including a number of pre-constructed questions guiding the talk, yet still allowing for side tracks if needed. The interview script was established with inspiration from similar research that had been conducted at the company previously. Given the uncertainty of how the everyday work of a developer at the company looked like, questions regarding their regular duties were included to fill this gap too. The interview script was first tested in two separate pilot interviews with two developers from the team. Smaller adjustments were made, such as removing and re-phrasing questions before proceeding to talk to the target audience. The full script can be found in the appendix, see appendix A.

Ten semi-structured interviews with developers from different tribes were conducted to gain primary source insights to the productivity blockers. The setting was most of the times a meeting room in one of the two Stockholm office sites; Urban Escape or Birger Jarl, or at the Gothenburg office. Both project members were present during each interview; one acting as the interviewer asking questions and the other

taking notes. The interviews were also recorded using a portable audio recorder once having received consent from the interviewee. Questions asked during the first round of interviews included:

- Could you categorize the type of development work you do?
- What kind of issues do you typically run into?
- Can you tell me about an incident that happened lately when you got really frustrated with a tool?
- On a scale from 1-10, how streamlined would you say that Spotify's development processes are?

The main selection criteria for the interviewees was the need to be an individual contributor, i.e. a person writing code and not solely working with management questions. It was also important to cover different tenure and not only talking to senior developers. Finally, it was avoided to talk to developers working within the Platform mission of the company since these developers' main task is to develop infrastructure for other developers, not primarily to use them themselves.

5.2.4 Focus group on Productivity

In order to create a more open setting where several developers could elaborate on each others ideas simultaneously, a 90 minute long focus group, see 4.1.1.3, was hosted at the Urban Escape office site in Stockholm. Four developers, with tenure ranging from four months to two and a half years, all coming from different squads were invited. The work was divided during the focus group and split into two roles; one moderator, with the main responsibility to guide the discussions, and one note taker, making sure all answers and opinions got documented. The focus group started with a small warm-up exercise to get everyone comfortable talking before moving on to the questions regarding productivity. These were some of the prompts:

- On post-it notes, write down what makes you productive.
- On post-it notes, write down the biggest blockers that hinders your squad from being able to move faster in your day-to-day work.
- Please take a minute to sort your productivity blockers. Put them in one of three categories; minor, medium or major.
- Now, let us sort these blockers together on the white board.

Between each prompt, the group members were encouraged to share their notes and to discuss each other's answers. The moderator distributed the word and made sure everyone got the chance to have their say.

5.2.5 Takeaways

From the literature review of external as well as internal research, it became evident that a reliable metric for productivity would be difficult to generate. Instead it was decided to move forward with the internal definition of productivity, see 2.1.2, and focus the design efforts on enhancing productivity by eliminating issues related to time sinks and thereby generating more time for the developers that can be used for productive purposes. It also became evident that Spotify already knew of multiple productivity blockers withing the company and that measure had already been taken to alleviate some of those. These problems, identified during an internal productivity survey, ranged from topics such as *Time wasted as a result of low quality infrastructure, Slow and/or low quality build-test-deploy process, Unclear ownership, Non standard tooling or infrastructure, Inability to discover information or get help* and more.

From the interviews, it became evident that the answers differed slightly depending on seniority. The problems recent hires had were in general different than the ones from more tenure developers. Perhaps since they had arrived with a pair of fresh eyes compared to someone who had been working for years, gradually adapting to the problems.

Since Spotify is operating from offices located in different parts of the world it was expressed as important, from a company perspective, to not only talk to developers from a single country and therefore risk finding a problem specific to that certain location. The first round of interviews were conducted with developers in Sweden, moving forward, it was decided to involve talking to developers from at least one of the other two major office sites in New York and London.

Whilst semi-structured interviews made it possible to dig deeper into a single individual's view of productivity, the focus group made it possible to gather four people's opinions on the manner at the same time. Also, the developers seemed to positively affect each other in the sense that they could elaborate on each others problems and add yet another perspective. The magnitude of the problems were also easier to determine once the group had discussed them collaboratively and together placed them in one of the three severity buckets.

Some of the key insights perceived as worth moving forward with were:

- Focus on more junior developers since Spotify is in blitz-scaling and this group is the fastest growing within the company.
- Make sure to also cover developers at Spotify located outside of Sweden.
- Utilize focus groups to a larger extent since it was perceived as a fruitful data collection method.

5.3 Define – Productivity

Once the first round of data was collected, it had to be analyzed to be able to draw any conclusions of the problem space. The purpose was to derive emerging problem themes and then present them for a few stakeholders in order to together decide what to hone in on and move forward with.

5.3.1 Transcribing

After having manually transcribed the first couple of interviews, the labour proved way more time consuming of an activity than planned. Considering the amount of interviews planned ahead, alternative solutions were investigated by the help of other researchers at the company. A service called Reduct was brought forward as a good candidate, see 4.1.6.5. Reduct allows the user to upload an audio or video file and then have it professionally transcribed by a Reduct employee. Since Spotify was already a consumer of the service, it was decided to make use of the tool for this thesis.

5.3.2 Affinity diagramming

The quotes that were extracted from the ten transcripts were placed inside a digital affinity diagram, see 4.1.2.1 hosted in Mural, see 4.1.6.4. The notes containing quotes were color coded based on the source, i.e. the interviewee, in order to quickly be able to distinguish between them. The aim of the affinity diagram was to unfold themes in terms of problem areas by going through each quote, one by one, and discuss what implicitly was being described in it. When a note matched what had previously been said, those notes were placed adjacent to each other. Eventually, clusters started to emerge and once every note had been processed, there were only a handful of notes missing a cluster.

The second step was to make larger problem areas of several clusters that were similar to each other. That way, the affinity diagram ended up with seven larger themes, see figure 5.3. At this stage, the color code could also be utilized to tell if a topic was raised by different individuals or mainly consisted by one or a couple of loud voices, which further would be considered when deciding what to move forward with.

5.3.3 Stakeholder presentation

After having analyzed the problem areas formed in the affinity diagram, compared them with what had been said during the focus group and what had surfaced in the productivity survey from 2018, a presentation was held for the project's two closest stakeholders at Spotify. The purpose of the presentation was to present the initial findings and, more importantly, to decide which problem area to move forward with. This was considered necessary to do in close collaboration with stakeholders of the company in order to align with existing initiatives or initiatives projected to be carried out in the near future.

Figure 5.3: Figure of first affinity diagram on productivity problems.

There were two problem areas that seemingly overlapped in all three sources, namely the problems of *discoverability* and the problem of *decision making*. These two problem areas were presented in more detail together with actual quotes from the first hand sources to back them up. The former, discoverability, touched upon the perceived lack of an overview of what previously have been built that could be utilized. As one developer put it:

> *"In general, finding out if someone has done whatever I want to do is difficult"*

But also different aspects of documentation and how finding good, reliable and up-to-date documents is a struggle today. As another interviewee said:

> *"You don't know where the documentation is, and you have to go searching for it. Some people put it in TechDocs, some put it in README files in the repository. Other people will have Wiki pages..."*

The latter, decision making, arguably had more to do with management and how to properly align squads and prioritise tasks. A developer expressed it as:

> *"Distributed teams working with other teams that have different priorities – these are the killers and productivity risks"*

Another individual contributor raised the concern of the time distribution as a con-

sequence of having to coordinate with stakeholders and understand what should be done:

> *"10% - 20% coding and the rest is spent trying to figure out what we're doing"*

5.3.4 Takeaways

After the stakeholder presentation, it was decided to move ahead with the problem of *discoverability* within the Spotify ecosystem. Mainly since this problem had a tight connection to tooling and infrastructure, an area closer to the heart of the stakeholders as well as the knowledge space of the thesis duo. Discoverability was one of the most discussed topics from the interviews and focus group as well as from previous research within the company. In the daily work of developers, finding documentation, example code and other reusables seemed to be an integrated part of their development process. Elevating these procedures and especially making them faster was perceived as a way to remove time sinks and provide developers with additional time for focus and productivity.

5.4 Empathise – Discoverability

The original scope of investigating blockers of productivity in general had led to a broad spectrum of different problem areas, it was therefore necessary to take a step back and collect more data to fully grasp the new scope. To understand the topic of discoverability and, more importantly, what productivity blockers are generated by poor discoverability, a set of the original research methods, albeit tweaked to fit the new lens, were re-visited to derive a deep understanding and empathy of the users' perceptions.

Discoverability in the aspect of software developers at Spotify relates to the discovery and interpretability of any type of information that might be of need. For example existing reusable components such as libraries or SDKs, example code, previous company projects, technical documentation, historical decisions, organizational structure, ownership etc.

5.4.1 Discoverability Literature

To get of the ground with this new scope, academical articles on the problems of discoverability, especially in the context of code, was read and analyzed to get a first understanding of the problem, see 2.2. Many of the articles were proposed by developers at Spotify, heeding the duo's call of trying to find articles within the realm of discovery in software development. Previous discoverability solutions were also identified within academic literature to work as inspiration in upcoming phases, see 2.2.3.

5.4.2 Interviews on Discoverability

Building upon the insights from the productivity interviews, see 5.2.3, a semi-structured interview script regarding discoverability issues and possible ways to bypass them was constructed. The new interview script was run through a pilot to test out the questions and possible correlating answers. The test indicated a few needed changes which were all carried out before the rest of the interviews took place. To see the full interview script from the discoverability interview, see appendix B.

Seven semi-structured interviews with developers from different tribes, software disciplines and tenure were conducted to get an understanding of discoverability issues within the company. Questions asked during this round of interviews included:

- How does your squad make sure you don't build something that has already been built before?
- When you are looking to find something another squad or employee has built, how do you go about doing that?
- How do you make sure other squads could utilize what you have already built?

The interviewees were selected in a manner to achieve a diversity of development disciplines such as web, mobile, back-end, machine learning and data engineering. Since the previous interview session, see 5.2.3, had yielded great insights when talking to staff with tenure less than two years, a majority of interview participants were selected based on when they were on-boarded at the company. As was the case with the productivity interviews, talking to staff within the Platform mission was avoided due to them not being the intended target user group.

5.4.3 Focus groups on Discoverabilty

Due to previous success and acquired insights from the previous focus group, see 5.2.4, focus groups were facilitated with the focus on discoverability as well. As with the interviews, see 5.2.3 and 5.4.2 a mix of developers with different programming disciplines, tenure and organizational belonging were selected. Learning from previous mistakes, two extra participants were invited to cater for potential late dropouts.

A workshop guide was created together with a slide deck that would work as a guide throughout the focus group. A warm-up exercise was used to set a creative, safe atmosphere, open for sharing. The participants were then guided through a set of questions to help them share their experiences and ideas. Some of the questions and prompts used in this focus group were:

- If you think about discoverability in general at Spotify, what are the things that come to your mind?
- What are the things that aids you with the discoverability of things at Spotify? Write down topics on post it notes. Write down as many things you can come

up with. It does not matter if they are big or small.

  - Discuss this

- On post-it notes, write down topics on how discoverability at Spotify hinders you personally from being able to move fast in your day-to-day work.
  - Together, as a group sort the discoverability blockers from big to small on the wall.

By the end of each focus group, the participants were asked to vote on a problem touched upon during the session that in their own opinion was the most important to address. The problem with the most votes was used as a backdrop for a small ideation session. The group was asked to together elaborate and brainstorm, see 4.1.3.2, on potential solutions to address this problem.

The focus groups on discoverability were held a total of four times. Once physically with a group of four developers in New York where a big meeting room was used for 90 minutes to host the session. The additional three were facilitated remotely using the video conference tool Google Hangouts, see 2.4.8 and the Mural app, see 4.1.6.4, as a collective, digital whiteboard to remotely create and share post-its and text objects, see figure 5.4. Out of these remote workshops, one was held with four developers from the Gothenburg office and the remaining two with a total of six developers from the office sites in Stockholm.

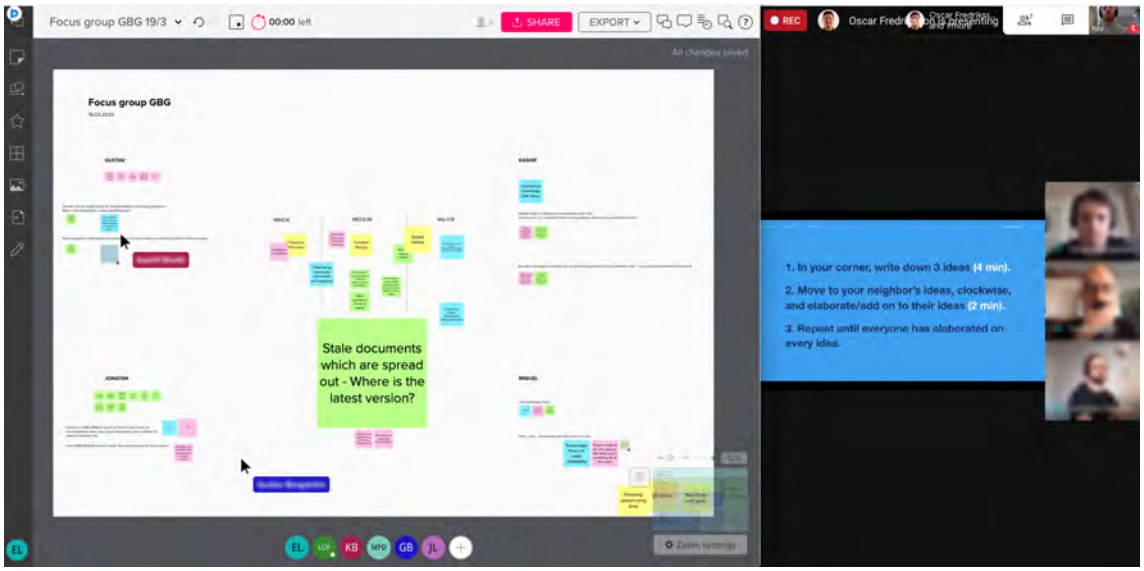


Figure 5.4: Remote focus group facilitated through Google Hangouts and Mural.

During these remote focus groups, a segment of ideation was added to see what kind of solutions could be formed around one of the discovearbility blockers. After the discoverability blockers sorting, the participants were asked to part take in a quick dot voting session, see 4.1.3.5, where they received two votes each for which problem they deemed most important to solve. After voting and deciding for a problem to solve for, the participants engaged in a short 6-3-5 brainwriting session, see 4.1.3.4. The ideas were then discussed to pick up intrinsic motivations from the developers

and get a better understanding to what situates a good or valuable idea for the developer clientele.

5.4.4 Guerilla Research

To evaluate the insights from the in depth interviews as well as gain additional insights on the topic of discoverability within the Spotify technical ecosystem, a set of quick and dirty unstructured interviews were conducted within the New York office space. These were performed by entering the work space of developer teams and asking anyone who had a few minutes to spare to answer the question:

"What are your thoughts on discoverability within Spotify"

In total, eight developers were interviewed this way. Some participants engaged deeply and involved co-workers to answer the question whilst others were less outspoken. At most, a group of four developers created what could be considered as a mini focus group and shared notes on problems and possibilities. Since the question was asked within the office space in a contextual enquiry manner, see 4.1.1.1, participants could showcase their issues by demonstrating them on their computer screens, see figure 5.5, something that was not done during the interviews.

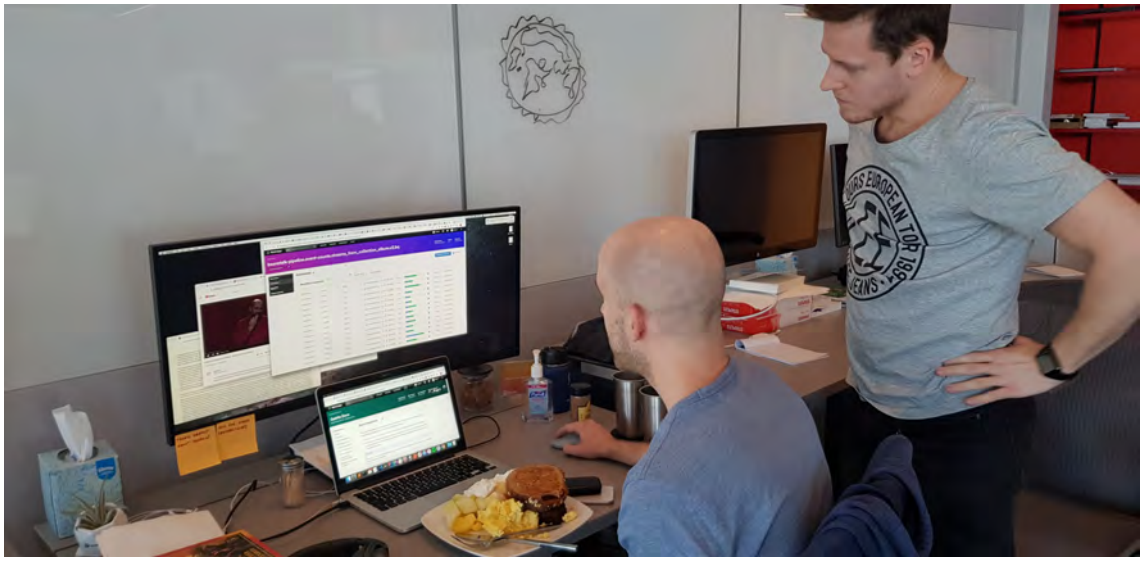


Figure 5.5: Guerilla research

5.4.5 Takeaways

The literature study proved valuable in better understanding the academic perspective on discoverability. Especially the concept presented by Shanmugasundaram et al regarding discoverability being divided between *discovery* and *interpretability*. This will be used to mentally structure the concepts within discoverability going forward, as well as a means to categorize the UXFs.

In comparison to the interviews conducted during the productivity research, the

topic of discoverability was rather difficult to explain at times and there was a constant struggle of keeping the interviewees on topic. Since the subjects covered within the topic was quite fragmented, the interviews turned out in a similar fashion. However, as was the case with the productivity focus group, the focus groups that ran during the discoverability studies provided great results. Putting more than three developers together to collectively elaborate on discoverability provided a nice way to understand the context of the problem spaces since the developers seemed eager to explain their issues to one another. By using each other's arguments as stepping stones, the participants could decorate their thought even further and arrive at more detailed conclusions. Even though the main body of the focus group work proved useful, the ideation sessions by the end of the remote focus groups did not. Since the developers voted for what problem to solve themselves they tended to go with a broad and obvious one which resulted in shallow solutions at best. Another reason to the low quality ideation session could have been the lack of time to ideate or an inadequate introduction to ideation and collaborative creation.

The main insights from this phase worth mentioning was:

- Focus groups provide a great way of quickly getting to the point of specific problems since the participants can leverage each others' answers. Therefor, next time, it is proposed to run a couple of interviews to get started, place a focus group in the middle and then finish off with a few interviews to dig deeper into the areas that never became clear enough.
- Make sure to always plan enought time for ideation. If engaging, having those extra 20-30 minutes can pay by having an interesting discussion afterwards or running a second creative session.

5.5 Define – Discoverability

To make sense of the vast amount of data collected, a thematic analysis was carried out with the purpose of deriving behavioural as well as problem themes to lay as a foundation for a solution concept. By the end of this phase, one specific problem area would be derived as the main problem to solve for in the upcoming ideation phase.

5.5.1 Thematic Analysis Coding

To analyze the data gathered from the discoverability interviews and focus groups, a thematic analysis coding approach was used, see 4.1.2.2. The cloud based collaborative spreadsheet tool Airtable was utilized for the process due to its advanced data-linking capabilities, see 4.1.6.1.

Although thematic analysis coding is described as a somewhat linear, step-by-step procedure, the analysis was carried out in an iterative and reflexive manner. 342 important and interesting citation blocks from the interviews as well as the focus groups were added to Airtable. Multiple codes were derived out of each of the

citations to capture their individual richness and contexts. However, after labeling each citation once, the codes proved too generic and unspecific to derive value from. Therefor, a second round of coding was inaugurated with the intention to generate actively interpreted codes directly from the citations.

To accommodate for a speedy process, a *code book* was generated by having four interviews being decoded twice, once each by each member of the thesis duo in isolation. Each citation and its codes were then discussed in detail to create a common understanding for the codes and their awarded connotation. Having generated a shared canon, i.e. the code book, the citations where split between the thesis duo who coded them in accordance to the code book. For every new kind of code that came about during this phase, a discussion was held regarding its significance before being added to the code book. Each individual coding segment added to each citation was peer reviewed and discussed before being added to the finished list of codes, see figure 5.6. 133 codes were generated from the 342 citations, where a plenitude of the citations were awarded with multiple codes. The full code book and amount of related citations can be fund under appendix C.

Figure 5.6: Thematic analysis coding example from two interviews and one focus group

5.5.2 Affinity diagram for codes

The codes were placed into an affinity diagram in order to extract overarching themes from the data. Codes were moved around and discussed one by one whilst also being traced back to the citations that created them to make sure the correct synthesis was derived. The codes were labeled either as behaviors, yellow, or problem areas, orange, depending on the citations they came from, see figure 5.7.

Some of the codes ended up in distinctively separate, stand alone themes, whilst some clusters grew big, with plenty of codes, that later had to be broken down into smaller themes for them to be fully understandable and actionable, see figure 5.7.

5.5.2.1 Discoverability themes

Eight overarching themes of discoverability problems and behaviors were found in the affinity diagramming process. These themes, and a brief description for each of them can be found below.

Figure 5.7: Themes from the thematic analysis coding.

1. **Relationships** has to do with the behaviors and problems related to making use of relationships within the organisation to alleviate the problem of discoverability. The term *Relationship driven development* was used by interviewees to describe how software is being developed by asking questions and getting assistance through one's network. In general, developers at Spotify make use of knowledgeable peers sitting on tacit knowledge, see 2.2.4, usually with a longer tenure, to find things within the ecosystem. Experienced colleagues can help out by pointing one in the right direction, by providing context, historical or technical, to a specific technology or issue or by surfacing less obvious technologies or services that can be used to solve a problem. On top of this, since time is of essence for a company like Spotify, asking a colleague usually renders faster answers than reading thought documentation. However, newly hired employees, in contrast, exclaimed unease at reaching out to their new found peers due to a fear of being perceived as incompetent, sometimes making them search for hours for the answers to their questions instead of asking for assistance.

2. **Documentation discoverability** encompasses the problem of finding and understanding the internal documentation at Spotify. Due to the use of many proprietary, internal systems at Spotify, well maintained documentation is of an essence. Developers complained about fragmented documentation, located at multiple places, and a lack of a single source or truth. Since documents are spread out, it is also difficult to remember where they were found when trying

to relocate something that one read in the past.

3. **Documentation content** was another theme that surfaced. Content were discussed as hard to maintain and difficult to keep up to date, leading to many developers not trusting or reading documentation at all. Documentation was also explained as hard to evaluate, leading to another level of uncertainty when trying to base decisions on them. Since documentation can also take on different formats, finding the right documentation for your specific case can be difficult. Developers pointed out that this has become a lot better since TechDocs was introduced to create a documentation standard and bring documentation closer to writing code, see 2.4.5.

4. **Historical information** referenced the problem of finding information of what has been done in the past and the reasoning behind the technology of today. The idea of having access to a road map, past and future, for certain technologies were brought up as a possible solution.

5. **Code exploration and evaluation** became the overarching theme for all problems related to finding reusables and other code components within the company. Developers expressed a feeling of uncanniness due to not knowing if they accidentally were re-inventing the wheel whilst developing certain features, and stories were told of squads that had done just that. For further explanation of this theme, see 5.5.4.

6. **Re-organizations** referred to problems and behaviors surfacing due to constant re-organizations at the company. Tribes and squads changing names and ownership of technology makes it difficult at times to know who owns and maintains what. Since Spotify let's their squads name themselves in order to accommodate autonomy, this makes findability and understandability of what a team does challenging, and even impossible at times, creating time sinks and hindering productivity.

7. **Slack** became the theme connected to all specific behaviors created by the introduction and the use of the messaging platform Slack, see 2.4.1. Slack has become *the* entry point for information at Spotify where the developers go to search for information, ask questions and receive answers. Since all information that is written within public channels is searchable, it has become an ad-hoc knowledge database where error messages are pasted into the search bar and where developers try to find if someone has asked their particular questions before them. Slack is mainly built as a chat and messaging platform and is not optimally built as a knowledge database. Developers surfaced this as a problem when telling about how conversations are removed form the platform automatically after a couple of months to alleviate issues of over populating the platform with content.

8. **Communication and alignment** related to the problems described regarding communication and alignment between squads. Spotify have development

offices globally over multiple time zones and this can sometimes affect teams negatively when trying to align on what is to be done. Especially difficult is the communication between squads in America and the ones in Europe where a majority of work is happening at one office when the other is outside their office hours. This sometimes leads to confusion on what the other office has done during their previous day which sometimes takes days to resolve due to the time difference.

