Exploring Business Value and User Experience of Open Source Health

Filip Bolling and Jonna Gustafson

DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2021

MASTER THESIS

debricked
cybersecurity

Exploring Business Value and User Experience of Open Source Health

Filip Bolling and Jonna Gustafson

LUND
UNIVERSITY

Exploring Business Value and User Experience of Open Source Health

Published by
Department of Design Sciences
Faculty of Engineering LTH, Lund University
P.O. Box 118, SE-221 00 Lund, Sweden

Subject: Interaction Design (MAMM01)
Division: Division of Ergonomics and Aerosol Technology
Supervisor: Susanne Frennert
Co-supervisor: Emil Wåreus at Debricked
Examiner: Mattias Wallergård

Abstract

Open source software is becoming more common in software development today and constitutes an increasing share in modern web applications. Open source has many benefits; it is cost-effective and increases development speed, quality, and security. With the advent of open source, companies and organizations have taken many different initiatives to minimize the risks of open source, disseminate knowledge, and facilitate the work of open source.

This thesis was carried out in collaboration with one of the initiating companies, Debricked AB, active in cyber security. The project's purpose was to explore the selection process of third-party components to develop a design proposal of a business valuable tool with high usability and good user experience, to help organizations and end-users improve their selection process of open source software. The target group for this tool is experts in software development and software security.

To achieve the project's purpose, a human-centered design process has been used. The users' current processes have been reviewed through user interviews to understand the users and their needs. The design proposal has been developed in the form of prototypes in several iterations. A sample from the user group has tested each prototype in each iteration.

According to the results, the proposed design fulfills the aim of the thesis with a business valuable tool with high usability that corresponds to the third-party component selection process with desired evaluation factors.

Keywords: Debricked AB, Open Source Software, Open Source Health, Open Source Evaluation, Usability, Human-centered design

Sammanfattning

Öppen källkod, open source software på Engelska, blir allt vanligare inom mjukvaruutveckling idag och utgör en allt större andel i moderna webb applikationer. Den öppna källkoden har många fördelar; det är kostnadseffektivt och ökar utvecklingshastigheten, kvaliteten och säkerheten. I och med den öppna källkodens framfart, har många olika initiativ tagits av företag och organisationer för att minimera de risker som finns med öppen källkod, för att sprida kunskap och för att underlätta i arbetet med öppen källod.

Det här examensarbetet utfördes som ett sammarbete med ett att av de initiativtagande företagen, Debricked AB, som är verksamma inom cybersäkerhet. Projektets syfte var att utforska urvalsprocessen för tredjepartskomponenter med målet att utveckla ett designförslag av ett värdeskapande verktyg med hög användbarhet och bra UX som ska att hjälpa organisationer och slutanvändare att förbättra sin urvalsprocess av programvara med öppen källkod. Målgruppen för verktyger är experter inom mjukvaruutveckling och mjukvarusäkerhet.

För att uppnå projektets syfte har en människocentrerad designprocess använts. Användarnas nuvarande processer har granskats genom användarintervjuer, för att förstå användarna och deras behov. Designförslaget har tagits fram i form av prototyper i flera iterationer. Varje prototyp i varje iteration har testats av ett urval ur användargruppen.

Enligt resultatet uppfyller den föreslagna designen uppsatsens syfte med ett värdeskapande verktyg med hög användbarhet som motsvarar urvalsprocessen av tredjepartskomponenter med önskade utvärderingsfaktorer.

Nyckelord: Debricked AB, Öppen Källkod, Evaluering av Öppen Källkod, Val av Öppen Källkod, Användbarhet Människocentrerad design

Acknowledgements

This report is a result of a Master Thesis in Interaction Design. The thesis has been conducted at the division of Ergonomics and Aerosol Technology at Lund University and in cooperation with Debricked AB.

We would like to express our gratitude to our supervisors Susanne Frennert from Lund University and Emil Wåreus at Debricked AB for their valuable help, inspiration and guidance during this thesis. We would also like to thank Ida Lång, UX-designer, the Data Science team for their contribution with support and expertise and for welcoming us to the company.

Finally, we would like to thank all test persons taking the time for interviews and providing us with a lot of valuable feedback.

Lund, September 2021

Filip Bolling, Jonna Gustafson

Contents

List of Acronyms and Abbreviations

API	application programming interface
FLOSS	free libre open source software
FOSS	free and open source software
GUI	graphical user interface
HCD	human centered design
ML	machine learning
OSH	open source health
OSOR	open source observatory
OSPO	open source program office
OSS	open source software
RQ	research question
UI	user interface
UX	user experience

1 Introduction

This chapter will introduce the purpose, research questions, and limitations of the project.

1.1 Problem Statement and Purpose

Within software development today, third-party open source components constitute a crucial part of the code base. These open source components contribute to higher quality, security, development efficiency, and better functionality. The amount of open source software (OSS) on the market has exploded, leading companies in this area to face several critical challenges within licensing, measuring quality, and finding all data needed to evaluate these components. These challenges occur both when choosing new OSS and when monitoring already imported OSS. [1]