5.5.3 Discoverability theme evaluation

In order to choose a discoverability theme for the upcoming concept creation, an evaluation process was initiated.

5.5.3.1 Stakeholder discussion

Two stakeholder meetings were held where seven out of eight themes were presented and discussed. The meeting participants were one senior product manager, one associate product manager and one engineering manager with insight into the thesis project and its objectives. After having presented the discoverability themes in detail, a discussion was held regarding the potential of solving each theme and its effect on the company. The stakeholders provided commentary on what was being done across the company to alleviate some of the problems and provided their view of what might be most effective and possible to tackle from a tooling and service perspective.

5.5.3.2 Evaluation of themes

On request of the stakeholders, a table was formed to evaluate the themes on subjects such as expected impact and anticipated return of investment on each problem. The sorting of problem severity done by the participants of the previous discoverability focus groups, see 5.4.3, was also utilized as weights in the evaluation. Another aspect taken into account was the thesis duo's perception of product potential in the different areas, i.e. how suitable they would be to design tooling for. Hence, all the three pillars of design thinking; human desirability, economical viability and technical feasibility were addressed.

5.5.4 Code Exploration and Evaluation

Based on the discussions and evaluation, it was decided to move forward with, and furthermore address, the problem of *Code Exploration and Evaluation* – which not only ranked as one of the top candidates after the evaluation but also sparked a great interest amongst the stakeholders. The different sub-parts of this problem area, communicated to make the case stronger during the stakeholder presentation, were as follows:

Difficulty determining the best practice exemplar
When developers are trying to find code that potentially could be re-used or at least

serve as inspiration, some seem to have a hard time determining the best practice exemplar, i.e. the one that should serve as the source of truth and therefore be followed. When given a number of results, seemingly doing the same thing albeit with code written differently and perhaps utilizing different underlying technologies, the developers would have want some clearer guidance on what to utilize. Today, they all have their own ways of trying to evaluate the different options, exemplified in the quote below:

> *"What team do I trust to do this thing? Do I know anyone who can do this thing? Like: Oh, I know the guy who wrote that code. Or I know the girl who wrote that code – she is super good."*

Problematic to find example code
It seems as the problem with code not only has to do with determining the quality of it but also finding it in the first place. When a developer is looking to find useful examples of how a problem has been tackled or solved previously, it is easier said than done given the huge code stack of Spotify. This is also mirrored in previous research at other companies, see 2.2.1. As one developer expressed it:

> *"In general, finding out if someone has done whatever I want to do is difficult."*

Inadequate search tools
Although the developers have a few options to chose from when looking to search for code, e.g. Spotify Codesearch, see 2.4.6, and the native search function within GHE, see 2.4.4, the results are at times perceived as outdated or too many to have the time to sift through. You would also need to know exactly how to word the search since these systems are text string based and only look for the exact matching strings. In a case where a squad has named their library or service oddly, something quite common at Spotify, as expressed by several interviewees, it can be very tricky to find it:

> *"It's also hard to kinda figure out what exists and yes, you can search in Backstage or whatever, and if you're lucky they named it well and you can find it, but it's not easy, right."*

Missing a market place
Lastly, developers seem to miss a natural place find overall reusables. Not only that, they lacked a place to market and share their own hacks, i.e. more experimental code that can address niched use cases that will not suit the majority of the developers, yet be useful for the few it covers. Today, these are mainly shared through word of mouth, either in person, in email news letters or by using Slack, see 2.4.1. Given the size of the company, neither are scalable solutions. One of the developers in New York said:

> *"See, I don't know if they'd be able to find my tool, unless they're like, in there doing a Slack search, you know? Or they'd have to ask someone"*

5.5.5 Problem statements

To lay as much of a solid foundation as possible for the ideation phase, it was necessary to convert the problem area of code exploration and evaluation to actionable problem statements using Point Of Views (POVs), see 4.1.2.3. More specifically, three POVs were articulated. From these three problem statements, several How Might We questions were formulated based on the research insights to assist in, and spark, the ideation sessions ahead, see 4.1.2.4.

5.5.6 Takeaways

After the stakeholder discussion and evaluation, it was decided to move forward with and address the problem of *Code exploration and evaluation.*

The problem statement in terms of POVs and their corresponding HMW's that were created to assist in the upcoming ideation phase were as follows:

1. Developers at Spotify need to be able to more easily discover reusable code because it would prevent them from having to "reinvent the wheel".

    - *How might we* make the process of discovering reusables feel less time consuming?
    - *How might we* make it interesting to browse reusables?
    - *How might we* make use of experienced colleagues within the same vertical for discoverability?

2. Developers at Spotify need to be able to evaluate code because it would help them in their choice of which code to utilize.

    - *How might we* help ranking code options?
    - *How might we* set up a way to show the best practice exemplar?
    - *How might we* show which code snippet is used the most across the code stack?

3. Developers at Spotify need to be able to share their hacks because it could help others in their day to day work.

    - *How might we* provide an easy way to package developers' solutions?
    - *How might we* incentivize users to share their hacks?
    - *How might we* make it easier to share hacks?

5.5.7 User Experience Factors - Draft 1

Given the overall project goal of creating user experience factors (UXF's) for a discoverability system, and at this stage having analysed all the research data, a first draft of factors was formulated. This draft was generated from the discoverability themes in the affinity diagram. By going through each theme and their related codes, UXF's based on what a developer at Spotify would want and need, from a discoverability point of view, was formulated and refined during discussion. Since the UXFs were based on the discoverability themes, i.e the summarized findings from the interviews and focus groups, they reflect comprehensive specifications for discoverability for any system at large, taking into account the findings from the broad spectra of themes generated from the research.

Summarized below are the initial set of UXFs, where the developers are referred to as *users*:

1. Allow users to evaluate the quality of the items discovered

What? By one or several criterias, the design needs to incorporate ways for the user to be able to evaluate the quality, rigidness and usefulness of the item.
Why? Users seem to have a struggle evaluating different options, be it documentation, code or technology owners when attempting to choose between multiple options. From a discoverability aspect, a software enterprise system should assist in this.

Example quotes supporting this UXF:

> *"I'll be like "Look at her code and then copy stuff out of that." If there was some magic way and get her up to be like, "We have ranked results by recently seen"".*
>
> *"Would be good to know if the quality of code is good - but how does one determine that?"*
>
> *"Or if you look at code and you see it has not been updated in a year plus, then I would avoid it."*

2. Assist users in finding the "blessed" choice

What? Assist the user in finding the Spotify supported and recommended way of doing or using something.
Why? Today, users get limited guidance on what way of doing things are considered best practice. This might result in them finding and making use of deprecated, i.e. outdated, solutions or documents. As a result, the code stack may get infected and ruined by bad code and unnecessary time would have to be spent fixing problems, thereof affecting productivity negatively.

Example quotes supporting this UXF:

> *"If I wanna find some sort of example code maybe I stumbled upon one and I didn't even know there are two better ways to do it."*
>
> *"It's not very straight forward what the best practices are. Say three different features, I'll implemented something very similar... which one of these are considered the best practice?"*
>
> *"You're always wondering, "Oh, is this the right way to do it?" Should I try to explore some other approach?"*

3. Allow users to easily interpret the purpose of an item

What? Give the user the possibility to interpret the code, document or service in a graspable way without having to digest or engage in too much content.
Why? According to some users, understanding the raw code or reading through technical documentation can be a tedious and time consuming task. Accessing short information about an item can be enough to evaluate its use case and purpose. This saves time, lessens the cognitive load and is therefore important for productivity.

Example quotes supporting this UXF:

> *"...you could save me two hours of reading code in whatever framework you use by just- that doesn't even have to be a detailed documentation. But like an article generated list- these are our end points, you can send it this way. Right? That's so much easier."*
>
> *"Otherwise it's more like, "Yeah, I'll just dig into the code myself" and try to be like, "Alright, how do I do this?"*

4. Allow users to access corresponding information or documentation for a given item

What? In connection with the item discovered, offer the user the ability to find more information that can aid in their understanding.
Why? According to some users, reading good documentation for a code or service can be faster than reading the actual production code for understanding.

Example quotes supporting this UXF:

> *"But having documentation gives you a quick summary of what is out there or what it's supposed to be doing. And if it's up to date then it's quick. Like it's faster than having to sift through code or look through it and understand it"*
>
> *"It was a bit challenging for me when I joined the company that some things don't have documentation at all. And then you don't know how*

you can learn about that."

5. Allow for perceived effectiveness during the discovery process

What? When in a search journey, make sure the user feel as if they are narrowing in on their target and coming closer to the anticipated goal.
Why? Some users seem to lose their attention, hence abandoning their attempt at finding what they are looking for, when not feeling progression early on. Discontinuing the discoverability process early might lead to lost findings as well as a potential added work load.

Example quotes supporting this UXF:

"When I'm trying to get something done, I don't really have the attention span to look through a bunch of stuff"

"For me, I think it's really hard. If you find a good document in the beginning of your search, I will use that but otherwise I will turn to ask people instead because it's so much work to find the right document and know whether it's correct or not."

6. Make browsing of items possible

What? Allow for a browsable overview of the items that exists.
Why? Many search systems provided within the Spotify eco-system has a blank starting state which forces the user to actively search for items instead of being able to browse. This removes the opportunity for serendipitous discovery of items that could be of interest without the user knowing of their existence in the first place. Adding a potential for stumbling upon useful items has the opportunity of enhancing productivity and effectiveness.

Example quotes supporting this UXF:

"I don't have a clear picture of what's out there if I haven't seen it before. (Features, libraries, products etc)"

"It's also hard to kinda figure out what exists and yes you can search in backstage or whatever and if you're lucky they named it well and you can find it but it's not easy, right."

7. Allow for ways to search for purpose or functionality of items

What? Make it possible to discover an item such as reusable code, a service etc. based on the functionality or problem the user is trying to solve, not just their name or description.
Why? As of now, a user can only search for names or keywords in the hunt for items such as libraries ,see library glossary, or SDKs. For example, today, users find code snippets either by guessing the name in a search query or by identifying functionality

in the Spotify app and locate that spot in the code repository to find the code behind that it. This involves much guesswork and is arguably a time consuming endeavour, getting in the way of programming or equally important objectives.

Example quotes supporting this UXF:

> *"In general, finding out if someone has done whatever I want to do is difficult"*
>
> *" ...you can search in backstage or whatever and if you're lucky they named it well and you can find it but it's not easy"*
>
> *"If I see a certain page [view] that looks like something I want to build, I would just try to find the name of the page and what feature that is and then I just go into the code page and try and look for the code."*

The first set of UXFs represents the needs and behaviours of developers at Spotify that would inform future design decisions. Furthermore, these were to be elaborated and iterated upon by evaluating and testing the future ideas and prototypes with users. At the end of each upcoming design thinking phase, findings would be crystallized in order to improve the first draft and eventually arrive at a final set of user experience factors.

5.6 Ideate

With the design challenge being outlined through the POVs and HMW questions, it was time to start the process of coming up with ideas to address the problems. The ideas were brought to life in various ideation sessions and through a workshop with users. The ideas were further explored through low fidelity prototypes, see 4.1.4.1 and evaluated using smaller design critique sessions with users.

5.6.1 Crazy 8's sessions

To kick-start the ideation phase, several Crazy 8's sessions, see 4.1.3.3, were run to initiate the ideation phase. For each round of Crazy 8's,a couple of the HMW questions were picked and used those as themes for that particular round. This not only helped framing the ideation sessions, but it also proved to be beneficial when sharing and discussing the ideas afterwards. Instead of talking about vastly different solutions and ideas, they were conceivably more related which made it easier to combine and elaborate on them.

As aforementioned, a sharing round was held after each completed round of Crazy 8's. This allowed for explanation and reasoning behind the ideas. Furthermore, the peer was also given the chance to ask questions. As a second step, another round of Crazy 8's was held, but instead of sketching eight ideas, the focus was now to boil them down into four, combining or building upon two or more of the previous ideas from the first round. See some of the sketches below in figure 5.8.

Figure 5.8: The thesis duo going through ideas.

5.6.2 Remote workshop

Whilst the Crazy 8's sessions resulted in a myriad of different ideas, a slight creativity block was felt based on a lack of software development knowledge and ability to technically evaluate the feasibility of the ideas. It was therefore decided to invite some users and stakeholders into the ideation in a participatory design manner, see 3.4.4. They not only had a better technical understanding of the current tooling and infrastructure but also utilize many of these tools on a day to day basis.

A remote workshop was planned out with the overall aim of generating new ideas that could inspire and open up the solution space further. Four developers and one designer were invited to participate in the workshop that lasted for an hour and a half. Two participants could be considered as stakeholders since they resided in the same squad as the thesis duo, whilst the rest came from different disciplines, offices and squads and did not know each other from before. Because of the covid-19 pandemic, see 7.1, the workshop had to be held remotely using Google Hangouts, see 2.4.8, and Mural, see 4.1.6.4.

The workshop was built around the How Might We question: *"How might we allow for discovery of example code & reusables when the user doesn't know what keyword to search for?"*. This question was the result of a merge of several of the previous HMWs, albeit with a greater focus on the discovery part and not knowing what

to search for. Wherein the latter was unveiled as one of the sub-problems of code exploration and evaluation in the first place.

The participants were asked to do a cycle of Crazy 8's based on the HMW question. Afterwards, the sketches were photographed by each participant and uploaded to Mural. A round table where everyone shared and described their ideas for the rest of the group followed before a second round of Crazy 8's was conducted to iterate on the ideas. This time with the goal of creating four ideas instead of eight. As a final request, the group was asked to categorize their four ideas in accordance with the Four categories method, described in detail under 4.1.3.6.

5.6.3 Concept Creation & Screening

After the workshop, the final ideas were discussed and compared with the ideas from the initial ideation sessions. The purpose was to transform the ideas to more well defined design concepts by combining and refining them. The ideas with resemblance were merged into one whilst the rest either remained stand alone or were abandoned completely. As a result, 17 more or less well defined concepts were created. Below, a few of them are described briefly:

- **Follow-the-leader**
  Similarly to the children's game where participants copy the actions of the leader, the idea here was to utilize the more tenured developers within the same discipline and showcase their search journeys for others to follow. Alternatively let them act as role-models and highlight their repertoire of reusables built up over the years for others to take inspiration from.

- **Smart chat bot**
  Like many customer services having chat bots being the first point of entry, Spotify could make use of the same concept but to assist in the discovery of reusable code. A 24/7 support bot that in a 21 questions manner would help the developer discover unacquainted reusables.

- **Open networks**
  Stemmed from developers perceived need to rely on tribal knowledge in their search for useful components, e.g. libraries and APIs, the idea here was to visualise developer's networks and the reason behind the connections. For instance, developer X is connected to developer Y since Y is an expert on how to set up A/B tests. Having this metadata, it would allow everyone to make use of each others networks.

What followed was an unstructured evaluation and screening of all the concepts. Based on their perceived potential, innovation height and how well they incorporated the user experience factors, three concepts were decided to move forward with and to further explore.

5.6.4 Concepts

The three concepts that made it through the screening are described in detail below.

5.6.4.1 Drill down

The Drill down concept was based on the difficulty of finding reusables, as a consequence of not knowing what solutions exists or how to phrase a search to look for them. Drill down is a generic technical term for finding information by moving from general to more detailed information on a website or database. In this concept, instead of providing an empty search field as the entry, demanding a search query to showcase results, the idea of a marketplace approach was born. Each item in the marketplace would be a reusable component, e.g. a library, an API or an SDK. Per default, all available components would be presented, proposedly showing starred or recommended items first. Each component would have several tags, based on current and new metadata, connected to them. A component residing in the iOS repository in GHE, would automatically be tagged with "iOS", for instance. The same component could also be tagged with "Configuration" by its owner, based on the purpose of its existence. These tags would furthermore determine where the item would live in a tree table category structure.

Each component would be displayed as a card. When clicking on a card, a brief description of the component would be given together with an example use case and the code making up the component.

As a user, looking to discover a reusable, the tree table of categories would be used to narrow down the search and consecutively the resulting items. As shown in figure 5.9, after the first layer of categories, the user has chosen "Web". All the resulting items are therefore updated accordingly and only show items containing the web tag. As the user goes further down the tree table, less and less results appear and a few items, matching the user's need will appear.

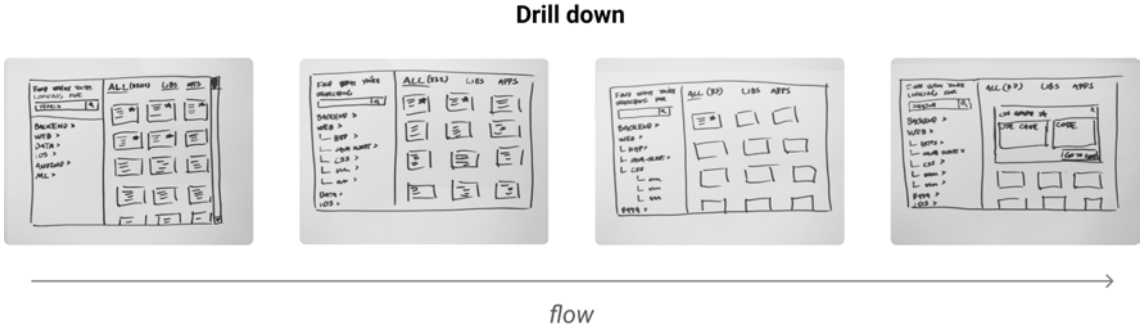


Figure 5.9: Proposed flow of the Drill down concept.

5.6.4.2 "Help Me"

Given the presence of tribal knowledge and relationship driven development, as referred to by some developers, "Help me" was designed to make this tacit knowledge, see 2.2.4, more tangible. Instead of asking a senior tenured colleague for the name

of a person that could assist them, a system would be designed to act as this middleman. Much like Expertise Browser, see 2.2.3, this system would recommend who to talk to, but based on search criteria, not on who has been doing changes on particular code.

The fundamental idea behind "Help me" is to have every developer specify their knowledge explicitly through metadata. Whatever areas self-perceived to fit a developer's knowledge would be used in the matching process.

Much like the Drill down concept, see 5.6.4.1, the empty state is to present all developers at the company. The results would be refined and filtered based on the metadata, presented as tags. Once being presented a list of developers with a seemingly matching competence coverage, a user could choose to click one of them and be taken to the direct message feature in Slack to start a private conversation with that person. This concept is described visually in figure 5.10.

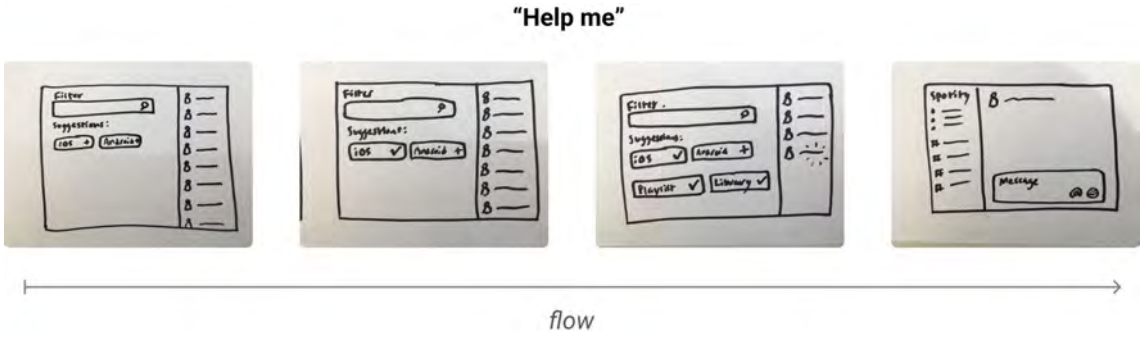


Figure 5.10: Proposed flow of the "Help me" concept.

5.6.4.3 Code tagging

Code tagging was arguably the least refined concept. The idea with Code tagging was to add another searchable layer to code by adding the ability to tag sections of it. This would be a way to address the issue of not having enough metadata options to be able to describe code on another abstraction layer. Compared to regular code comments that can be found intertwined with the production code, see figure 5.11, a tag would add another way of discovering and interpreting code without cluttering it.

```
10    # Review builds for pull requests (PR) via Slingshot.
11    gcpProject: xpn-system-z-1
12    reviewDeployments:
13      enabled: true
```

Figure 5.11: Example of a regular code comment, highlighted with a red box.

The purpose of the actual tag is up to each and every developer to decide. Some might find it useful to link a section to the reasoning behind it, whereas others might use it merely to describe the code in more human understandable and search

friendly words. The tag itself would be added to the code through each developer's IDE during or after their coding session. See figure 5.12 for reference.

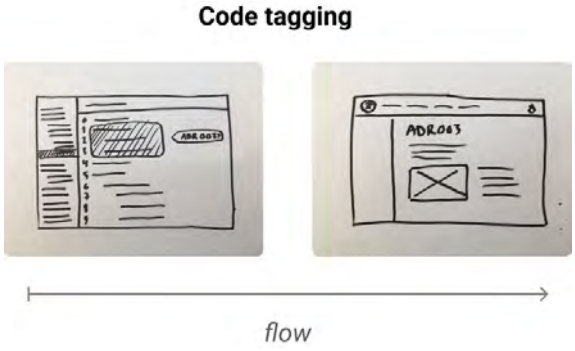


Figure 5.12: Proposed flow of the Code tagging concept.

5.6.5 Design critique

Based on the low fidelity representations of the final three concepts, six design critique sessions, see 4.1.3.7, were held remotely with six Spotify developers of different software disciplinary backgrounds and with varied tenure. The purpose of presenting the concepts in a low fidelity state, see figures 5.9, 5.10 and 5.12, was to encourage the participants to speak more freely and not feel impeded to critique the fundamental ideas of the concepts and their technically feasibility.

The design critique launched with a short description of the problem each concept was trying to solve, and a walk through of the proposed user flow. A discussion then followed regarding their overall impression of the concept; whether or not they could see themselves benefit from such a system in their day-to-day work, what technical as well as behavioral implications it might have and how the participant would propose to solve or make use of those.

5.6.6 Takeaway

Throughout the ideation phase, developers were involved to a large extent to help out in evaluation, refinement and iteration on ideas in an educated manner. By doing so, the technical feasibility of the concepts were consistently being considered without limiting the creative process. What started as a myriad of different ideas ended up becoming three standalone concepts. Based on the feedback from the design critique sessions, it became evident that all three concepts held potential whilst at the same time induced both excitement and concerns to mixed extents.

The concept of Drill down, where a user would discover buried technologies and code by walking down a tree table, was perceived as better than the existing solutions, yet technically challenging to achieve. Plenty of existing technologies would need to exist in multiple places within the tree table due to their technical complexities. This would both defeat the purpose of a clear discovery process and at the same time clutter the table. On the contrary, if a technology was forcefully placed in one

drill down path solely, this would force the user to be very specific in their choice of discovery path.

That said, the biggest potential, and what sparked the most interest during the design critiques, was the core of the idea: the marketplace approach that showcased the storefronts in terms of components by default without hiding them. Compared to similar systems at Spotify or found in the literature research, see 2.2.3, a marketplace has the advantage of making the existing reusables browsable. For example, in comparison to previous solutions, not tying the solution to the Integrated Development Environment (IDE) removes the requirement of having to add code as a search query before any results are shown. A marketplace allows for browsing of content without a forced query. However, recommendations were made to revise the content of the storefronts. Being presented with a component, e.g. a library, API or SDK, with an arbitrary name, would not help much if the developer does not know the context of the reusable or how to make use of it. As proposed in the Drill down concept, this kind of information would first be presented to the user once a component card was clicked. What was suggested was to change the storefront to what a developer is actually looking for when going down a path of trying to find a reusable component, namely what functionality and use case they support. More specifically, instead of showcasing the component itself, it was proposed to let the use case of the component be the storefront, and inside each use case present the technical solution supporting it.

"Help me" proved interesting to the developers, mainly due to the opportunity of finding knowledgeable colleagues within fields one was not familiar with. Especially less tenured developers could see great potential in being able to opt into a curated web of competences linked to unfamiliar colleagues, instead of constantly having to bother their squad mates. However, a shared concern that arose was how the ties between knowledge metadata and the person in question would be created and maintained. If one would add only a few knowledge fields to one's profile, that person would be difficult to find, whilst the opposite would yield an excessive amount of results, making it difficult to choose who to contact. This problem would be equally evident if one went down the quantitative route of tools like Expertise Browser, see 2.2.3. A developer with long tenure and a history of development within multiple areas would show up often in result fields, resulting in many questions for him or her.

Discussing this with developers, it became evident that the current way of finding people, i.e. by searching in Slack channels, provided several advantages over this concept. First of all, asking questions directly in bigger Slack channels provides multiple eyes on the problem. Posting a question within a large channel yields a great likelihood that the suggested contacts through the "Help me" system would also reside in that particular channel. Secondly, asking questions privately via Slack's direct messaging feature has the disadvantage of being somewhat intrusive and, more importantly, non-searchable for the rest of the organization. A system like "Help me" would therefore arguably increase the siloed tacit knowledge rather than spreading it, therefore influencing both discoverability and productivity negatively.

On the positive side, the developers liked the idea of representing metadata as tags. More specifically, they liked the vision of using human and/or system made tags as filters in their mode of discovery. The concept of "Help me" was, hence, terminated but the idea of utilizing tags was brought forward.

The concept of Code tagging within the IDE was generally touched upon briefly and evoked rather lukewarm responses from the developers. Tagging, and in a way commenting, code, is already done excessively at Spotify as part of the coding conduct. Introducing a new way of tagging code, creating consistency and a process for how these tags would be searchable and discovered deemed challenging for the developers, both from a technical as well as a behavioral perspective. Making sure that this would be adopted by a majority of the individual contributors within the company, was perceived as unreasonable by the developers, and the concept was therefore discarded.

Yet again, the Code tagging concept was built on the concept of using tags, much like in "Help me". This was expressed as far better than the text based search system currently in place today. Whilst most developers were intrigued by the idea of being able to search for and discover a library via automatically or human generated tags, these tags would suffer from the same problem of creation and maintenance as the tags in "Help me". The difference here would be that the responsibility could be placed on the owners of the component, wanting the component to be found.

To summarize:

- The fundamental idea of a marketplace, proposed in Drill down was decided to move forward with.
- Change the idea of component storefronts to instead display the use cases of the components rather than the components themselves.
- The two concepts, "Help me" and Code tagging were abandoned but the utilization of tags was kept for the forthcoming final concept to be explored.

5.7 Prototype

Having decided to move forward with a marketplace based design where users would search for and discover reusable code components and technologies such as SDKs, APIs or libraries via their use cases, the project moved into the prototyping phase.

This phase comprised of designing medium fidelity prototypes, see 4.1.4.2, in Figma, to manifest the discoverability idea and to have something concrete to base a design critique and following usability test on. The prototyping work was divided into the two cornerstones of discoverability, see 2.2, outlined below:

1. **Prototyping for Discovery** which meant exploring how to narrow down a search and how a user would want to explore and search for use cases.

2. **Prototyping for Interpretability** which entailed investigating and supporting the understanding of the content that would be presented from the search.

Using Figma as the main prototyping tool allowed for working collectively in real time and sharing the progress with stake holders and users continuously, see 4.1.6.2.

5.7.1 Prototyping for Discovery

With the feedback from the design critique session in mind, see 5.6.6, the prototyping of a home page for a use case based marketplace commenced.

Different approaches for searchability and discovery were explored by benchmarking industry leaders, as well as looking into the tools of search and discovery created or used by the developers at Spotify. The benchmarked tools were search engines like Google, marketplaces such as Amazon, Blocket, Zalando and Boozt, the Q&A platform Stack Overflow Enterprice and Hackernews as well as Github Enterprise. The internal technology platform Backstage, see 2.4.4, was also analyzed to find what usability patterns and technical solutions already existed and were common to the developers.

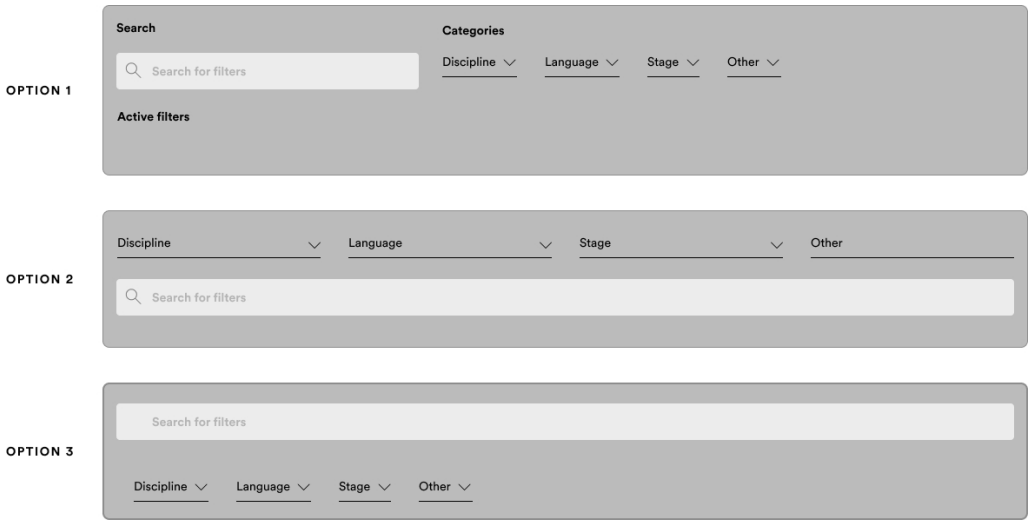


Figure 5.13: Three versions of the prototyped search bar and category drill down.

5.7.1.1 Search Bar & Category Drill Down

Common for all the platforms was a search bar, something that also was regarded as necessary for the final design concept. Since the design would involve tags related to use cases with vastly different substance, it was evident that the tags eventually would become plentiful in numbers. To allow for a swift localization of these tags, a search bar was regarded as the best option, which furthermore was strengthened in the feedback from the design critique, see 5.6.5. A way to find tags via category drop downs were also introduced as a way to narrow down the search but also to provide inspiration of what could be searched for, see figure 5.13. A search bar with correlated filters follow the *mental model*, see 3.4.8, of what a user can expect from

a search system like this and also incorporates two of Jordan's design principles, see 3.4.5, namely *compatibility* and *appropriate transfer of technology*. To create *visual clarity*, *consistency* within the product and provide relevant *feedback*, one of the bigger prototyping challenges became how to help guide the user whilst searching. To further assist the user during the search, a search result marker, displaying the amount of results that would be displayed if a certain tag was chosen, was added to the search drop down.

5.7.1.2 Auto-complete

Taking inspiration from other systems, it was clear that searching for tags would benefit from auto-complete, since it greatly increases the chance of the user entering a search query with meaningful results. The system should advocate for a search towards tags whilst also allow for free text search. How to visualize and communicate these differences and what effect they would have to the system was vital for the *user control*, see figure 5.14.

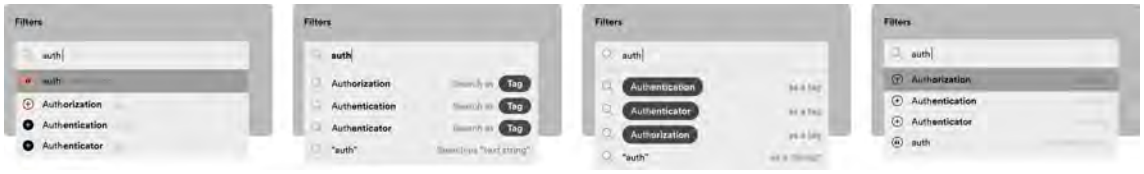


Figure 5.14: Four versions of the prototyped auto-complete search dropdown.

5.7.1.3 Search & Filtering Mix

To allow for discovery of tags and not limiting the user to only find use cases by manually typing in the search bar, the idea of merging the search bar with the category drop downs was introduced and explored. That way, if a user does not know exactly what to search for, it would still be possible to follow the flow of the categories and create a search query by simply discovering and selecting one or multiple tags from the category drop downs. This way, the system would take *consideration of the user resources*, by allowing for multiple ways to get started with a query.

This complicated the user experience slightly since an action within either the search bar or one of the category drill downs would have to achieve a response in the other one. Bellow are two different examples of how this could be achieved. The first example shows a tag based search that adds search tags one by one to the search bar, see figure 5.15. The search bar acts as the container for the search query itself and indicates through the text cursor that another tag added in the system would be added as the third tag in the search bar. The second example hosts a separate Active Filter area under the search bar and drop downs to be used as the container for the tags, see figure 5.16. Each tag added, by either the search bar or drop down menus, would show up there.

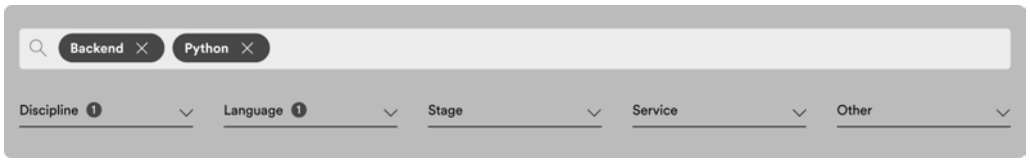


Figure 5.15: Tags being added directly to the search bar.

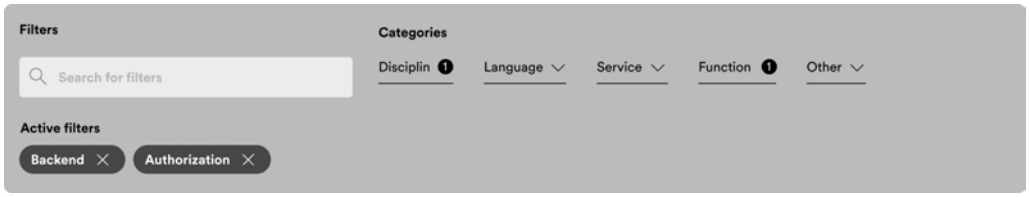


Figure 5.16: Added tags appear in a separate Active Filter section.

5.7.2 Prototyping for Interpretability

With the slight change of direction that came with the new use cases storefronts, there was a need to talk to a few developers to better understand how to best portray these use cases and what a use case actually should entail. Three developers were contacted and asked to share their opinions on the matter.

When it came to the interpretability, there were two main areas to consider: the actual storefront in terms of the *cards*, and the *use case page*, i.e. the page that follows once a user decides to explore a card. Both of these are tightly connected to content and making sure the user understand and can interpret the use cases.

5.7.2.1 Storefront Cards

Given the wish to present a number of use cases at the same time, each card housing a use case therefore had a limited space to account for. In accordance with Jordan's design principles of *consideration of user resources* and *visual clarity*, it was also important to not clutter each card with too much information. Only the most essential information necessary for a user to determine whether or not the use case carries fruit would be possible to show.

Based on these criteria and the input from the talks with three developers, a few different designs of cards were explored, see figure 5.17. It was quickly deemed necessary to showcase the tags connected to the cards to make the connection to the filters as clear as possible. Having placeholders for a header and a short description were considered minimal viable product and were hence included in all versions. To cater for the expressed desire to have a way of evaluating the use case already at this stage, a few different metadata points were tried at the header of the cards. More specifically, the notion of incorporating likes, views or date of last update, see figure 5.18.

Throughout Backstage where the proposed system is suggested to live, the possibility of starring items is featured heavily. In case a user stars a page, item or section,

Figure 5.17: Different design proposals for the cards.

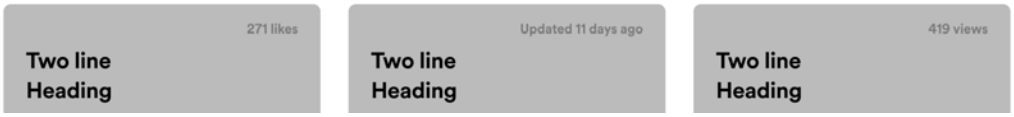


Figure 5.18: Different metadata in the headers.

this will appear under the Favorites section on the Backstage homepage, see figure 5.19. For this reason, it was explored to add a way of starring items directly from the cards view.

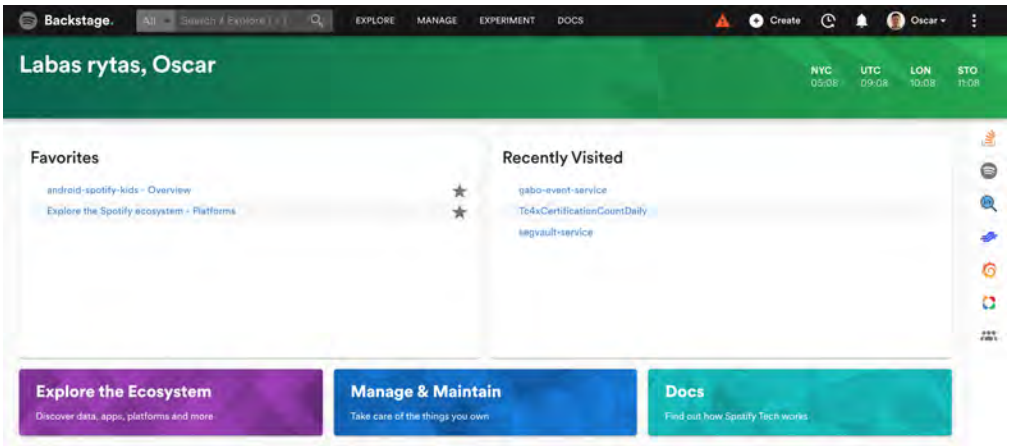


Figure 5.19: Backstage home page with Favorites section to the left.

5.7.2.2 Use Case Page

As mentioned earlier, clicking on a use case card would take the user to a use case page. If the purpose of the card was to summarize the use case briefly, the purpose of the use case page is to describe it excessively. Not necessarily in terms of a longer description but rather by highlighting more metadata connected to it. For instance, the owner of the underlying technology behind the use case, links

to technical documentation in TechDocs, see 2.4.5, as well as links to the code repositories in GHE, see 2.4.2.

After discussing what to include on use case page with developers, the idea of creating a sandbox environment where developers could play around with an example app, manifesting the use case and its promises, emerged. This would occur directly in the web-browser, similarly to other online code editor services like Glitch and Codepen. It seemed as many developers, as a part of their discovery and evaluation of reusable components today, already include to test out the code in their own IDE to interpret it and see if it fits their need. By adding this ability straight in the browser it would prevent the developers from having to context switch. The first design therefore allocated a big part of the page for such a section, see the Example app section in figure 5.20.

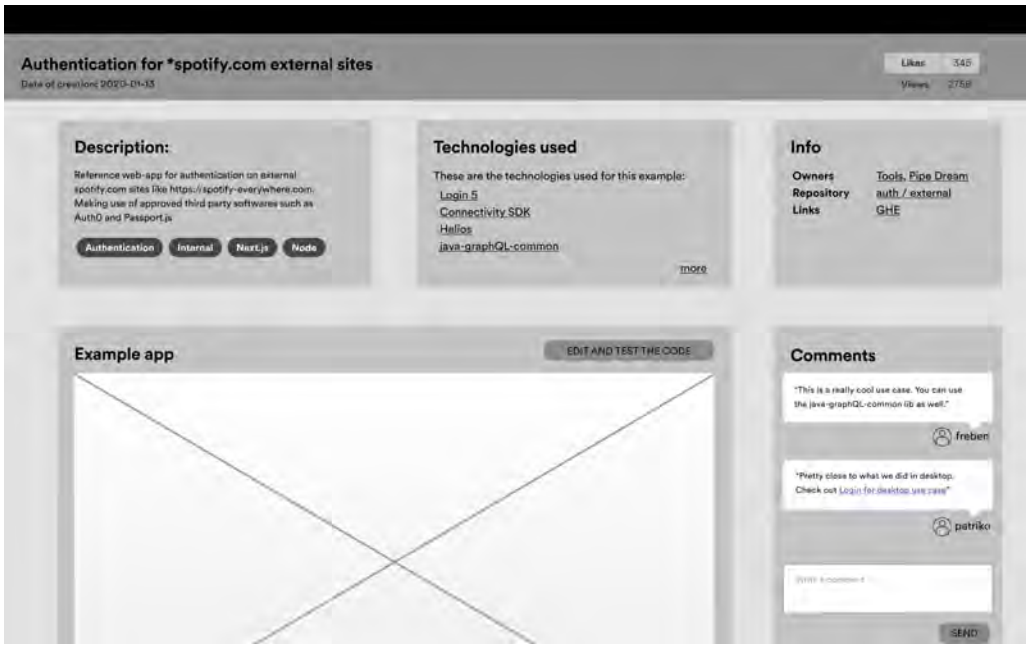


Figure 5.20: Initial design of the use case page.

Another feature exploration was the ability to have a comment section in the use case page view. This would allow any viewer or owner to add comments. Be it more FAQ based or as a way to show appreciation. Alternatively to communicate errors and bugs.

One of the more important sections of the use case page is the Technologies used section, see figure 5.20. This is the place where the user would see the underlying components building up the use case, i.e. what used to be the storefronts in the initial Drill down concept. Linking to the Backstage pages of these were considered as a necessity in a minimal viable product.

5.7.3 Prototype Feedback Sessions

Once having prototyped for both the discovery and interpretability of use cases, three feedback sessions were run with different developers. For the full session script, see appendix D. During the sessions, the developers were first introduced to the vision of the concept, as stated below:

> *"Our vision is for the use cases section to become a marketplace where squads and individual contributors can share and highlight their reusable components (in the form of libraries, APIs, datasets, SDKs etc.) but in a context. Instead of just listing the entities as a list of items, they are explored through different use cases. Each use case makes up a card and is described briefly for the user to get a glance of what the case is about. One use case can make use of, and highlight, several libraries in combination, similarly to how these entities are used in the code repositories."*

What followed was a period of prototype screens being shared with slight alterations of certain features. For instance, the different types of cards were showed with different evaluation criteria. The developers were consistently asked to provide their feedback and preferences for each feature. As can be seen in figure 5.21, the proposed categories of filters and their content were also displayed to better understand how this way of adding tags was perceived.

Figure 5.21: A developer giving feedback on the categories of the medium fidelity prototype.

5.7.4 Takeaways

From the prototype feedback session, a number of conclusions could be drawn.

Use cases as a concept was widely understood and the developers had a similar view on what to expect from a use case item. Two of the developers made parallels to the internal *Golden Paths*, see 2.4.4, and described these best practices as a collection of use cases. They also saw the potential for a use case page to be populated by less common cases than the Golden Paths and therefor hold more specific information. These positive remarks further strengthens the UXF rational behind *7. Allow for ways to search for purpose or functionality of items.*

The feedback on the system categories varied. The first two categories from the left, discipline and language made sense to all the developers, whilst the other three did not. Assuming the hidden content was difficult and most of them exclaimed that they had to open the drop downs to even have a single clue. Therefore, it was decided to keep the categories of discipline and language whilst working on bettering the rest to live up to the UXF *7. Allow for ways to search for purpose or functionality of items*. Two other categories that were proposed for the drop downs were *dependencies*, referring to if certain use cases would be limited to greater dependencies within the code repository, and *Spotify verified*, where one could filter for manually verified and blessed use cases that are best practices and proven to work. The wish for this last category further prove the UXF of *2. Assist users in finding the "blessed" choice*.

Regarding evaluating a use case on both the use case card and page, the most pragmatic metric was the *last updated* marker. Since use cases, like documentation, will degrade over time it can not rely on likes and views as evaluative data like other content. An old use case would most likely have a plenitude of likes and views, even though the use case itself might be deprecated since many years. The traffic was also deemed too low to uphold likes as a credible source of evaluation whereupon this was scrapped. One developer also expressed a worry that use cases would turn *"clickbaity"* if views were chosen as a metric, prompting the developers to write promising headlines to get more views. It was hence decided to stick to last updated.

When entering the use case page, the developers expected to see code snippets, library references, documentation and information regarding owners and how to contact them. A majority expected some sort of tutorial or step-by-step guide on how to understand and make use of the code. A quote describing why is found below:

> *"There is more to the use case than the code itself. I don't only want to know that "this works", I want to know what other people that have thought about this deeply have decided, and why. It should clearly explain the "why's" to the use cases. There is otherwise a risk that I will apply the code without knowing exactly why and how it works which could eventually lead to security issues etc."*

When it came to what could be added to further strengthen the concept, a few elements were proposed. One suggestion was to be able to see how many people or squads are actively using the particular use case which was described as *"very potent"* during the feedback session. It would also work as *"a way to have proof that it works"*. The need for even more evaluative markers and data points further establishes the UXF of *1. Allow users to evaluate the quality of the items discovered*. The developers also wished for the use cases to have a deep integration with documentation already existing at the company, allowing for the ability to dig deeper into technical details if necessary, further proving the UXF of *4. Allow users to access corresponding information or documentation for a given item*.

In regards to updating the prototype, it was decided that a real use case, populated with relevant and accurate data from the internal systems at Spotify, would be prototyped. This to be able to fully describe the idea of the use cases concept to developers during the usability test. The idea of creating a standalone sandbox environment within the use case page, where the developers could try out and explore the example app manifesting the use case in real time, was dropped based on the developers' feedback. They regarded it as too much overhead to create these example apps and such an increase in their workload that there would be a great chance of individual contributors rejecting to make them due to the load. On top of that, some technology disciplines, like machine learning or back-end development does not have an as easy way to create code that can render front end content, which further strengthened the case of dropping the idea of an example app. Instead, it was decided to render information that already exists within the Spotify ecosystem, or at least information that would not take much time to add in addition. Technical documentation in the form of TechDocs and the README.md files, already in each item's code repository, are examples of where most of the information could be scraped. By utilizing these, there would be minimal to no overhead in creating a use case but rather an aggregation of data and information that already exists. For more rich, curated content tailored for the use case format, an additional use case markdown file could be proposed as an extension of the repository.

5.7.5 User Experience Factors - Draft 2

During the prototyping phase, the UXFs were revised and updated as more insights were gained. Compared to the first set of factors that were mainly based on the research findings from the Empathize and Define phases, the second draft and the additional factors that came with it were rather grounded in the user's feedback on the concepts and the proposed future flows, as well as by studying external solutions.

The following factor was re-formulated:

7. Allow for ways to search for purpose or functionality of items

Which instead was changed to *Allow for alternative ways of discovery by making use of metadata* to not limit the factor to the act of searching but rather include the wider perspective of discovery that could include alternative ways of narrowing results. The new formulation of using the word *metadata* also more clearly communicates how the purpose and functionality, used in the original phrase, is meant to be extracted, whilst not limiting it to be that exact kind of informative metadata.

The following factors were added:

8. Make users feel in control of their discovery by communicating the effect of their changes

What? To clearly state the cause and effect of adding additional filters or keywords in a search query, it is important to communicate the implications this has on the

resulting items shown. Be it to clearly communicate why a certain result is shown, how the number of items has increased or decreased, or the difference between a tag and regular text string that the developers are familiar with.

Why? By utilizing and encouraging the, in this context, unfamiliar concept of tags, it is important to assist the learnability to make the system feel less estranged and counter-intuitive for users in the upcoming encounters. Furthermore, Jordan's design principles, see: 3.4.5, *user control* and *feedback* also goes well in line the factor.

Example of external applications supporting this UXF:
The online clothing store Boozt.com uses tags for a shopper to be able to filter the items shown based on their preference. As soon as a filter, i.e. a tag, is picked, the corresponding articles shown will update accordingly. Every item shown will furthermore state that they indeed match, as in the example shown in figure 5.22 where the brand of the item is clearly communicated for the user to understand the connection.

Figure 5.22: Boozt's way of communicating the connection between an added tag and the results presented.

Google's search engine communicate the connection between a string and the result shown by making the matching keywords bold, see figure 5.23.

9. Allow for further exploration of related items

What? When being presented with a use case and the underlying technology behind it, adding the possibility to explore other, similar use cases or technology could further aid in the exploration and discovery process. Instead of ending up in a dead end and having to back to the results view, a user could benefit from being able continue their journey straight from the result with a knowledgeable developer guiding the next potential steps.

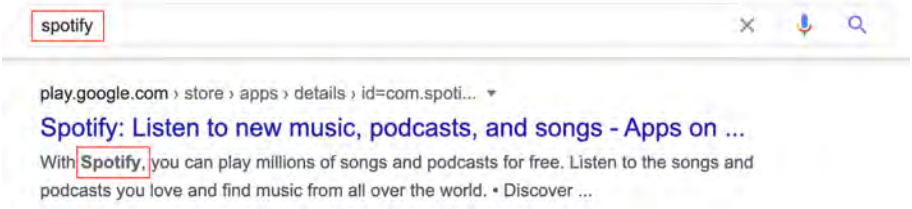


Figure 5.23: Google's way of communicating the connection between the search keyword and the results shown.

Why? By having owners recommending other related use cases or technologies, it would remove the need for the developer to know the exact matching tags, but still be able end up at what is considered as the right place. By trying to propose a similar option, the system arguably assist in *error prevention and recovery*, being one the cornerstones of Jordan's ten design principles, see 3.4.5.

Example of external application supporting the UXF:
Boozt.com, as mentioned before, provides a list of related items, see figure 5.24, to the one currently viewed as a way for the customer to find an item that potentially will suit them even better.

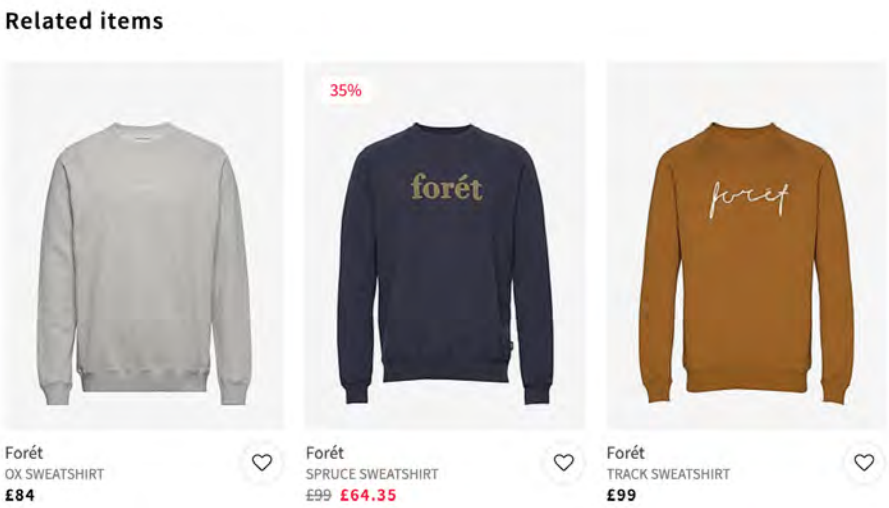


Figure 5.24: Boozt.com presenting the user with related items based on what they are currently looking at.

10. Assist users' discovery process by suggesting related filters

What? To further assist the users in finding the right item, the system should suggest additional related filters that will return in a better and more narrow list of results. The suggestions could either be based on search patterns from previous users or occurrence in relation to already specified filters.

Why? By having the system suggesting related filters, the user might find filters they normally wouldn't use and therefor invites for a more exploratory discovery

process.

Example of external application supporting this UXF:
Google uses the related tag pattern frequently in their images section of the search engine, see 5.25.

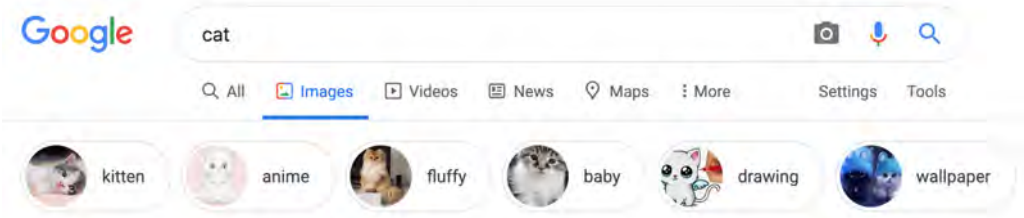


Figure 5.25: Google suggesting related tags to refine the search results.

5.8 Test

The testing phase involved usability tests with six developers, conducted to be able to evaluate and refine the final concept before moving on to make a more high fidelity prototype, see 4.1.4.3. Apart from being able to tweak the design, the purpose of the tests were to evaluate the usability and intuitiveness during a first encounter with the system. Finally, the last iteration of the user experience factors was carried out and a couple of new factors were added to the list.

5.8.1 Preparing Prototype

With the feedback from the prototype feedback sessions in mind, the design was tweaked and in cases where several different options were explored, one was settled with, with either design principles or user feedback backing the choice. In order to run thorough usability tests, the prototype had to be extended with more screens as well as come to life to a greater extent using animations. This was done by prototyping the ability to search in the search bar, albeit with limited keywords, and by allowing the user to explore every category dropdown.

It was also raised as a wish to populate the prototype with real data and information. This way, the concept would be perceived as more trustworthy and real, and therefore would receive more pertinent feedback compared to running the tests with out of context, bogus information.

5.8.2 Usability test

The usability tests, see 4.1.5.1, were performed using the medium fidelity prototype living in Figma, see 4.1.6.2. and the testing tool Lookback, see 4.1.6.3. To start off with, the testers were given a scenario and a goal that aimed to place the testers in the right mindset. For the full usability test script, see appendix E. The key-points for them to keep in mind for the test were:

- You are a back-end developer at Spotify.
- You are looking to build a feature for your back-end, written in Python, that can track different user events.
- You are determined to try out the new Use Cases section within Backstage.

Throughout the tests, the developers were asked to think aloud to express their reasoning and thoughts in every step of the journey. At times, the testers were asked to pause for a minute to answer questions, like:

- What do you think "Recommended" means in this case?
- How do you perceive the different suggestions?
- If you were to evaluate this use case - how would you do that?

The test was designed to force the user to go to a wrong place and adding the wrong tag, for the simple reason that it was desired to also test how they would recover from faulty states, likely to happen if the concept is to be implemented later.

After reaching the goal and the right use case, the testers were asked about their overall impression of the design, what they liked the most and least about it, as well as how they would compare using this tool with their current discovery process. As a final request, the testers were asked to fill in a quick questionnaire with five Likert scale questions, see 4.1.1.4. For the full questionnaire, see appendix F.

5.8.3 Usability test analysis

To follow up the usability tests and the answers from the questionnaire, an analysis session was run where the test were all revisited and walked through whilst glancing at the notes taken during each test. A list of actionable insights for the concept took form and potential new UXFs, based on the behaviors from the users during the tests, were written down.

5.8.4 Takeaways

In general, the concept of discovery of technology via use cases was received well. A majority of the test participants understood instantly what a use case was and what to expect from each use case component. The ones that misunderstood the page at first glance adjusted their mental model as soon as they entered a use case card and got the information laid out for them. Since the current prototype has use case cards shown out of the box, a future user, new to the system, can easily click one to be informed of the basic setup, thereby learning what to expect from the system, fast.

In contrast to the discoverability systems of today, where one has to know the name of what one is searching for, searching and browsing through use cases was perceived as *"1000% better"*, as one tester phrased it. Albeit exaggeratedly expressed, the

superiority is in line with what previous research also claims, see 2.2.2. Some of the comments were:

> *"With the number of tools at Spotify, I think this would be very helpful"*
>
> *"This looks very important, I hope this makes it to Backstage."*
>
> *"You guys are on to something, the core idea is good"*
>
> *"This is my favorite part, that I can browse instead of the old way when it was just through code search and you kinda had to know what you were looking for"*
>
> *"I like to poke around initially, good to see that you're showing the cards from the start"*
>
> *It feels a little bit like maybe we are currently doing this in documentation but more adhoc. We don't use the term use cases at Spotify - It is more spread out. Actually formalizing it "THESE are the use cases and you can find them all here" - I think that is a really good idea."*

These comments strengthen the overall idea even more but also proves the system to adhering to some of the UXFs: *6. Make browsing of items possible and 7. Allow for alternative ways of discovery by making use of metadata.* Below are further insights gathered from the usability tests.

Insights on discovery

One of the things evaluated through the test was the search bar and corresponding categories. The prototype had taken a controversial stance by mixing the tag based search with the finding of tags inside the drop downs, see 5.7.1.3. This showed to be a bold move, going against some developer's mental model, resulting in the feature sometimes being praised and sometimes criticised by the testers. One participant called it *"A strange UX pattern"* whilst a separate participant called it *"cool"*. However, both finding and showing the selected tags within the search bar as well as the drop downs proved redundant and had to change. This became especially clear when thinking of eventually having to add every imaginable tag under the drop downs, potentially forcing the user to use internal search bars in each drop down to find the appropriate tag. Therefore, the idea of combination drop downs were abandoned and replaced with classical filter options instead. To keep providing a way to find or browse for tags, it was decided to implement a recommendation feature to the tags, recommending popular tags to add to the search based on the current search query. This decision also goes hand in hand with the UXF of *10. Assist users' discovery process by suggesting related filters.*

Observing the way the users interacted with the proposed search results in the search window drop down, it was evident that a majority pressed "Enter" after having typed only a few letters of the word. There was a mental model regarding

being able to select and add the top suggested tags directly into the search bar by the click of a button, much like Google. The prototype had been designed to be able to accommodate for this, but it was decided to encourage this behavior by adding further visual elements showcasing this functionality, thereby adhering to one of Jordan's rules on *compatibility* with other systems.

As expected, the category titles of *Stage* and *Service* were hard to comprehend at first glance. With categories of tags dropped as a feature, it was decided to re-name these drop downs and give them a slight different functionality now that they would work as more traditional filters.

When it came to the *Recommended* tab, the participants were divided between either understanding it as an editorial section, hand curated by an infra squads at the company or a statistical sorting on most used or interacted with use cases. The participants were, however, confused with this tab disappearing once a search was initiated. In case a Recommended pattern was to be utilized, it would have to be revised and made clear what it meant.

Insights on interpretability

Once the participants made their way into the use case pages, any uncertainty regarding what a use case was, faded. Clicking the card and being transported to the use case page showed the focus on documentation, meta data and example code that some of the participants had thought would be templates or automatic code generators.

Although the metadata being shown was appreciated, four additional data points were requested.

- How many times this use case is being used within the Spotify code repo. This was requested as an additional way of evaluating the code to make sure one was not the first user making use of the library or connected technology.

- The three status indicators found on Github, namely experimental, production and deprecated. These were suggested from the wish to also be able to evaluate them based on this information. In some cases, a developer might be fine with using something that is in an experimentation phase but for some projects, the use case needs to have some indication that it can be trusted and will not be adding bugs or similar problems.

- Presenting potential dependencies. Some use cases might be based on technologies that have specific dependencies within the tech stack. Showcasing these would be vital since some use cases would be useless if the developer could not adhere to the dependency premise.

- Display most recent library version. Libraries could come at different versions and displaying the newest one would save time of the developers search for which version to implement.

Except for the meta data, the most discussed feature was the comment section. Comments in their current shape and form did not appeal to the majority of the developers. They deemed it as *"not transparent enough"* regarding who would be notified when a comment was added and how well the comment section would age over time. One developer exclaimed *"I would not expect to find help in the comments"*, thereby invalidating one of the reasons to its existence. Instead, it was proposed to add either Github issues or Stack Overflow Enterprise Questions related to the use case to the use case page, thereby accommodating for the ability to find and receive help as well as asking questions. Looking through the options, it was decided to explore a Stack Overflow plugin integration connected to the tags of each use case.

Another desire from the participants was to be presented with actual production code examples of where the use case was being used currently within the code stack. Being able to first receive a short summary of what can be expected from the technology, how it is installed, and then be presented with one or more real examples on how this had been implemented in the past was regarded as *"very potent"*. Therefore, it was decided that this would be added as a new UXF, see 5.8.5 and be explored in the final prototype. One developer summarized it like this:

> *"[This use case is] asking me to use this tech, but can you show me how this is done successfully? I don't mean examples but real production code. Like, "This [use case] is being used by these well known services at Spotify" - that would be nice!"*

In the scenario created for the usability test, the developers were lead to a page that was supposed to be understood as deprecated. They were supposed to realize this and move over to a newer use case linked within the deprecated one. A majority of users needed a long time to realize that they indeed had entered a deprecated item, and some of them even had to be told so. Since the use case page will be based on documentation and other types of meta data, there is no way to guarantee each use case to be perfectly up to date. Therefore, there needs to be a way for the system to flag a use case as potentially deprecated based on some combination of meta data and for the user to easily be notified or warned when arriving at one. This needed to be addressed in the final prototype.

The likert scale results from the questionnaire was used mainly to see how the prototype performed towards the UXFs and to know if it could be used as an example of good discoverability. On every question, the prototype was validate as 4 or higher on a scale from 1-6, which was regarded as a success and something that would be enhanced when applying the proposed changes in the high-fidelity prototype. For full results from the questionnaire, see appendix G.

5.8.5 User Experience Factors - Draft 3

The final iteration of the user experience factors was made based on the feedback and insights gained from the usability tests. Neither of the previous factors were changed or removed, but a few new ones were added, namely:

11. Consider the entry point of the user

What? The system needs to adhere to the knowledge and information at hand for the user when the search commences. Make sure to research and support the information that is at hands for the user when s/he enters the discoverability phase and explore how this could be utilized to present a relevant search result.

Why? Each discoverability context is different and adhering to the knowledge or preconceived notions of the specific user is vital to create a good discoverability system. This goes hand in hand with Jordan's design principle Consideration of User Resources, see 3.4.5. In the case of Spotify, it was apparent that the developers knew of what functionality they wanted to find but lacked the connection to a use case and the technology permitting it.

Example quotes supporting this UXF:

> *"If I was new or in a new domain then the [use cases] seem very useful actually, finding something I don't know about would have been difficult in my current way. [There is no way] to search for something mobile related."*
>
> *"...you don't have to know what you are looking for [when using this]."*
>
> *"The ability that I can browse, as opposed to code search where you had to know what you were looking for...here you could come across something you didn't know you were looking for."*
>
> *"Maybe it's just me. Like, I need to search better or something. (when trying to find how to connect backend to web)"*

12. Assist users when reaching a dead end

What? When in a journey of searching for items, there is a risk reaching a dead end in terms of the system not finding any proper matches. In this critical situation, it is important to assist the user to recover from this state. This could, for instance, be done by suggesting a change of the current filters that would yield in matching items.

Why? In a situation where a user is presented with 0 results, the risk of the user abandoning the system is the highest and should therefore be treated as very severe. The factor is also closely related to Jordan's design principle Error Prevention and Recovery, see 3.4.5.

Example of external application supporting this UXF:
One of Sweden's biggest second hand marketplaces online, Blocket, tries to assist the user when reaching a discovery dead end by suggesting straightforward actions, as shown in figure 5.26 to extend the search area to the whole country of Sweden instead.

Figure 5.26: Blocket giving suggestions that might yield in search results.

13. Allow for saving items

What? When encountering an item that, for any reason, is important enough now or has potential for the future, a user should be able to save this item and return to it via a separate, simpler path than it was discovered.

Why The reason to why a user would like to preserve an item can be many. Either, the item was of such use now it deserves to be elevated, bookmarked and kept somewhere to be able to come back to. A different reason could be a serendipitous discovery of something that might prove useful or interesting in the future, which makes the item liable for saving. A third reason could be lack of time and saving interesting findings for later.

Example of external application supporting this UXF: One of the worlds biggest e-commerce sites, Amazon, lets their users save items that are of interest by either adding the product to the digital cart or to a wish list, see figure 5.27.

Figure 5.27: Amazon presenting two saving options.

14. Strive for transparency and assist in scrutiny

What? In a professional context, as in the case with an enterprise software highlighting reusable components, it is important to be transparent to the user with the information presented and the sources behind that information; where it comes from, who the owner is etc.

Why? If a user is to trust, and furthermore utilize, an item discovered through a system, the system should provide the user with enough information and/or sources to more information to make the user feel safe. Whilst more information can be desirable, it is still important to consider the user resources, as expressed as one of Jordan's design principles 3.4.5.

Example quotes supporting this UXF:

> *"I don't know who built these [use cases], so I don't know how reliable they are"*
>
> *"It's usually hard to just take a library without understanding what is going on under the hood."*

15. Display examples of the items in their proper contexts

What? To assist in the interpretability of an item, the system should include the ability to present items in their intended context, i.e. in the context they are supposed to live or be used. In the case of a code component discovery system, this could be manifested by showcasing real examples of production code utilizing the component. An analogy in an online clothing store environment would be to not only show a picture of the shirt itself but also a picture where a real person is wearing the shirt.

Why? By letting the user familiarise with the context of the item, it is arguably easier to understand its full potential, arguably aiding in the interpretability of it.

Example of external application supporting this UXF:

(a) Item itself (b) Item in context

Figure 5.28: The Human scales online store showcasing an item both in and without context.

5.9 Refinement

Based on the feedback and insights from the usability tests, see 5.8.4, the medium fidelity prototype was changed accordingly. For images and a complete description of the final design concept, see section 6.1.

Look and Feel

To make sure that the final design concept would fit within the preexisting internal software ecosystem, an internal design system was used in order to achieve coherence, see figure 5.29. This made the refinement process of the medium fidelity prototype simple and quick since most objects could be translated seamlessly via the design system from medium to high fidelity.

Since the solution was proposed to live within Spotify's internal developer platform, Backstage, already existing navigation components and headers were utilized to mimic those graphical elements.

Figure 5.29: Extracts from the internal Spotify design system

Changes on discovery

The search bar and drop down combination turned into regular filters and a search bar with a popular tag recommendation feature. The *Recommended* tab was removed and replaced by a special tag living in the use case cards that shows use cases supported or validated by internal infra teams. These specially tagged cards would, by default, be showcased first in the resulting card view – something that can be changed by changing the current sorting option in a drop down. The design and copy of this new tag were decided by letting internal development team members vote for the tag they believed to be the most instructive, see figure 5.30.

Changes on interpretability

The requested additional data points, see 5.8.4, were added, see figure 5.31. The comment section was removed based on the feedback and was replaced by a Stack Overflow Enterprise component as an alternative source of help and FAQ. Production code was given it's own segment in the use case page since it had received much positive feedback from the test participants. To handle the problem of accidentally bumping into deprecated use cases, it was decided that deprecated solutions would not be displayed by default. Instead, the users had to opt-in to even be able to

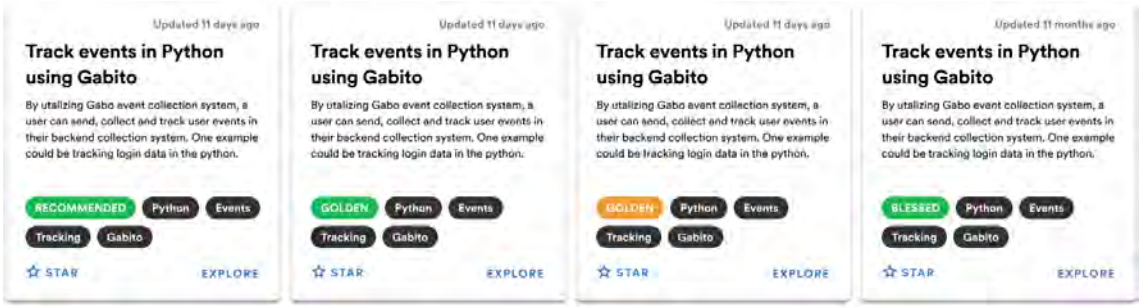


Figure 5.30: Different designs for the "approved" use case cards.

find deprecated solutions. In this case, a warning tag would be added to the use case card as well as on the use case page to notify developers about the code being deprecated.

Figure 5.31: Use case page header with added data points such as *Active users*, *Lifecycle*, *Dependencies* etc.

6

Results

The results of this thesis project, presented in this chapter, are divided into two parts. First, the final design concept, the Use Case Marketplace, that manifests the user experience factors created in this thesis is described. Secondly, the final set of 15 user experience factors to take into consideration when designing for discoverability is presented, thus, answering the research question of the thesis project:

Which user experience factors should be considered when designing for discoverability in the context of enterprise software to enhance productivity?

Discoverability is defined as a quality of discovery and interpretability; being able to search for, locate and understand items or assets.

The result consists of a set of 15 user experience factors to take into consideration when designing for discoverability in the context of an enterprise software and are the consolidated list of factors from the previous three drafts mentioned in the Process & Execution chapter, see 5.

6.1 Design Concept

The final design concept named Use Case Marketplace is a new browsable forum for use cases – use cases of existing reusables like code components, libraries, APIs and SDKs as well as experimental hacks. The use case, i.e. the reusable's functionality, is the storefront and the way that a user discovers a technology. The reason for this is simple; the user can articulate what it is s/he wants to accomplish or what functionality they are looking for in a technology. This knowledge can therefor be applied as data to run a search query on when looking to find reusables as possible solutions. In contrast to today, when users are searching blindly for arbitrary and unknown technologies and reading up on their potential use case, this model of discovery is flipped on its head. Instead, it is the use case that is searched for and discovered, pointing to the underlying technology providing the functionality.

A developer initiating a project could be compared to a chef wanting to cook an Indian stew. To the chef, the name of an ingredient is of little help if the attribute of

it is unknown. Hence, looking for and picking ingredients with unknown attributes is of no use. Each ingredient will need to be studied and tested to see weather or not they add anything to the stew. However, if the chef could receive ingredients based on descriptions of necessary attributes such as flavor, color, texture etc., it would be a lot easier. In the developer's case, being seated in front of a data base with reusable components with unknown functionality is of no use, especially if their names are arbitrary and provide little to no description of it's functionality. In the Use Case Marketplace, however, a developer can search for the attributes of reusable code components and find every reusable that matches those attributes.

Since Backstage, as described in detail in 2.4.4, already aims to become a "platform of platforms" and the one entry point for everything regarding services, components and tooling at Spotify, it makes sense to propose an extension of this widely adopted platform, and to have it host the Use Case Marketplace as well. Especially since Backstage already have an Explore section, as can be seen in figure 6.1.

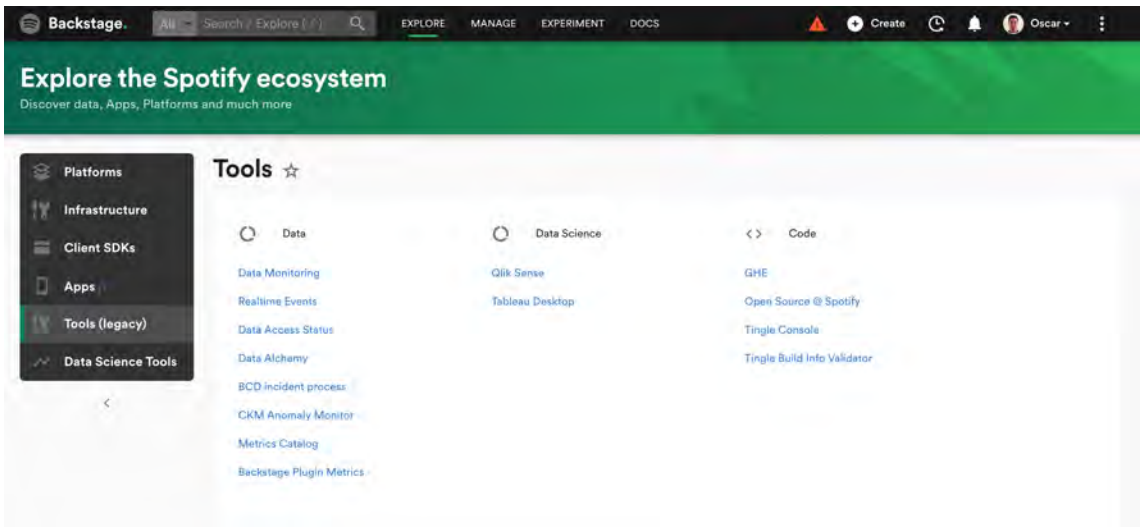


Figure 6.1: Explore section of Backstage as of today.

The Use Case Marketplace can be divided to two parts, namely the *Discovery page* and the *Use Case pages*, both of which are described in detail below.

6.1.1 Discovery page

The use cases' discovery page is where a user will land when clicking on the new "Use Cases" list item in the sidebar navigation of Explore, see figure 6.2. This is where users will discover the different use cases available, displayed as cards, by searching for matching filters or text strings, represented as tags, using the search and filter section.

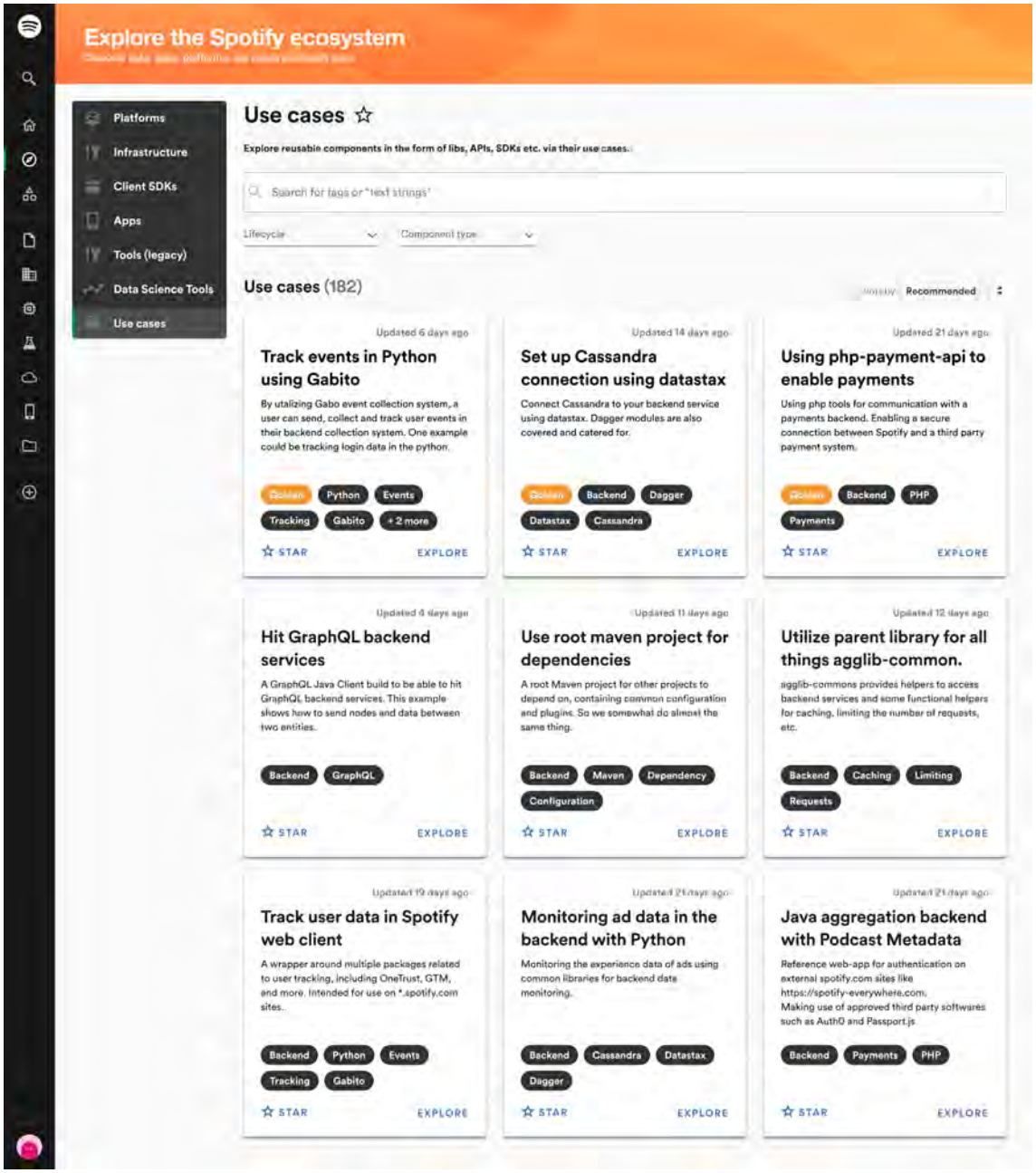


Figure 6.2: The use case discovery page.

6.1.1.1 Search & Filter Section

The search & filter section is located on top of the discovery page under the header, and offers two ways for a user to narrow down the number of results shown; either by using the *search bar* to find tags or by specifying filters in the *filter drop downs*, located below the search bar. See figure 6.3.

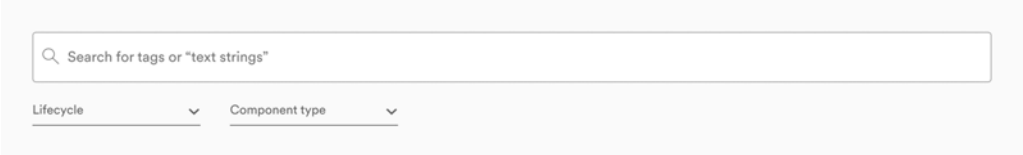


Figure 6.3: The two filter drop downs under the search bar.

Search Bar

The search bar is the fastest and most powerful way of narrowing down the results. As soon as user has typed a letter, the system will suggest a number of auto-complete choices of tags. These auto-complete choices will update when additional letters are added and are sorted based on which tag would result in most hits, see figure 6.4. Through the search bar, a user can either search for pre-defined tags that correlate to tags within the use case cards or for a general text string tag that queries the use case titles and descriptions. The pre-defined tags have been curated manually and are specified by the owners and developers of a given use case, allowing the tags to have a more explicit relation to the it. To make the distinction between the two tags clear, there is an outline of a tag and a plus sign surrounding the pre-defined tag options and quotation marks for the text string, as well as a small caption to the very right on each option clarifying the difference.

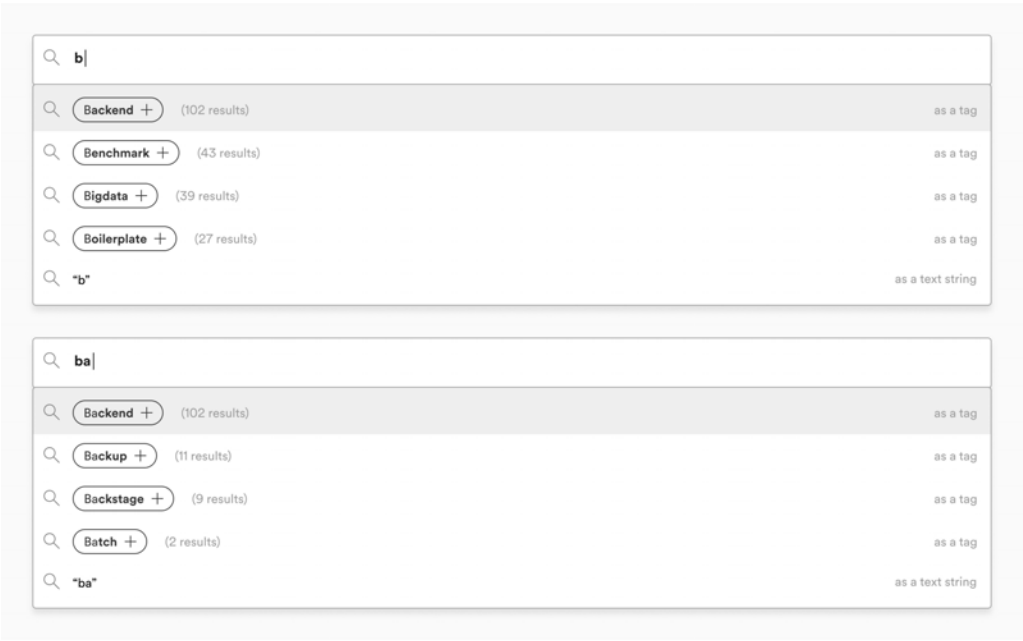


Figure 6.4: The difference between searching for tags on "b" and "ba".

The system has been designed to prioritise pre-defined tags over text string search tags, meaning that the string search option will be presented at the bottom of the auto-complete list of choices, see figure 6.5. The reason for this is the explicit relationship the pre-defined tags symbolise. Compare this to a text string search tag that picks up everything matching the specified string and has no guarantee of being as tightly connected to the use cases as the other tags.

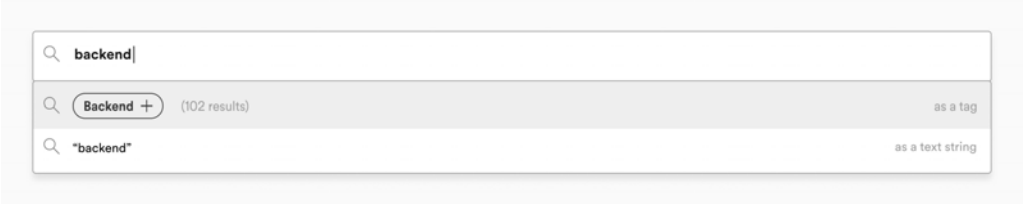
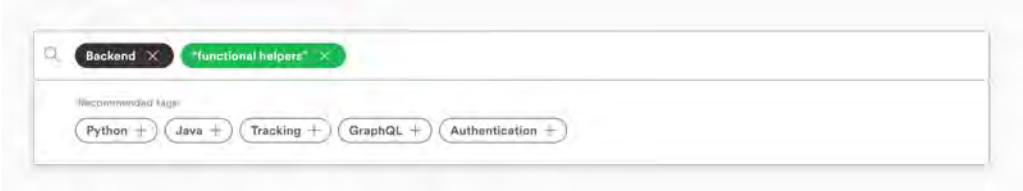


Figure 6.5: The option of either searching for "backend" as a pre-defined tag or text string tag being shown at the bottom.

To differentiate between the two different tags when active, the tags have different colors, see figure 6.6a, each of which are connected to how they are being displayed on the cards. The pre-defined tags are black in the search bar as well as on the cards, whereas the text string tag and the matching string part on the card that yields the match are both highlighted in the same green color, see figure 6.6b.



(a) The two different tags next to each other.

(b) The card, with a highlight of the string that matches the text string tag.

Figure 6.6: The two active tags and their communicated connection to a use case card.

As soon as one tag is added, be it a pre-defined tag or a text string tag, it will appear in the search bar to indicate it being active. These will stack up in an "and" behavior as more tags are added, meaning that the results shown satisfy all the specified tags. Once a first tag is added, it is possible for the user to start typing to find a second tag to add or to choose any of the new recommended tags that appear, see figure 6.6a. These recommendations are shown based on their occurrence with the already added tags. If, let's say, a "Backend" tag is added, "Python" will show up as a recommendation since these two often appear together in searches and in cards. To remove a tag, a user clicks on the "X" on the right hand side of the tag.

To cater for expert users, the first option in the list of the auto-complete choices is marked as active by a greyer shade of its bounding box, meaning that once a user presses enter on the keyboard, the first tag option will be added as an active tag. This allows for a fast search behavior without having to rely on the mouse or trackpad once having started to type, see figure 6.4 and 6.5.

Filter Drop Downs

The second option for a user looking to narrow down the number of results is to use the filter drop downs. Each drop down houses filters connected to that specific drop down category that can be checked or unchecked using check boxes, see figure 6.7. By default, every check box is checked to show as many results as possible. In case a user wants to remove certain types of use cases, for instance experimental ones, it is just a matter of clicking the "Lifecycle" drop down and unchecking the Experimental checkbox.

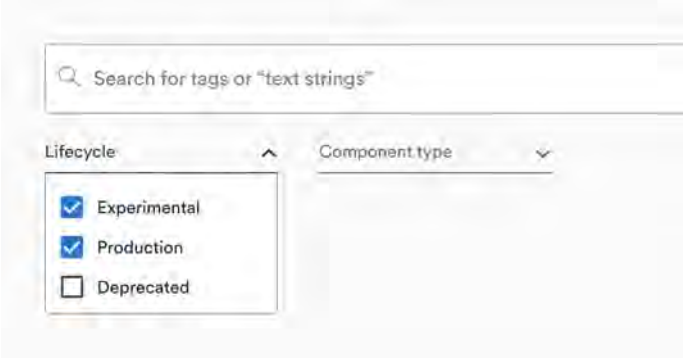


Figure 6.7: Check boxes to check or uncheck currently active filters.

6.1.1.2 Use Case Cards

As aforementioned, each use case is represented as a card on the discovery page, see figure 6.8. Given the limited space and the wish to not overload the user with information, each use case card only contains a headline, a short description, its corresponding tags shown as chips and an indication of when it was last updated. Some cards, representing a use case that holds great standard and are perceived as best practice, i.e. reliable and stable, can be rewarded with a "Golden" chip by Spotify's infrastructure teams. This is a way for the Spotify organization to

surface certain use cases and verify their validity and usefulness. If a technology behind a use case becomes deprecated, this will be clearly communicated by a red "Deprecated" chip and a warning triangle next to the title. The same deprecation warning triangle will also be shown next to the title in the use case page in case a user still decides to explore a deprecated use case.

On the bottom of the card, there are two buttons: "Star" next to an index of a star, and "Explore". By clicking on "Star" the use case will be saved to the user's Backstage homepage. "Explore" simply takes the user to the use case page, described in 6.1.2.

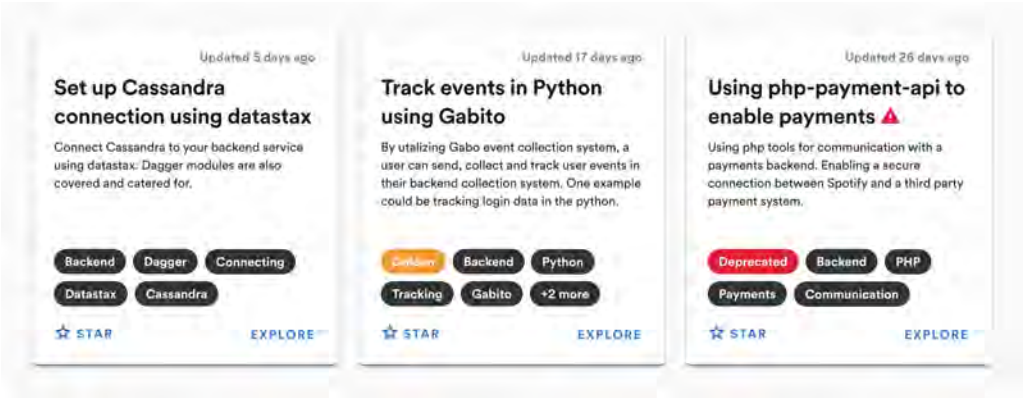


Figure 6.8: Three different use case cards.

By presenting the use cases as cards with a small description, it is possible for users to quickly scan the use cases whilst still being able grasp the gist of them. From talking to developers, the date of last update is an important factor to consider when evaluating if a use case should be explored or not. For this reason, this metadata point is shown on the top right of each card.

By highlighting the tags connected to each use case, the connection to why it is currently being shown is clear and it is also possible for a user to get inspiration of additional tags to add. In a case where the active combination of tags yields zero results, the user is presented with an option that will assist the user in how to find use cases with potential proximity to the current query. For instance, a user can be recommended to exchange one of the current active filters, as show in figure 6.9.

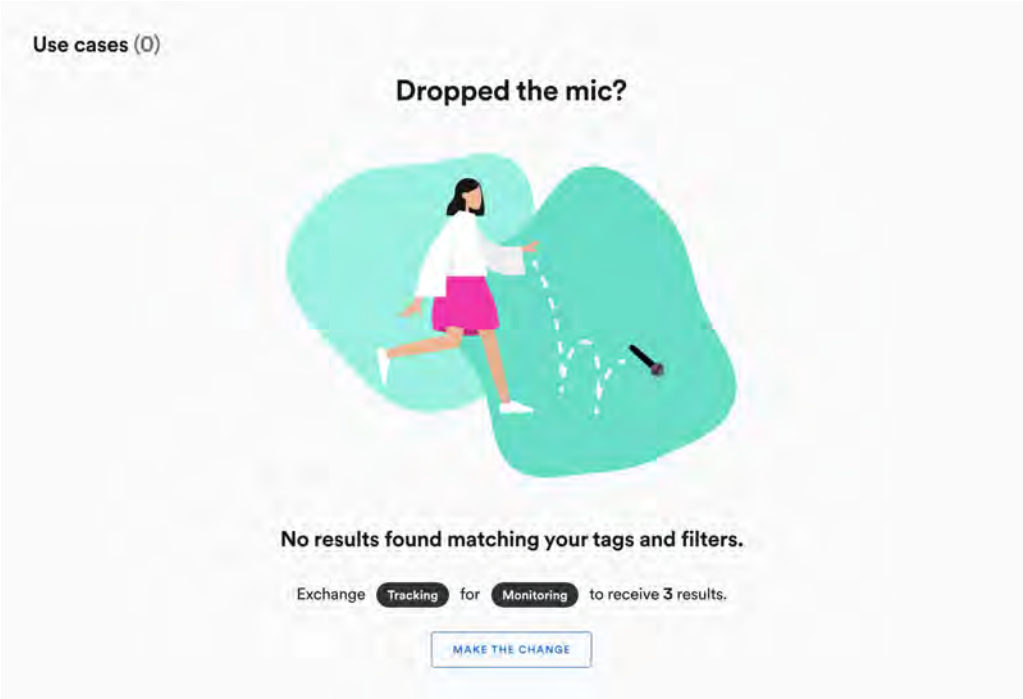


Figure 6.9: The system trying to aid a user's recovery.

6.1.2 Use Case Page

When clicking on a use case card, the user will be taken to the use case page, see figure 6.10. Here, in the different tiles of the page, it is not only possible to find out more information about the use case but a user is also able to see examples of the code and how to get started with it. Since the use cases need metadata to be populated, how this data is proposed to be collected is explained under Source, see 6.1.2.2.

6.1.2.1 Tiles

As briefly touched upon, the use case page is divided into a number of different tiles as a way to group the information and make it easier to digest for a user.

Header

The header holds the title of the use case as well as information regarding the date it was last changed, the number of active users of the technology, the current status in terms of life cycle, e.g. experimental, production or deprecated, as well as the owner.

Description

The *Description* tile essentially contains the same information as shown on the use case card. In case a description is longer than the character limit of the card, the full description will be found here. The description is a way for the owner to describe the

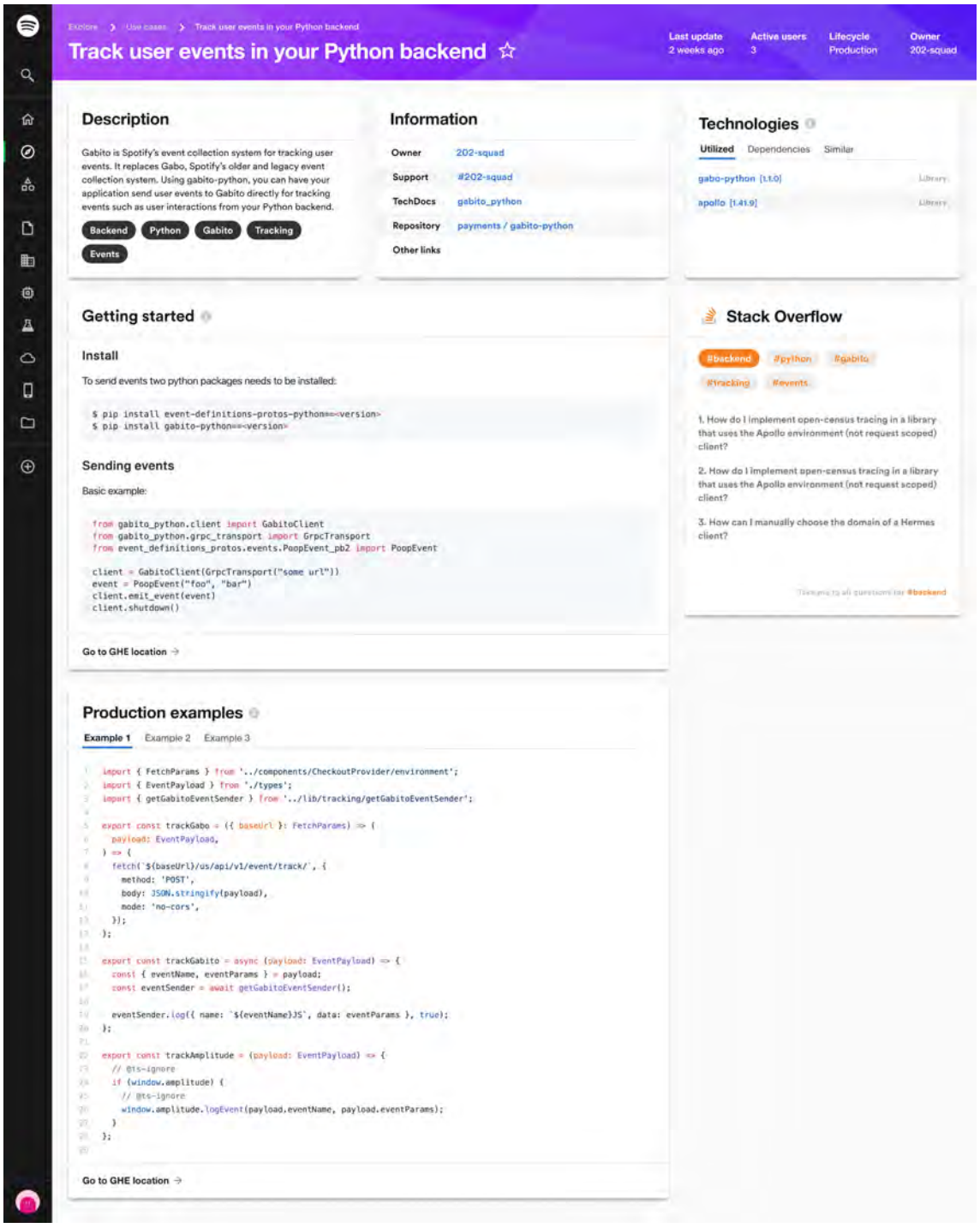


Figure 6.10: The use case page as a whole.

use case in human understandable words. This is also where the user, once again, is presented with the tags connected to the use case.

Information

In *Information*, the user will find useful links to the owner, i.e. a squad or an individual contributor, a shortcut to the support channel in Slack if applicable, as

well as links to more information about the use case and technical documentation linked to the technology behind it. There is also a link to the GitHub Enterprise Repository where the use case lives. More about this under Source, see 6.1.2.2.

Technologies

Under *Technologies*, there are three available tabs, namely: *utilized*, *dependencies* and *similar*. Under utilized, links to the underlying technologies of the use case will be listed. In a case where the technology highlighted in turn is in need of any supporting technology, e.g. another library or SDK, this will be listed under dependencies. The similar tab concerns potentially interesting technologies that not necessarily are utilized in this particular use case but have similar functionalities or use cases. Since this is not something an owner always knows, this section will potentially be left blank at times.

Stack Overflow Enterprise Connection

In the *Stack Overflow Enterprise* tile, users are able find questions & answers related to the tags for each use case, therefore potentially addressing issues with the use case or the technologies supporting it. A user can choose which tags to filter on based on their own preference. By clicking on any of the questions, the user will be redirected to the corresponding question page in Stack Overflow where it is possible to also write a new question or comment. The Stack Overflow tile therefore creates an alternative source of help to the support channel in Slack, mentioned earlier.

Getting Started

Getting Started is aimed at helping users to get started and to try out the technology and the use case themselves. Copy friendly code snippets to paste and run in the user's own IDE is mixed with small annotations and explanations.

Production Examples

The purpose of the *Production Examples* is to showcase real life examples of the technology being used in action, in the production code. Despite the risk of becoming lengthy and harder to understand, it still allows for a user to get more context around the technology. Different examples can be accessed via tabs on top of the tile.

6.1.2.2 Source

A use case is made up by information from a markdown file existing in the code repository of the technology used, see 6.1.2.1. In a case where a use case is built upon a single library, SDK, or API, the UseCases.md file would exist in that reusable's code repository. If the use case, however, is built upon an example app or hack day project, the markdown file would be hosted within that project's code repository. This allows the information within the use case to be tightly tied to the technology that supports it. Any changes to the technology would consequently affect the repository and also the markdown file, preventing the use case from going stale

or out of date. Having the information live in GHE also allows any developer at Spotify to propose a change to the document and furthermore use case, making the information crowd-sourced and even less prone to staleness and deprecation.

The pre-formated markdown file is divided into sections, corresponding to the different tiles on the use case page, e.g. the first section would be the title of the use case, the second one the description etc., see figure 6.11. Some information and metadata such as owner can be taken directly from information already existing within a yaml file in the code repository.

Figure 6.11: An example of how the UseCases.md file could look like.

6.2 User Experience Factors

The final 15 User Experience Factors are divided into the two areas of discoverability, namely *Discovery* and *Interpretability*. Excerpts from the design concept presented in this master's thesis are used to showcase examples of how this could be done within a discoverability system.

6.2.1 Factors for Discovery

Factor 1:

Make browsing of items possible.

To grant an optimal discovery process, giving the user the option to browse either all or a selection of the available items is crucial. Browsing creates opportunities for inspiration and serendipitous discovery where a user spontaneously can discover an item of interest.

As shown in figure 6.12, this could be done by having the items in the discovery section of an enterprise software being presented already from the start. Compared to having to enter a tag or text string to be given results.

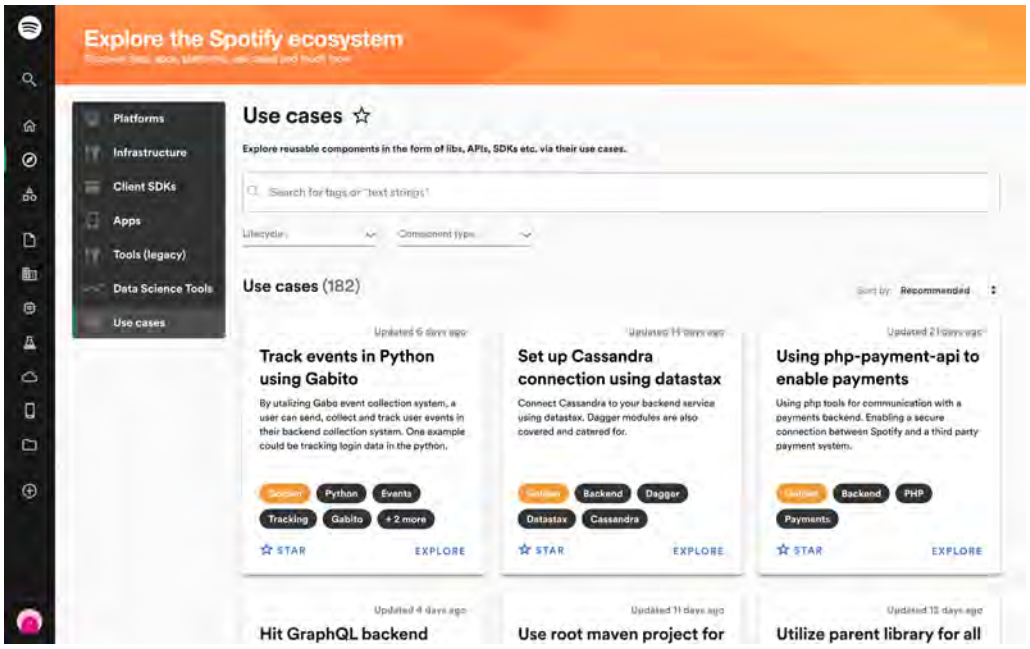


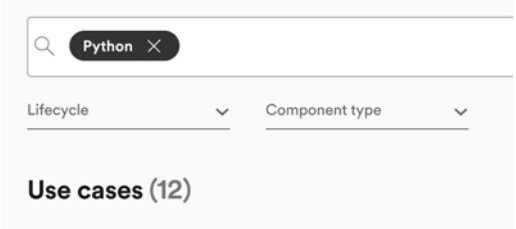
Figure 6.12: The discovery page of the Use Cases Marketplace concept where items are browsable from the start.

Factor 2:

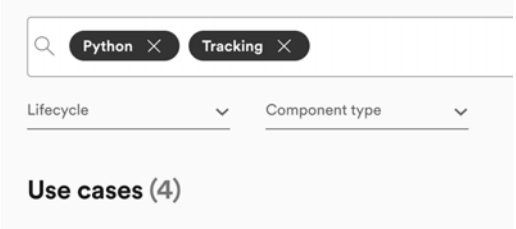
Allow for perceived effectiveness during the discovery process.

As a user looking to discover an item, it is important to feel progression from steps taken. From a system's perspective, it is therefore crucial to communicate and update the results shown as more filters are added.

By consistently showing the number of results a given set of tags yields, and how this number updates as soon as they are changed, is one way of catering for this factor, as can be seen in figure 6.13.



(a) One tag retrieving 12 use case results.



(b) Two tags and 4 results remain.

Figure 6.13: The number of results, as well as the displayed cards, updates as soon as one more tag is added.

Factor 3:

Consider the entry point of the user.

Understand what information and knowledge a user enters the discoverability journey with and how to utilize that to present relevant search results and to aid in their discovery.

In the Use Case Marketplace, the user is be presented with a couple of recommended tag options when the search bar is first clicked, see figure 6.14 This presents common tags for the user to start narrowing down the results with. This way, the user gets assistance on how to start interacting with the system.

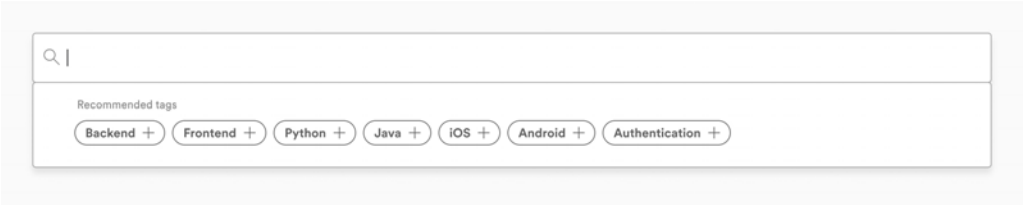


Figure 6.14: A user is given a few recommended tags to make the threshold of starting an exploration as low as possible.

Factor 4:

Allow for alternative ways of discovery by making use of metadata.

Users have different preferences in terms of procedure in a journey to find an item; some prefer searching with text strings whilst others prefer to start by adding filters based on metadata. Regardless of which, it is important for the system to cater for several ways of discovery, not forcing a single option.

In the Use Cases Marketplace, the user have the option to filter the cards shown, on top of being able to add tags, to limit the results to better suit the user's need, see figure 6.15.

Figure 6.15: The ability of changing the Component type filter being showcased.

Factor 5:

Assist users in finding the "blessed" choice.

When looking for items in an unknown vast territory, making a choice can be cumbersome and difficult. To propel the user into taking action, and to furthermore assist the user in finding a recommended or supported item based on any domain specific criteria, recommendations is of great help. Marking, or in other ways communicating selected or supported items will be one criteria that the user can utilize as a lever or metric when making a decision in the discoverability journey.

The "blessed" or, by the company, recommended choice could be communicated by a unique, separate tag. In the Use Case Marketplace design, the Golden tag in the use case card serves this exact purpose, see figure 6.16.

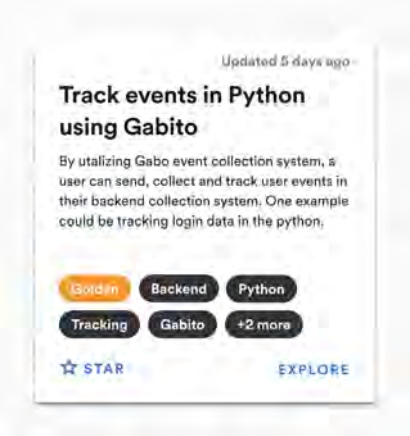


Figure 6.16: A use case card marked with a Golden tag to indicate its validity.

Factor 6:

Allow for saving items

When encountering an item of interest, there needs to be a way to save that item for later. The lack of time or attention might force a user to proceed when multiple items of interest are shown. Having the ability to save an item for retrieval later is useful. More importantly, the retrieving process of an item needs to be easier than the discovery process, or else, no time would be saved. Done right, being able to preserve items lightens the cognitive load and enhances the discoverability process.

Throughout Backstage, the ability to *star* items is a widely adopted feature. In the proposed extension, this feature is supported via the ability to star use cases directly from the cards or in the header on the use case page, see figure 6.17.

Figure 6.17: The header of a use case with the built in ability to star it.

Factor 7:

Make users feel in control of their discovery process by communicating the effect of their changes.

When interacting with a system aimed to aid in discovery of items, clearly communicating cause and effect of certain actions in the system, e.g. adding filters or changing text strings, is important for a user to feel in control of their own discovery journey. If the system fails the user in this, the trust of the system and the items residing in it is ruined.

As an example, one way to communicate the effect of an action is to highlight the resulting number of matches before that action even is executed, see figure 6.18 where a user of the Use Case Marketplace gets this information next to the different options in the search bar.

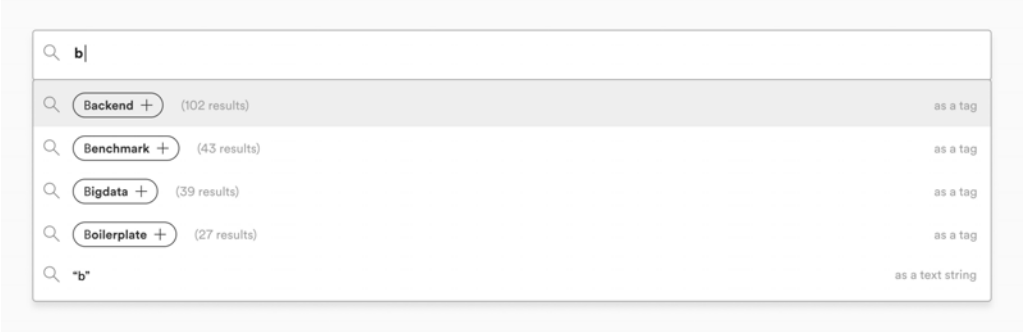


Figure 6.18: In the search bar next to the tags, a user is able to see how many results adding that tag will yield.

Factor 8:

Assist users' discovery process by suggesting related filters.

To further assist the users in finding the right item, the system should suggest additional related filters that might return a more precise list of results. The suggestions could either be based on search patterns from previous users or occurrence in relation to already specified filters. By having the system suggest related filters, the user can find filters they normally would not have used and potentially sparks a more exploratory discovery process.

In the Use Case Marketplace, this factor is manifested through the suggestion of additional related tags to aid in the user's discovery, see figure 6.19.

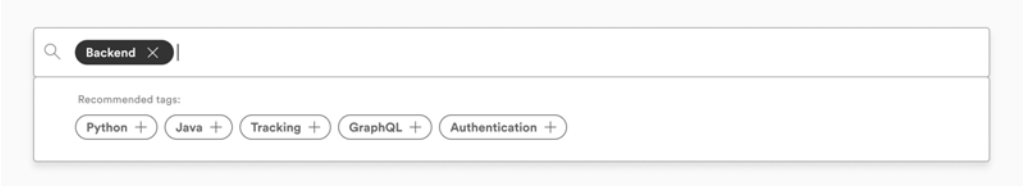


Figure 6.19: Recommended tags being presented based on the initial tag.

Factor 9:

Allow for further exploration of related items.

When a found item, seemingly satisfying the user's need at first, turns out to be a mismatch, the system should allow for further discovery of human or system marked related items. Instead of falling short, it should invite for an extension of the discovery journey.

To further assist user's exploration of related items, the Technologies tile in the use case page and, more specifically, the *Similar* tab inside of it, presents a number of technologies that may be of interest for the user, see figure 6.20.

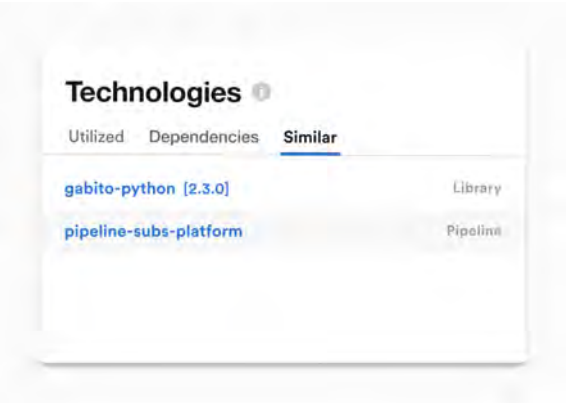


Figure 6.20: Similar technologies presented for the user to extend the discovery journey.

Factor 10:

Assist users when reaching a dead end.

When in a discoverability journey, there is a risk of reaching a dead end in terms of the system not finding any proper matches for a search. In a situation where the user will be rendered with an empty query call, there is an inherent risk of the user feeling discouraged and either abandoning the discovery journey or loosing faith in the system. To alleviate this, the system should present suggestions on how to proceed forward.

In the design concept, this is done by having the system help the user when they, arguably, need it the most; when their search yields zero results, see figure 6.21.

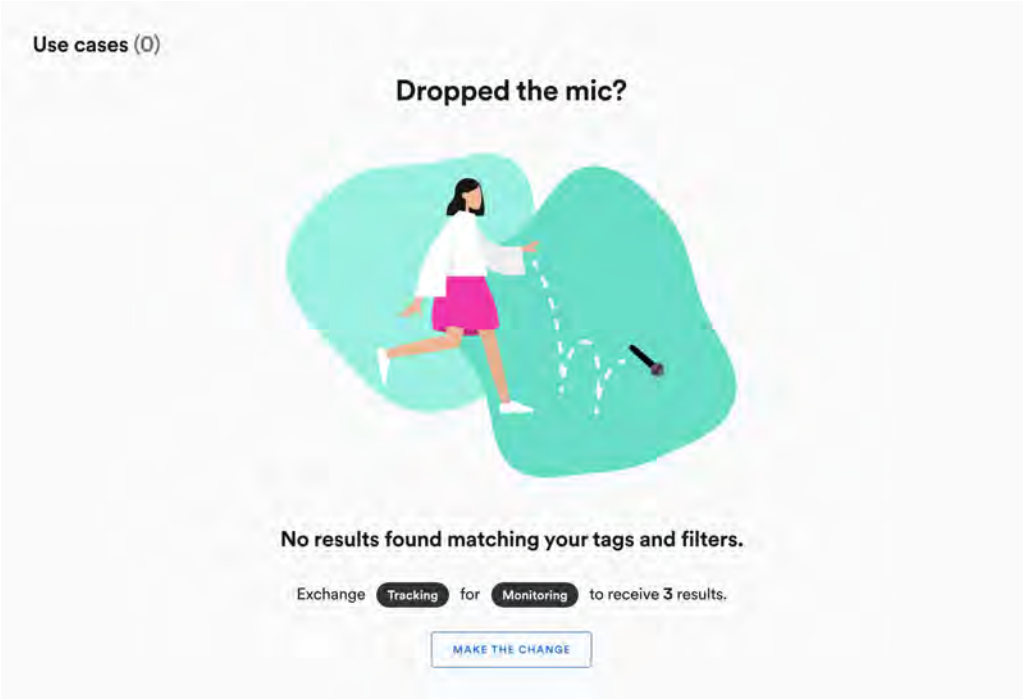


Figure 6.21: A suggestion of how to recover from a state where a user no longer have any matching use cases.

6.2.2 Factors for Interpretability

Factor 11:

Allow users to evaluate the quality of the items discovered.

Closely connected to the evaluation of an item is its quality. A system therefore needs to incorporate ways for a user to be able to evaluate the quality of an item. Preferably at an early stage to not risk having the user go down the road of exploring numerous items without success.

As an indicator of quality, the number of days since an item was last updated is shown on top of a card of a use case, see figure 6.22. This way, a user will get an indication if the use case is out of date or not.

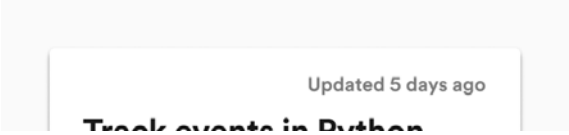


Figure 6.22: The number of days since an item was updated, displayed on top the use case cards to aid the user's evaluation of quality.

Factor 12:

Allow users to easily interpret the purpose of an item.

As much as quality is important to consider in the act of evaluating an item, equally important is the purpose when in the act of choosing to explore it in the first place. As with Factor 11, this would preferably be supported by the system at an early stage of the discovery.

To support users' interpretation of a use case in the Use Case Marketplace, a title and small description is displayed directly on the card, see figure 6.23. This way, the user do not have to enter each use case in order to determine if it could be useful or not.

Figure 6.23: An example of what a title and a description of a use case card can look like.

Factor 13:

Display examples of the items in their proper contexts.

To assist in the interpretability of an item, the system should include the ability to present items in their intended context. This will help the user to create a mental model on how the item's creator or maintainer would suggest the item to be used, implemented or in other ways consumed. By letting the user familiarise with the context of the item, it is arguably easier to understand its full potential and thereby aiding in the interpretability of it.

For a user to get a better understanding of how the technology behind a use case is being used in the actual production code, the *Production example* tile inside the use case page showcase a few examples of this, see figure 6.24.

Figure 6.24: The Production examples tile showcasing real production code utilizing the technology behind a specific use case

Factor 14:

Allow users to access corresponding information or documentation for a given item.

To further assist a user in their understanding and interpretation of an item, the system should provide the user with links to other relevant information sources and documents regarding the item.

In the *Information* tile of a use case page, the user can access technical documentation regarding the technology to derive an even deeper understanding of it, see figure 6.25

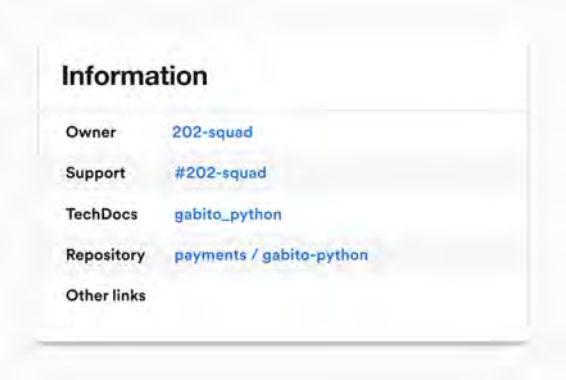


Figure 6.25: Useful links in the Information tile for a user looking to get an even better understanding of a use case or the technology behind it.

Factor 15:

Strive for transparency and assist in scrutiny.

In order to instill trust in the item shown, a system aiding in interpretability should assist its users in finding any kind of information deemed necessary. This means that the system should either accommodate a multitude of information touch points for the user, or supply endpoints by which the user can leave the site to find more information. Transparency with the system's intention as well as with any information that exists about items presented help with building trust and increases the notion that the system is all encompassing.

In order to assist users' analysis and scrutiny of a use case and its perceived usefulness, each use case's tags are connected to the corresponding Stack Overflow Enterprise tags. The top three questions and concerns for each tag that pops up in Stack Overflow Enterprise are therefore also shown on the use case page, see figure 6.26.

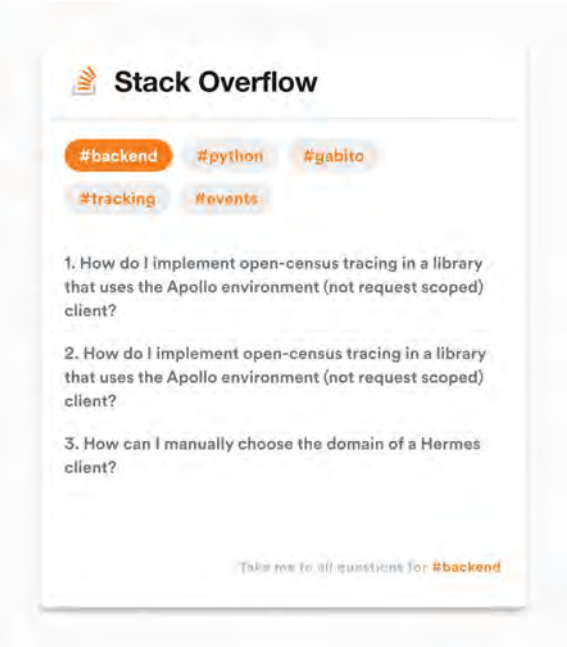


Figure 6.26: Stack Overflow Enterprise is connected to the use cases to easily be able to further scrutinise a use case.

7

Discussion

In this chapter, a discussion around the thesis project will be held. It will incorporate the process undertaken to execute the project and results in terms of the final design concept and the user experience factors. Potential ethical issues will also be touched upon, as well as some suggested areas for future works.

7.1 Process & Execution Discussion

Limited Software Development Knowledge

Given the thesis duo's limited knowledge of software development practices in general and at Spotify in particular, a considerable amount of the user research was dedicated to get on par and understanding these. This arguably had to be done in order fully empathize with the users and to understand the developers' lingo, but it potentially had some implications on the final results. If more time could have been spent on the later parts of the design thinking phases, more guidelines could have been extracted and the design concept could have gone through yet another iteration, refining it further.

Individual Contributors vs. Teams

The selected focus of talking to individual contributors rather than whole teams of developers probably influenced the outcome of the project as well. By talking to developers from different squads and tribes, the duo could cover a wide range of preferences and habits since these developers, to some extent, acted as proxies for their teams. However, if the duo would have focused on analysing a couple of teams more closely, the insights would probably not have been as diverse, and the problem addressed in the end would, perhaps, have looked different.

The COVID-19 Pandemic

Due to the COVID-19 pandemic of the spring of 2020, Spotify, like many other companies, had to take precautionary measures. This resulted in a global lock down of all Spotify offices and a call for every individual contributor to Work From Home (WFH). This meant adjustments needed to be made to some of the planned work.

Weekly meetings with company mentors and university supervisors were moved to online conferencing tools. Instead of hosting interviews, workshops, focus groups, design critiques and user tests live at the office, these needed to be done remotely. It proved more difficult for participants of these sessions to build upon each other's sentences due to the inherent audio lag from the video conferencing tools. Not gathering everyone in a room for group activities did make it more difficult to set a proper atmosphere and tonality for the sessions, which may have resulted in less motivated participants. The effects were, however, both good and bad. Good, in the way that the participants were always reached at the comfort of their own home, at their own WFH work station, which might have put them in a more focused and appropriate contextual state. It also permitted the participants to turn on music in their own headphones during an ideation session if in the mood of doing so.

However, as aforementioned, this was also fuel for problems. Since the developers were at home, there was no way for the facilitator to decide what kind of environment the session would take place in. This meant that the participants could have a child or pet run between their legs whilst participating in a workshop, or having to stay quiet due to a partner working from home in the same room. It also meant that the participants were somewhat in charge and could mute their microphones, turn off their cameras, turn on music, read emails or leave the meeting at will, without the control of the facilitator, making the situation different from a regular workshop. This meant that trust between every one involved in each session had to be bigger, since there were less ways of checking on one another. Having and communicating a clear structure also became more important to herd the participants in the right direction. Despite this, setting up workshops, focus groups and design critiques remotely made it easier as a facilitator to book and execute sessions since this took place online. Also, whilst running the sessions, distributing the word to the participants was easier in comparison to the live groups, since there were more natural pauses in the conversations.

Although challenging, Spotify employees are used to remote ways of working due to its global office sites. It meant that all infrastructure and tooling, like video conferencing tools, collaborative design tools and communication software were already in place and widely adopted. This was also the case for internal processes and the ways of working together, which made the whole WFH situation less severe for this project compared to if it would have been done at another company.

Limitations in Tests

During this thesis, a multitude of tests were executed. Either as design critiques, see 4.1.3.7, where early sketches of solutions were shown and addressed, or as usability tests, see 4.1.5.1. However, the usability tests conducted in this project was leaning more towards qualitative user experience tests. Instead of testing the usability and understandability from an interaction design point of view solely, the qualities of the Use Case Marketplace were rather tested in order to evaluate the proposed UXFs the system was built on.

Another important factor to consider, that probably affected several user experience factors as well as the design concept, was the execution of the usability tests themselves. Like any usability test with a mock-up, the inherent limitations in interactions and exploration makes it hard to fully evaluate all aspects of a concept. Taking the time to make the prototype fully interactable, or even coding it, would of course have allowed for a more representative evaluation of the concept that fed into the user experience factors. However, the potential gain versus the cost in time in such a scenario would not motivate doing it. In the case of this project's prototype it was limited in the way that the user could only search for a pre-decided set of tags to narrow down their search. Furthermore, these tags had to be entered in a specific order. Despite including the possibility to find the tags either through categories or by manually typing in the tags using the keyboard to make it feel more real, it is impossible to know if this search pattern would be the actual behavior of a developer encountering the system for the first time. Would the developer think in terms of tags or rather search in a more natural language manner as is the case with other search engines like Google? It is only possible to speculate at this stage. What can be said, however, is that even though some testers seemed to enter the test with a mental model not fitting the prototype perfectly, after using the search bar for the first time, they understood its underlying functionality and could continue using it without any difficulties. The guessability could therefore benefit from being addressed to a larger extent whilst the learnability arguably already is sufficient.

On the positive side of having to run the usability tests remotely, the testers were allowed to stay in their right context, in accordance with the ethnographic research approach chosen for the project. Compared to a regular usability test, which in most cases is conducted in a semi-staged laboratory environment with a different computer and set-up than the subject is used to, the participants in this study got the chance to use their own equipment, browser and office setting. Albeit whilst sharing their screen to the thesis duo in a video conference call.

Whilst the medium fidelity prototype was well perceived during the usability tests and received great responses in the questionnaire, see 5.8.4, it was not compared to the current systems available. A control group could have been given the exact same scenario and goal of finding a specific reusable but with access only to the tooling in place today. However, this would most likely have showed how this is not possible to do today without the help of other colleagues. A control group test like this could have provided the thesis with a stronger case, but was not prioritised in interest of time.

7.2 Design Concept Discussion

The final design concept, addressing the productivity bottleneck of discoverability at Spotify, is a Use Case Marketplace, surfacing reusable assets and code by letting developers search for their use case as a discovery mean. Reusing previously built components allow developers to save time by not having to reinvent the wheel. Previous research show how developers will spend time on finding these assets if

they know where to look and how to utilize them, see 2.2. The final solution of the Use Case Marketplace also takes into consideration all four stages of help seeking, namely *search, select, integrate* and *verify*, see 2.2.2.

The rationales for developing this concept and placing it within Backstage are plenty. Firstly, a place to search for these kind of assets are missing from the internal Spotify tech ecosystem. Backstage is currently being used when trying to do this today, but has only limited functionality when it comes to discovery and interpretability. Secondly, the Backstage platform is widely adopted by the developer clientele, where a majority utilise it daily. Since TechDocs and the current component library exist within this platform, it make sense to host the Use Case Marketplace there as well. Thirdly, referring back to the quote in Background, see 2.1.1, software tools are not a silver bullet but should work as human aid that amplifies the positive impact of skill-full teams. Use Case Marketplace is perceived and proposed to be such an aiding amplification tool.

For the Use Case Marketplace to be populated with many, yet relevant, use cases, it was necessary to find a way for them to be created easily and without a major hassle to keep them up to date and well maintained. Since every reusable, code component and technology already has its own repository in GHE with markdown files and yaml files, it was proposed to add another markdown file by default. Whilst not being ideal to add yet another file to maintain, the potential gains transcends the costs. The extra time needed to create and maintain a use case by its owner squad will save time to numerous others. In the long run, this is estimated to enhance productivity by saving time for a majority of individual contributors, thereby allowing for more output with the same amount of input, see 2.1.2.

One could argue that one of the bigger drawbacks of having the use cases connected to the code repositories is that it limits the user base to developers, and developers only. If the system allowed for an alternative way of creating use cases, other contributors like Product Owners or Engineering Managers and disciplines such as design, marketing and HR would be able to extend the knowledge database. Despite not being as applicable when the end goal is not to find a reusable code component, the *Getting started* section of the use case page could still be valuable as an how to guide of certain, Spotify specific tasks.

The placement of the marketplace in the Explore section of Backstage makes a lot of sense, but it is still not fully evaluated. Some developers expressed how they did not even know that this section existed, nonetheless what it is used for. It may be the case that the concept would benefit from being a stand alone service, or connected to Spotify Code search 2.4.6 – which already is considered as a place to find code today, but this was not prioritized or tested within the time frame of this thesis.

Since this project was solely conducted at Spotify, the generalizability of the solution can definitely be questioned. For any other major tech company with huge code stacks, the concept could definitely be valuable for the exact same reasons as to why it is valuable for the developers at Spotify. But taken into account a small, start-up

business with every developer sitting in the same room, in comparison to Spotify's thousand plus developers, a solution like this would most probably be redundant and a waste of time. In this case, simply asking the neighbour colleague may be a more viable solution. However, incorporating a use case marketplace already at an early start-up phase and making it part of the development processes may save valuable time and resources in the future, when the company is growing in size and the aforementioned discoverability issues start to surface.

Another argument that could be made is that a solution like Use Case Marketplace might inherently damage the vital networking culture at Spotify. If a solution could be found in the Use Case Marketplace, there is suddenly no need for one to ask colleagues for help, neither on Slack or in the office. However, a counter argument could be made that use cases surface teams and individuals, otherwise unbeknown to the user finding it. Through means such as the *Stack Overflow Enterprise* tile and the possibility to reach out to owners and users of the technology, one could argue that use cases has the chance of enhancing communication. If that is the case, there is a great chance that the understanding of the company structure and who owns what becomes better and that the feeling of siloed teams and organizational fragmentation lessens.

7.3 User Experience Factors Discussion

As a mean of providing an answer to the posed research question, 15 User Experience Factors (UXFs) were formulated and iterated upon over the course of the thesis. These factors aim at taking a broad approach regarding how to design a system for discoverability in an enterprise software context, in order to account for multiple types of discoverability systems. The UXFs are generally written in a proposing language, suggesting that the user of the factors apply them in the way that makes sense to their user group and context. This is extra important with regards to factors 2, 11 and 15 where the context and situation for the discoverability system will have a great impact on the outcome of these UXFs.

As a UX designer, the UXFs could be utilized as, or for:

- **Heuristics**, meaning that the UXFs could be used as a rule of thumb regarding how to design a discoverability system, taking into account both discovery as well as interpretability.
- **Evaluation**, where the above heuristics could be utilized to evaluate and judge an existing system's or interface's discoverability.
- **Idea generation**, where the UXFs could be used to create HMW questions, see 4.1.2.4, to ideate against or as food for thought to spark a creative process.
- **Checklist**, to go through when designing systems for discoverability to make sure no user experience need is overlooked.

As aforementioned, it is proposed that the application of the 15 user experience

factors will lead to the creation of a system with potent discoverability qualities. The opposite could be said for product or systems that disregard the recommendations made through the factors. The inverse to a system built on the UXFs from this report, would inherently be the discoverability system currently active at Spotify. Plenty of times during the research, developers have exclaimed how it is impossible to find a reusable asset without knowing its name or place. There is no promotion or assistance of browsing reusables to allow for serendipitous discovery and furthermore, it is perceived difficult to know how reliable a reusable is to assess if one dares to incorporate it in one's code solution. Because of this, developers refrain to ask colleagues for help or simply skip the process of looking for reusables in the first place.

As the UXFs has formalized in parallel with a research and design project, they have undoubtedly been influenced greatly by this exact fact. For example, the majority of UXFs take a general stance to what a discoverability system should consist of whilst a few factors depend on the fact that the search system makes use of tags or has a recommended system algorithm connected to it. One can argue that the user experience factors would look different if a different path had been chosen for the project and, more importantly, if the project would have been carried out at a different company. The proposed 15 factors have mainly been based on insights from the qualitative research performed by the thesis duo. They are based on findings from interviews, focus groups, design critiques, dialogues, usability tests as well as benchmarking sessions. An argument could be made that the factors would look different had the qualitative research been done on other participants. A counter argument could be made that information has been collected from more than 70 employees of various tenure, professional backgrounds and organizational positions over the course of the project. The insights may lack certain perspectives, but is very well rooted in the individuals building up the organization of Spotify. A majority of these employees come from previous employments at other technology firms of different scale. They bring with them the overall notion of how to work with software, and these people and their practices are ultimately what creates the culture at a company like Spotify. Even though Spotify is known for its unique autonomous team setup, see 2.3.1, they are made up of general software developers with general problems. Therefore, the UXFs presented in this thesis, which are based on research on Spotify employees, is deemed to be general enough to be used in a separate context.

With all this taken into consideration, this set of 15 UXFs should not be seen as complete coverage of all aspects of a discoverability system for enterprise software, but rather a bedrock to build upon. To conclude, the UXFs are proposed to work well during experimental design considerations when designing for discoverability systems. They are also considered universal enough to suit a great range of cases. Still, they should not be regarded as a complete and final set, since further work is needed in evaluating them during use, at scale, by different types of discoverability platforms. In addition, specific discoverability systems and their problem areas should be explored in more detail before applying the proposed UXFs since edge cases might exist where the recommended factors from this report will not apply.

7.4 Ethical issues

In regards to ethics, an important aspect to consider is the massive amount of data collected in this project. The data has been collected with the consent from participants and the promise of representing it anonymously. Whilst names consistently have been removed from the text files, the audio and video recordings are still being stored at Spotify's server space provided by Google. The access rights are currently restricted to the members of the thesis duo, but what happens to those restrictions as soon as the thesis workers leave the company remains unclear. In order to further assure the participants' anonymity, the recordings will be deleted from these cloud based servers as soon as the project comes to an end.

During the project, most of the data collected have been transcribed by the third party service Reduct, see 4.1.6.5. Since this is a Spotify approved vendor, utilized by many other researchers at the company, the trustworthiness of the tool and their handling of the audio and video files were perceived as very high. Not the least since user and data integrity is of great importance for Spotify as a whole. One could argue that the safest thing would have been for the duo to transcribe all data by themselves, and whilst this is true, the counter argument would be that the time saved instead could be spent on other necessary tasks and allowed for a bigger sample size and user coverage.

The overall goal of enhancing productivity sounds like a reasonable goal for a modern, industry leading tech-company to have, but it is important to take into consideration at what cost this is achieved, from a developer perspective. To combat the situation, the thesis duo consistently made sure that whatever features were designed fulfilled the requirements of *Ethical Interaction Design*, see 3.4.7. Rather than coming up with ways to force faster work, instead the thesis duo tried to eliminate frictions and inefficient discoverability flows, normally considered as pain points by the developers. This way, both the company of Spotify and the developers would be satisfied.

7.5 Future work

Due to specific limitations such as time constrains, narrowing of scope and ideas selected throughout the design process, many topics were left untouched. Below follows suggestions for future work to be extrapolated upon further.

Whilst the user experience factors proposed in this thesis has been generated and evaluated by the deep involvement of developers at Spotify, additional research is needed in different domains and with a different set of users to thoroughly evaluate the proposed UXFs. The benchmarking done within the scope of the thesis seems to prove the UXFs to be general enough to potentially support other areas than software development but more research on this needs to be done. It is also important to emphasize that the UXFs have not been tested with designers looking to design a discoverability system. It is therefore hard to draw any conclusions on how clear and

actionable they are from a designer's perspective. Whilst this was not within the scope of the project, it is a relevant aspect to consider and something that should be investigated further.

Another topic that could benefit from being explored additionally is related to the creation, maintenance and deprecation of documentation, mentioned in 5.5.2.1. Since this thesis' first interview, developers have alluded to the problem space of having to rely on documentation with ambiguous legitimacy. The vicious cycle of creating and constantly maintaining documentation for the potential good of someone else is robbing many developers on time that could be used to write business logic instead of regular text. This thesis has proposed one solution to this problem, namely to create a new source of information, mainly generated from data already existing within the company. However, this system still relies on developers creating and maintaining some sort of documentation for the proposed use cases to stay valid. Investigating how to, in a more effective way, create high quality documentation without spending valuable developer hours is needed. This area would benefit greatly from a mixed research approach, investigating both the technical as well as behavioral possibilities.

As of 2020, Spotify has put their money where their mouth is and dedicated a whole infrastructure squad to exploring and enhancing discoverability of internal systems at the company. As a final suggestion, looking deeper into how enhancing discoverability can affect productivity at a software company like Spotify is of essence. The decision made during this thesis, to leverage discoverability as a mean to positively impact productivity, was based on the literature researched, qualitative interviews and historical data. The same goes for the issue of code exploration and evaluation that the final concept of a Use Case Marketplace was based on. In short, the quantitative evidence that efficient internal discoverability of reusables positively effects productivity among individual contributors still needs further proofing. Spotify's investment into discoverability is in the right direction and might provide answers to this question.

8

Conclusion

The purpose of this thesis project has been to explore design considerations and factors to be taken into account when designing for discoverability in the context of an enterprise software. More specifically, the aim has been to answer the following research question:

Which user experience factors should be considered when designing for discoverability in the context of enterprise software to enhance productivity?

What started with rigorous user research to truly understand the problems of productivity at the company of Spotify, ended up being a project addressing the issues of discoverability. Once realising many of the topics being discussed pointed towards problems of discoverability in terms of finding the right information, documents, owners and code, the scope was adjusted accordingly. To fully map out the problem space of discoverability, several rounds of semi-structured interviews, focus groups and guerrilla research were conducted, whilst digesting related literature simultaneously to get a more theoretical understanding of the manner. Throughout the project, an ethnographic research approach has been used and a design thinking process employed to allow for empathising with the users in an iterative exploratory design approach. During this process, User Experience Factors (UXFs) have been created, reformulated, legitimized and ultimately manifested in a final design concept.

A total of 49 developers were interviewed or in other ways studied. The data collected during these encounters was broken down and analyzed in a thematic analysis coding manner before being thematized in an affinity diagram. The problem of *Code Exploration and Evaluation* emerged as one of the bigger clusters and sparked a great interest amongst the stakeholders. It was hence decided to move forward with and let this theme serve as the scope of the exploratory design work that followed.

Several iterations of ideation and prototyping were conducted to explore the solution space of the problem. By involving developers and other stakeholders in these phases in a participatory design manner, the technical feasibility and behavioral desirability of the solutions was consistently considered. Having developed three thought-out concepts, each addressing the problem of code exploration and evaluation, albeit

in different ways, six design critique sessions were held to decide which concept to move forward with. The concept perceived to have most potential was one with a marketplace approach to reusable code components. Following additional user studies and stakeholder discussions, this resulted in the idea of searchable reusables that would be discovered via their use cases. Instead of searching for reusables via their name and technical framework which is current practice, the functionality or use case of a reusable is searched for and discovered, pointing to the underlying supportive technology. The reason for this is simple; as a developer looking for a potential reusable, the reusable itself is of little importance. What is important, however, is the functionality that potentially solves the problem at hand. By searching for that, the technology supporting the use case can be found and utilized.

Lastly, the final concept of a Use Case Marketplace was refined and usability tested with six developers before final changes were made. Furthermore, the concept was evaluated against the generated UXFs through a questionnaire and observations from the tests. Finally, the list of UXFs was iterated based on the new insights gained.

The aim of the final design concept, the Use Case Marketplace, is to aid in the discoverability of reusable code components at Spotify. By assisting in the finding of these components, developers might sustain from re-invent the wheel each time they are looking to build new features. In addition, they might not have to ask reoccurring questions in internal forums, risking to waste someone else's time. The new use case marketplace approach removes the need for a developer to known the exact name of a potentially useful component by instead showcasing and letting developers find reusables based on their intended use cases. By surfacing all use cases at the starting page by default, the first ones based on editorial recommendations, the system invites for serendipitous browsing, previously unheard of at Spotify. As a consequence of easing the discoverability of components, the time and efforts saved could instead be spent on other important development tasks, like writing business code. Therefore, by allowing for an improved and frictionless discoverability process, the concept arguably supports and caters for increased productivity at the company.

The final set of User Experience Factors to be utilized when creating an enterprise software system for discoverability is presented below, divided into the two areas of discovery and interpretability:

Factors supporting and enhancing Discovery:

1. Make browsing of items possible.
2. Allow for perceived effectiveness during the discovery process.
3. Consider the entry point of the user.
4. Allow for alternative ways of discovery by making use of metadata.
5. Assist users in finding the "blessed" choice.
6. Allow for saving items.

7. Make users feel in control of their discovery process by communicating the effect of their changes.
8. Assist users' discovery process by suggesting related filters.
9. Allow for further exploration of related items.
10. Assist users when reaching a dead end.

Factors aiding and improving Interpretability:

11. Allow users to evaluate the quality of the items discovered.
12. Allow users to easily interpret the purpose of an item.
13. Display examples of the items in their proper contexts.
14. Allow users to access corresponding information or documentation for a given item.
15. Strive for transparency and assist in scrutiny.

Bibliography

[1] Reduct.Video. URL: https://reduct.video/.

[2] Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts. Technical report, International Organization for Standardization, Geneva, CH, 3 2018.

[3] S. Ålund. Find your way in the Spotify software ecosystem, 2017.

[4] S. Ålund. Platform of Platforms, 2019.

[5] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek. An exploratory study on reuse at Google. *1st International Workshop on Software Engineering Research and Industrial Practices, SER and IPs 2014 - Proceedings*, pages 14–23, 2014. doi:10.1145/2593850.2593854.

[6] H. Beyer and K. Holtzblatt. Contextual design. *interactions*, 6(1):32–42, 1999.

[7] L.-O. Bligård and A.-L. Osvalder. Methodology for prediction and identification of mismatches in the interaction between user and artefact, part 2. Technical report, Chalmers University, Göteborg.

[8] J. Blomberg, J. Giacomi, A. Mosher, and P. Swenton-Wall. Ethnographic Field Methods and Their Relation to Design. pages 123–155. 2017. doi:10.1201/9780203744338-7.

[9] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006. doi:10.1191/1478088706qp063oa.

[10] M. Brian. Google's new 'Material Design' UI coming to Android, Chrome OS and the web, 6 2014. URL: https://www.engadget.com/2014/06/25/googles-new-design-language-is-called-material-design/?guccounter=1&guce_referrer=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAAHA4s3ZgcR4u3f5_NnYsGa4G19m8nFNyr_yqnbZ4lsaJz7wELksSZuz-ni73rWhPmC3hg_CgRjjoNJovtxGkhmAVib72iSg1e-IV8LW96R9pO_YBdel0nmLfIxMCw6zcKatjedchl-KPudpyV-

HTn6yev18Y1YsZx257gYKN0gEP.

[11] V. Brohma. Codesearch. URL: internalspotifypage.

[12] R. Buchanan. Wicked problems in design thinking. *Design issues*, 8(2):5–21, 1992.

[13] A. Cooper, R. Reimann, D. Cronin, and C. Noessel. *About face: the essentials of interaction design*. John Wiley & Sons, 4 edition, 2014.

[14] C. Cousins. What Is Figma? a 101 Intro | Design Shack, 11 2019. URL: https://designshack.net/articles/software/what-is-figma-intro/.

[15] N. Cross. The nature and nurture of design ability. *Design Studies*, 11(3):127–140, 7 1990. URL: https://www.sciencedirect.com/science/article/abs/pii/0142694X9090002T, doi:10.1016/0142-694x(90)90002-t.

[16] R. F. Dam and T. Y. Siang. What Kind of Prototype Should You Create?, 12 2019. URL: https://www.interaction-design.org/literature/article/what-kind-of-prototype-should-you-create.

[17] R. F. Dam and Y. S. Teo. 5 Stages in the Design Thinking Process, 12 2019. URL: https://www.interaction-design.org/literature/article/5-stages-in-the-design-thinking-process.

[18] R. F. Dan and T. Y. Siang. How to Select the Best Idea by the end of an Ideation Session, 2019. URL: https://www.interaction-design.org/literature/article/how-to-select-the-best-idea-by-the-end-of-an-ideation-session.

[19] V. Distler, C. Lallemand, and V. Koenig. How Acceptable Is This? How User Experience Factors Can Broaden our Understanding of The Acceptance of Privacy Trade-offs. *Computers in Human Behavior*, 106:106227, 5 2020. URL: https://linkinghub.elsevier.com/retrieve/pii/S0747563219304467, doi:10.1016/j.chb.2019.106227.

[20] K. Dowd, J. Briselly, and O. Elizarova. Participatory Design in Practice, 12 2017. URL: https://uxmag.com/articles/participatory-design-in-practice.

[21] R. Elliott and N. Jankel-Elliott. Using ethnography in strategic consumer research. *Qualitative Market Research: An International Journal*, 6(4):215–223, 12 2003. URL: https://www-emerald-com.proxy.lib.chalmers.se/insight/content/doi/10.1108/13522750310495300/full/pdf?title=using-ethnography-in-strategic-consumer-research, doi:10.1108/13522750310495300.

[22] F. Fagerholm and J. Münch. Developer Experience: Concept and Defi-

nition. page 73–77, 2012. URL: https://arxiv.org/pdf/1312.1452.pdf, doi:10.1109/ICSSP.2012.6225984!

[23] J. Fereday and E. Muir-Cochrane. Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods*, 5(1):80–92, 3 2006. URL: http://journals.sagepub.com/doi/10.1177/160940690600500107, doi:10.1177/160940690600500107.

[24] Forrester. *The Total Economic ImpactTM Of Stack Overflow For Teams: Cost Savings And Business Benefits Enabled By Stack Overflow For Teams*. 11 2019. URL: https://cdn.sstatic.net/Shared/Product/Teams/TEI_of_StackOverflow_Enterprise.pdf.

[25] R. Friis Dam and Y. Siang Teo. Define and Frame Your Design Challenge by Creating Your Point Of View and Ask "How Might We", 2017. URL: https://www.interaction-design.org/literature/article/define-and-frame-your-design-challenge-by-creating-your-point-of-view-and-ask-how-might-we.

[26] R. Friis Dam and Y. Siang Teo. Introduction to the Essential Ideation Techniques which are the Heart of Design Thinking, 2019. URL: https://www.interaction-design.org/literature/article/introduction-to-the-essential-ideation-techniques-which-are-the-heart-of-design-thinking.

[27] R. Friis Dam and Y. Siang Teo. Stage 2 in the Design Thinking Process: Define the Problem and Interpret the Results, 2019. URL: https://www.interaction-design.org/literature/article/stage-2-in-the-design-thinking-process-define-the-problem-and-interpret-the-results.

[28] J. J. Garrett. *The elements of user experience : user-centered design for the Web and beyond*. New Riders, 2011.

[29] C. Geertz. *The Interpretation of Cultures*. Fontana Press, 1973.

[30] S. Gibbons. Design Critiques: Encourage a Positive Culture to Improve Products, 10 2016. URL: https://www.nngroup.com/articles/design-critiques/.

[31] S. Gibbons. User Need Statements: The 'Define' Stage in Design Thinking, 3 2019. URL: https://www.nngroup.com/articles/user-need-statements/.

[32] R. L. Gold. Roles in Sociological Field Observations. *Social Forces*, 36(3):217–223, 3 1958. doi:10.2307/2573808.

[33] Google. Crazy 8's. URL: https://designsprintkit.withgoogle.com/

methodology/phase3-sketch/crazy-8s.

[34] Google. Material Design. URL: https://material.io/.

[35] B. Hameed. Announcing TechDocs (beta) — the way to create and consume technical documentation at Spotify, 5 2019.

[36] M. Hammersley and P. Atkinson. *Ethnography : principles in practice*. Routledge, 2010.

[37] B. Hanington and B. Martin. *Universal Methods of Design Expanded and Revised: 125 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport Publishers, 2019.

[38] R. Hartson and P. S. Pyla. *The UX Book: Process and guidelines for ensuring a quality user experience*. Elsevier, 2012.

[39] Z. Holman. Introducing GitHub Enterprise, 11 2011. URL: https://github.blog/2011-11-01-introducing-github-enterprise/.

[40] R. Holmes, R. J. Walker, and G. C. Murphy. Strathcona Example Recommendation Tool. Technical report, 2005. URL: http://jakarta.apache.org/commons/httpclient/.

[41] K. Holtzblatt. Contextual design: Experience in real life. In *Mensch & Computer 2001*, pages 19–22. Springer, 2001.

[42] B. Houck, C. Bird, and T. Zimmermann. Explorations into developer productivity at Microsoft, 12 2019.

[43] IDEO. How Might We. URL: https://www.designkit.org/methods/3.

[44] Interaction Design Foundation. What is User Experience (UX) Design? URL: https://www.interaction-design.org/literature/topics/ux-design.

[45] P. W. Jordan. *An introduction to usability*. Taylor & Francis, 2002.

[46] H. Kniberg. Spotify Engineering Culture - Part 1, 7 2019. URL: https://www.youtube.com/watch?v=Yvfz4HGtoPc.

[47] C. Kopp. Product Differentiation, 9 2019. URL: https://www.investopedia.com/terms/p/product_differentiation.asp.

[48] A. Lam. Tacit knowledge, organizational learning and societal institutions: An integrated framework. *Organization Studies*, 21(3):487–513, 2000. doi:10.1177/0170840600213001.

[49] H. Li, Z. Xing, X. Peng, and W. Zhao. What help do developers seek, when and how? *Proceedings - Working Conference on Reverse Engineering, WCRE*,

pages 142–151, 2013. doi:10.1109/WCRE.2013.6671289.

[50] Lookback.io. All Lookback's features. URL: https://lookback.io/features/.

[51] S. Mac Naught. 635 Brainwriting Method for Ideas Generation in Content Marketing, 3 2014. URL: https://www.staceymacnaught.co.uk/635-brainwriting-method/.

[52] A. Meyer, G. Murphy, T. Zimmermann, and T. Fritz. Enabling Good Work Habits in Software Developers through Reflective Goal-Setting. *IEEE Transactions on Software Engineering*, PP:1, 2019. doi:10.1109/TSE.2019.2938525.

[53] A. Mockus and J. D. Herbsleb. Expertise browser. In *Proceedings of the 24th international conference on Software engineering - ICSE '02*, page 503, New York, New York, USA, 2002. ACM Press. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1007994http://portal.acm.org/citation.cfm?doid=581339.581401, doi:10.1145/581339.581401.

[54] D. Mortensen. How to Do a Thematic Analysis of User Interviews, 4 2020. URL: https://www.interaction-design.org/literature/article/how-to-do-a-thematic-analysis-of-user-interviews.

[55] Mural. What is MURAL?, 2020. URL: https://support.mural.co/en/articles/2113691-what-is-mural.

[56] G. Niemen. Golden Path and Golden Path Tutorials, 2018.

[57] D. A. Norman. Human-centered design considered harmful. *interactions*, 12(4):14, 7 2005. URL: https://dl-acm-org.proxy.lib.chalmers.se/doi/pdf/10.1145/1070960.1070976?download=true, doi:10.1145/1070960.1070976.

[58] D. A. Norman. *The design of everyday things*. Basic Books, 2013.

[59] D. A. Norman. Some observations on mental models. In *Mental models*, pages 15–22. Psychology Press, 2014.

[60] A. Osborn. *Applied Imagination-Principles and Procedures of Creative Writing*. Read Books Ltd, 2012.

[61] G. Peoples. Why Spotify Is Still Sprinting for Maximum Market Share, 4 2020. URL: https://www.billboard.com/articles/business/streaming/9325846/spotify-strategy-market-share-analysis.

[62] G. Petro. The Need For Speed: Time-To-Market Acceleration Is The Ultimate Retail Differentiator. *Forbes*, 10 2017. URL: https://www.forbes.com/sites/gregpetro/2017/10/06/the-need-for-speed-time-to-market-

acceleration-is-the-ultimate-retail-differentiator/.

[63] J. S. Poulin. Technical opinion: reuse: been there, done that. *Communications of the ACM*, 42(5):98–100, 1999.

[64] H. W. J. Rittel and M. M. Webber. Dilemmas in a general theory of planning. *Policy sciences*, 4(2):155–169, 1973.

[65] C. Sadowski, K. T. Stolee, and S. Elbaum. How developers search for code: A case study. In *2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings*, pages 191–201. Association for Computing Machinery, Inc, 8 2015. doi:10.1145/2786805.2786855.

[66] C. Sadowski and T. Zimmermann. *Rethinking Productivity in Software Engineering*. Apress, 2019. doi:10.1007/978-1-4842-4221-6.

[67] R. Sandler. How to use Airtable, the spreadsheet app taking Silicon Valley by storm, 3 2018. URL: https://www.businessinsider.com/airtable-review-guide-app-walkthrough-spreadsheets-tables-blocks-2018-3?r=US&IR=T.

[68] K. Sankar and S. A. Bouchard. *Enterprise Web 2.0 fundamentals*. Cisco Press, 4 2009. URL: https://archive.org/details/enterpriseweb20f0000sank.

[69] A. Scopatz and K. D. Huff. *Effective computation in physics : field guide to research with Python*. O'reilly, July, 2015.

[70] M. Sewak, D. R. Khilnani, A. Tronson, K. Okamoto, S. V. Kurhekar, L. Zhang, D. Sanchez, M. Ahmad, T. Assudani, B. Bahmani, and others. Finding a Growth Business Model at Stack Overflow. *Inc. Stanford Case Publisher*, pages 1–35, 2010.

[71] G. Shanmugasundaram, V. P. Venaktesan, and C. P. Devi. Modeling Measures for Service Interpretation in Discoverability of Service Oriented Architecture. *Procedia - Social and Behavioral Sciences*, 73:128–135, 2 2013. doi:10.1016/j.sbspro.2013.02.032.

[72] Slack. What is Slack? URL: https://slack.com/intl/en-se/help/articles/115004071768-What-is-Slack-.

[73] Spotify. Developer Productivity Top Level Metrics. *Internal Spotify document*, 2019.

[74] M. Steen. Tensions in human-centred design. *CoDesign*, 7(1):45–60, 2011.

[75] G. W. Stocking. *Observers observed: essays on ethnographic fieldwork*. University Of Wisconsin Press, 1986.

[76] The International Organization for Standardization [ISO]. ISO 9241-11:2018(en), 2018. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en.

[77] L. Torvalds. Tech Talk: Linus Torvalds on git, 5 2007. URL: https://www.youtube.com/watch?v=4XpnKHJAok8.

[78] A. Trendowicz and J. Münch. Chapter 6 Factors Influencing Software Development Productivity—State-of-the-Art and Industrial Experiences. *Advances in Computers*, page 185–241, 2009. doi:10.1016/s0065-2458(09)01206-6.

[79] R. Turner. How to perfect the facilitation tool, "sticky dot voting" - 4-H Leadership, Citizenship & Service, 11 2018. URL: https://www.canr.msu.edu/news/how_to_perfect_the_facilitation_tool_sticky_dot_voting.

[80] S. Wagner and F. Deissenboeck. Defining Productivity in Software Engineering. *Rethinking Productivity in Software Engineering*, page 29–38, 2019. URL: https://link.springer.com/chapter/10.1007/978-1-4842-4221-6_4, doi:10.1007/978-1-4842-4221-6{_}4.

[81] J. Wang, X. Peng, Z. Xing, and W. Zhao. *Improving Feature Location Practice with Multi-faceted Interactive Exploration*.

[82] A. Williams. User-centered design, activity-centered design, and goal-directed design: a review of three methods for designing web applications. In *Proceedings of the 27th ACM international conference on Design of communication*, pages 1–8, 2009.

[83] A. Williams. GitHub Pours Energies into Enterprise – Raises $100 Million From Power VC Andreessen Horowitz, 7 2012. URL: https://techcrunch.com/2012/07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/.

[84] C. Wilson. *Interview techniques for UX practitioners: A user-centered design method*. Newnes, 2013.

[85] Y. Ye and G. Fischer. Reuse-Conducive Development Environments. Technical report.

Appendix

A Interview Script Productivity

Introduction

- Thesis students from Chalmers - Researching tools and infrastructure for DX and productivity.
- We **don't** come from a coding background
- Productivity = not management or organisational aspects (meetings, squads, goals)
  Productivity = Efficiency in tooling and infrastructure to remove bottlenecks.
- We are **not researching you**

- Is it ok to record?
  Also, if there's ever a question you don't feel like answering, let us know and we'll just jump to the next one.

- Any other questions before we get started?

- We'll spend 5 min getting to know you, then about 20 min talking about your work flow and 20 min talking about productivity before coming in for a close.

Warm-up

1. How long have you been working at Spotify?
    a. What did you do before Spotify?

2. What's your role here at Spotify?
    a. What squad are you in?

3. What is the role of your squad?
    a. What does your squad constellation look like? (BE/DE, UX etc)
    b. How many services/pipelines/client-features do your squad have in production?
    c. How do you distribute work on services/features in your squad?

Workflow

4. Could you **categorize** the type of development work you do?
    a. Which are the 2 main ones [if it helps - think of last week]?

5. [For each of the two categories…] Can you give me a high-level overview of the steps you take for it - from a SDLC perspective?
    a. [Probe deeper on these if needed]
        i. What's your role on the project?
        ii. How do you decide what technologies to use?
        iii. How do you make use of potential pre-existing components?

iv. How long does that type of project typically take?
v. Are there any typical time sinks?
vi. [What does the output look like?]
vii. How do you collaborate with other squads? When?
viii. What tools do you use?
ix. How/where do you look for support when you get stuck?

b. What was the easiest part of working on that project?
c. What kind of issues do you typically run into? DIG HERE

Development - e2e

6. Can you tell me about an incident that happened lately when you got really frustrated with a tool?
    a. Does this happen regularly?
    b. How could you see this being fixed?

7. On a scale from 1-10, how streamlined would you say that Spotify's development processes are?
    a. Elaborate on that please. DIG
    b. What would it take to raise that number / How can Spotify support in doing this?

8. Does your squad have any "hacks" or "unique tools" that you have developed on your own for your own gain or productivity?

Closing

9. Have you used, seen or heard about any tools or infrastructure that could be of great use here at Spotify?

10. [How could Spotify better support you in being productive?]

11. In regards to productivity bottlenecks or development friction, is there something you have thought of that has not been brought up during this interview?

Thank you so much for taking your time answering these questions. It means everything for our research. If you don't have anything more to add, we could finish the interview here.
If something comes to your mind after this interview, feel free to reach out to us.

B Interview Script Discoverability

Introduction

- Thesis students from Chalmers - Researching tools and infrastructure for DX when it comes to discoverability.
- We **don't** come from a coding background
- Productivity = not management or organisational aspects (meetings, squads, goals)
  Productivity = Efficiency in tooling and infrastructure to remove bottlenecks.
- We are **not researching you.**

- Is it ok to record?
  Also, if there's ever a question you don't feel like answering, let us know and we'll just jump to the next one.

- Any other questions before we get started?

- We'll spend 5 min getting to know you, then about 20 min talking about your work flow and 20 min talking about productivity before coming in for a close.

Warm-up

1. **How long have you been working at Spotify?**

2. **What's your role here at Spotify?**
   a. **What squad are you in?**

Workflow

3. **What project are you currently working on?**
   a. What is your role in this project?
   b. When you are coding, what code are you normally writing? New stuff/maintenance?
   c. Does your work involve any maintenance?

4. What things in your daily developer work would you rather **not** do?
   a. Are there any typical time sinks?

Overview

If you were to describe
How do you keep an overview on what your squad owns?

- Do you visualize it in any way?
-

How do you make sure other squads could utilize what you have already built?

- Do you highlight your solutions in any way?
- What is your reaction/response to squads that want to use your code?

How does your squad make sure you don't build something that has already been built before?

When you are looking to find something another squad or employee has built, how do you go about doing that?

- What is your procedure like?
- What information are you looking for?
    - Owner, documentation, code?
    - How do you know if the code/component is good to use?
- What works well?
- What are your major issues?
- How would you want it to work?

On a scale from 1-10, how well would you say that you understand the dependencies that exist within your squad?

- How does it tie together with the rest of Spotify?
- What are your dependencies?
- How could you see Spotify better support your understanding of the dependencies?

In what other ways could Spotify improve discoverability in the tech ecosystem?

Development - e2e

5. Can you tell me about an incident that happened lately when you got really frustrated with discoverability?
    a. Does this happen regularly?
    b. How could you see this being fixed?

6. On a scale from 1-10, how streamlined would you say that Spotify's development processes are?
    a. Elaborate on that please.
    b. What would it take to raise that number / How can Spotify support in doing this?

Closing

7. In regards to productivity bottlenecks or development friction, is there something you have thought of that has not been brought up during this interview?

Thank you so much for taking your time answering these questions. It means everything for our research. If you don't have anything more to add, we could finish the interview here.
If something comes to your mind after this interview, feel free to reach out to us.

C Code book

Code book

#	Interpretive codes	Number of citations within this code
1	Fragmented documentation sources	16
2	Experienced colleagues help with understanding	15
3	Experienced colleagues help with discoverability	14
4	Utilising previously answered questions	13
5	Need for up-to-date ownership	13
6	Understanding other's implementations by reading example code	12
7	Asking is preferred over searching	10
8	Non descriptive names hinder discoverability	9
9	Difficulty evaluating code options	8
10	re-organisations mess up ownership	7
11	Names are non descriptive	7
12	Siloed information in employees can become buss factor	7
13	Velocity of answers is important	7
14	A wish for one entry point for everything	7
15	Historical information is hard to find	7
16	Existing components/services can be hard to find	7
17	Difficulty evaluating documentation options	7
18	Dedicated Slack channels for support	6
19	Slack search doesn't give wanted search results	6
20	Missing owners for some stuff	6
21	Easiness of creating documentation is important	6
22	Squads have Tribal knowledge	5
23	Documentation perceived as not reliable	5
24	Desire to create knowledge database	5
25	Cloud search is used for general information finding	5
26	Some things miss documentation	5
27	Being bounced to find the right place	5
28	Prefer to find solutions themselves	5
29	Using Backstage to find ownerships	5

#	Interpretive codes	Number of citations within this code
30	No way to see best practices	5
31	Fear of being perceived as Incompetent	4
32	Slack is used if you know who to reach	4
33	Documentation is generally NOT up-to-date	4
34	Slack as the primary entry point	4
35	Proprietary systems limits external searches	4
36	Changes makes updated documentation hard to maintain	4
37	Hard to word searches	4
38	No natural place to share hacks	4
39	Up-to-date documentation is useful	3
40	Aligment over time-zones is tricky	3
41	Experienced colleagues arn't always up-to-date	3
42	Want to see use cases	3
43	Access is sometimes a blocker	3
44	Missing reasoning behind recommendations	3
45	Consider audience for documentation	3
46	Forced reviews of documentation	3
47	Ownership gets transfered forcefully	3
48	Non descriptive names of documents are problematic	3
49	Backstage search generates too many results	3
50	Same documentation exist in multiple places	3
51	Need for clearer migration processes	3
52	Going outside normal is tricky	3
53	Data discoverability can be tricky	3
54	deprecated code is not removed	2
55	Good documentation is faster than reading code	2
56	Documentation as summarization	2
57	Communication between teams is difficult	2
58	Want to be informed before asking	2
59	Need smarter search than just keywords	2
60	Mixed messages due to un-updated information	2

#	Interpretive codes	Number of citations within this code
61	Documentation search doesn't give wanted results	2
62	Non up-to-date documentation is not removed	2
63	Terminology can be difficult as a new joiner	2
64	Data is understood by reviewing upstream dependencies	2
65	Educating on how to do documentation	2
66	Using coding history for discovering contributors	2
67	Hard to remember where something was written	2
68	Slack as solution bank not scalable	2
69	Certification for documentation	2
70	Different places for different kind of documentation	2
71	YAML files indicating ownerships and purpose	2
72	Lack of trust in certain search tools	2
73	Hard to get a holistic picture of who owns what	2
74	Difficult finding right documentation for your purpose	2
75	Search is only used to find previously seen information	2
76	No way to find similar code examples	2
77	Existence of components get siloed into tribes/squads	2
78	Understanding the roadmap is important for everyone in the squad	2
79	Double-checking as a way to Quality assure	1
80	Difficult finding squad via function	1
81	Most outdated documentation minorly off	1
82	Duplicated work due to lack of communication	1
83	Date of change aspect important for code evaluation	1
84	Reading code = Trust	1
85	Consensus is nice but slow	1
86	Praise is an incentive	1
87	Need for collaborative discussion space	1
88	Remove need for documentation	1
89	Learnability dependent navigation in TechDocs	1
90	Search might show outdated results	1

#	Interpretive codes	Number of citations within this code
91	"How to's" of internal tooling is appreciated	1
92	Testing documentation on new joiners	1
93	Tools that feel simple are used for multiple things	1
94	Reference documentation for specific users	1
95	Automation is valued by developers	1
96	No behavior of saving things	1
97	Another tool = more complexity	1
98	Documentation fragmentation due to docs as code	1
99	No good way to see who is calling a service	1
100	Squads want to get rid of specific ownership	1
101	Hard to create fitting names of squads	1
102	Waiting for answers to questions can be a blocker	1
103	Hard to remember previously successful searches	1
104	Documentation sometimes exist in hidden places	1
105	Platform teams should have descriptive names	1
106	A network of people alleviates discoverability	1
107	Interest in using Spotify code search more	1
108	Some squads get unofficial ownership	1
109	Code is easier to write than documentation	1
110	Conveying your problem can be difficult	1
111	Maintaining consistency is important	1
112	Allow for doing rather than complaining	1
113	Internal user needs are obtained by PM/TOs asking around	1
114	Example code is found by identifying similar functionality	1
115	Small services risk to not be well-maintained	1
116	Missing timeline of decisions	1
117	Need to make sure change of ownership can be done smoothly	1
118	Developers create their own "How to's"	1
119	Waiting for reviews can be a blocker	1
120	Require walkthrough of every Golden Path	1
121	Unfortunate expectation to miss out on useful code	1

#	Interpretive codes	Number of citations within this code
122	Talking to colleagues provide added context	1
123	Squads might have no interest in scaling their service	1
124	Mentor program to support newcomers	1
125	Trusting specific developers to create good code	1
126	Fragmented information sources	1
127	Changing department demand new knowledge sources	1
128	Hard to find previous code solutions	1
129	Missing "How to's" on regular use cases	1
130	Difficult to evaluate code quality	1
131	Business logic for decisions would be appreciated	1
132	Previous project colleagues help with discoverability	1
133	Some documentation lack context	1
	SUM	417

D Prototype feedback script

Mid-fi prototype feedback

Introduction

Recording?

We are two Master's thesis students working with one of the Platform Infra squads called Tools who owns Backstage. For the past few months we have researched discoverability of reusable code within the Spotify ecosystem.

During this session, we will present a concept that we've worked briefly on for the past week. The idea is to extend the Explore section of Backstage with a new tab we call **Use Cases**.

- First of all, what comes to your mind? What do you expect to find under this new section?

The Idea

Our vision is for it to become a marketplace where squads and individual contributors can share and highlight their reusable components (in the form of libraries, APIs, datasets, SDKs etc.) but in a **context**. Instead of just listing the entities as a list of items, they are explored through different use cases. Each use case makes up a card and is described briefly for the user to get a glance of what the case is about. One use case can make use of, and highlight, several libraries in combination, similarly to how these entities are used in the code repos.

- Now that we have given you the brief, what are your initial thoughts?

Let's dig into the prototype! The prototype is pretty low fidelity - we haven't focused on colors, button styles, fonts or anything like that.
The kind of feedback we're interested in is mainly on the idea of "discovering reusables via use-cases" as such, as well as what content that would be displayed.

Feedback on categories

- What do you think about the current categories? Do they make sense to limit the search?
- Let's go through them - what do you expect to see under each category?
- What other categories would you like to see to assist in the search?

Feedback on evaluation criteria; date / views / likes

When evaluating a use-case to understand if it potentially would be useful, we want to give some quick information in the cards.

- What's important?
- What else is important to see in a card?

Before opening up a card to get to the use case view

- What do you expect to see in the use case view?

Then show it, ask what they think.

Ask about the ability to edit / sandbox in the use case view.

E Usability Test script

Usability test - Script

Start
Thanks for joining us today.
We're 2 master thesis students, working on how to enhance discoverability of reusable code at Spotify.

We've created a concept of how this could be done and we'd like you to try out a medium fidelity prototype to test some features and come with feedback.

This will be recorded so that we can look at it afterwards.
First of all, it's important to emphasise that we're testing the system - we're not testing you in any way.

Also, we'll follow you on the screen but we'd like you to think out loud which means that you should tell us what your thoughts are when moving through the system.
For example "I see that this thing is underlined and I understand that as a link so I will click on that".

Any questions before we start?

Scenario:
You are a backend developer at Spotify, currently trying to build a feature in your Python backend that can track different user events.

You don't know which library or other technologies that already exist that you can use for your project. You therefore go to Backstage to see what you can find.

We want you to try the new **"Use cases"** section, found under the **Explore** tab. Please try to navigate to this place.

Backstage → Explore → Use Cases

From what you can see:
- *"What do you think this page is?"*

- *"What do you think use cases are?"*

Over the cards, you can see a title called "Recommended":
- *"What do you think Recommended means in this case?"*

Search

We'd want you to try to find a use case:

- *"How would you go about doing that?"*

We want you to start looking for a use case by looking for the keywords **Backend, python, monitoring** in that order.

- *"What happened now"*
- *"How do you perceive the different suggestions?"*
- *Följdfrågor*

You realize that **Monitoring** didn't give you the use cases you were hoping for. Instead you want to remove that tag and replace it with the keyword **Tracking.**

Here you have your search results - Now, remember that you're trying to build a feature in your Python backend that can track different user events.

Gabo

- *"How do you perceive this page"*
    - *Follow up question*
- *"What does "Technologies used" mean?"*
- *"What does "Related tech" mean?"*
- "If you have a question regarding this Use Case - where would you go for that?"
- "If you were to evaluate this use case - how would you do that" → Status
    - "Given those evaluation criterias - what's your evaluation of this use case?"

Go to Gabito

Now you've ended up where you're supposed to be.

- Given the different sections, "do you feel like you have what you need to get started with this technology?"
- Are you missing something?
- "If this is what you needed - how would you continue after this?"

DONE!

What's your overall impression of this whole Use cases feature?
What do you like the most about this feature?
What do you think could be improved?

- How would you change it?

How would you compare exploring through use cases to the current way of searching for reusable code?

F Questionnaire

Feedback: Use cases concept

Thank you for participating in our usability test and trying out the concept of use cases.

We would love to get some final feedback so we can keep improving the concept.
* Required

1. 1. How well do you think this system assists you in finding new technologies? *

   Mark only one oval.

	1	2	3	4	5	6	
Not at all	◯	◯	◯	◯	◯	◯	Very much

2. Why? *

3. 2. How well do you think the system lets you evaluate the quality of the use cases? *

   Mark only one oval.

	1	2	3	4	5	6	
Not at all	◯	◯	◯	◯	◯	◯	Very much

4. Why? *

5. 3. How well do you think the system helped you find the optimal solution to your problem?

   Mark only one oval.

	1	2	3	4	5	6	
Not at all	◯	◯	◯	◯	◯	◯	Very much

6. Why? *

7. 4. How much did you feel in control of the results shown as an effect of your search? *
   1 = Very dissatisfied 5 = Very satisfied

   Mark only one oval.

	1	2	3	4	5	6	
Not at all	◯	◯	◯	◯	◯	◯	Very much

8. Why? *

9. 5. How well did the system explain the purpose of a use case? *

   Mark only one oval.

	1	2	3	4	5	6	
Not at all	◯	◯	◯	◯	◯	◯	Very much

10. Any other general thoughts/comments/feedback?

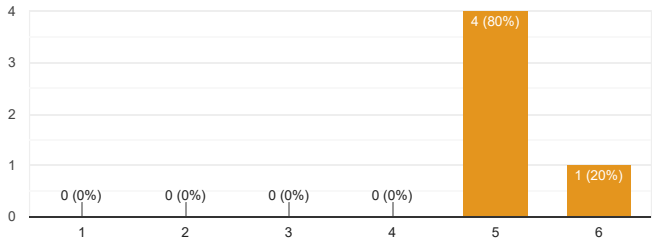
G Questionnaire answers

Feedback: Use cases concept

5 responses

1. How well do you think this system assists you in finding new technologies?

5 responses



Why?

5 responses

I think there are a few strange UX patterns, but it is still 1000% better than what exists currently

I love the ability to browse through the use cases, because it can allow me to favorite whatever use case i feel i can use in the future.

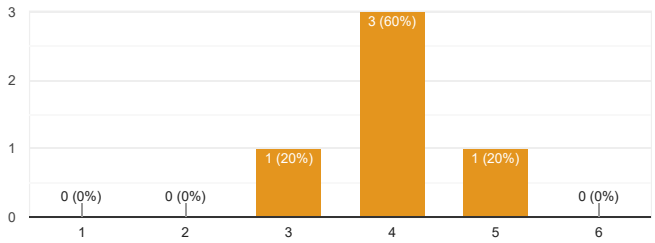
"technology" is very overloaded term. I'd assume "library". If t works as advertise I think it would be really helpful.

I really like the idea of tags.

A little bit diffucult to say without having used it more. It seems like the search experience is good. The result that you end up on is good. Those are the important parts. How do I find the documentation and once I am there,

2. How well do you think the system lets you evaluate the quality of the use cases?

5 responses



Why?

5 responses

I would want more information on testing, how many other teams are using it, lifecycle, and support

pretty good once you click into it to see the details, though i personally would prefer it wasnt only based on the comments section.

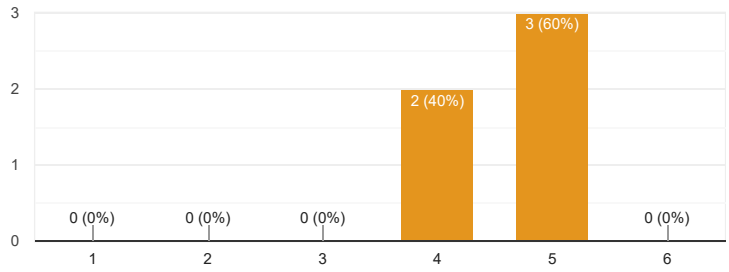
I can only infer the quality of use case by order of search result.

Maybe I would look at the comments. But usually, I don't think they would have comments.

Quality = How valuable and useful it is. It is pretty good. But it is a difficult problem. I feel like it is not perfect but i am not sure what ciuld be done either. It is better than what we have. There should be a "Golden Path" badge in Backstage.

3. How well do you think the system helped you find the optimal solution to your problem?

5 responses



Why?

5 responses

I think it was very easy to search and find what I needed

pretty good because it had good descriptions and title names. it would be a perfect score if there were sample code once you click one.

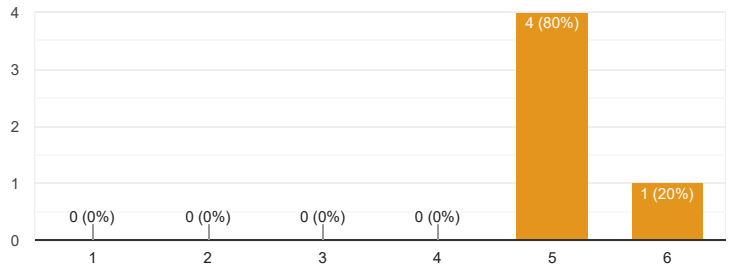
I an ideal scenario this worked for me. I am little concern about real world use case.

I had to go to the old use case to find the new one. Maybe you could order it by last updated but the problem was that the tags were wrong.

Difficult to answer. That is related to the last question. I am faily optimistic that it gave me the perfect sulution - BUT I am not sure. I am lacking a promise. "We're promising that this is the optimal solution". But, seeing the owner "tools" and they sound important. I like that.

4. How much did you feel in control of the results shown as an effect of your search?

5 responses



Why?

5 responses

The search functionality looked great

loved being able to sort by date or relevancy but i wish it would have a "most popular" sorting as well.

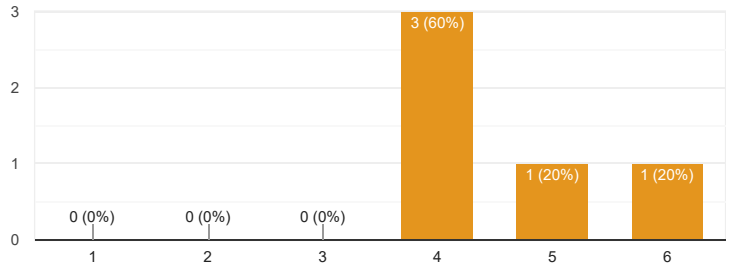
This seems very straight forward search.

The tagging system is really good I think.

I think I've used it to little to give it a 6. It seemed like I was in control.

5. How well did the system explain the purpose of a use case?

5 responses



Any other general thoughts/comments/feedback?

4 responses

looks great so far!

The name "use-case" might be little confusing. There might be initial barrier to people actually starting to use this.

Maybe it could be more information. An FAQ and or stack overflow to make sure if I can use it or not.

Personally I like overexplaining. It does a good job and I am optimistic.

H Project Plan Gantt-chart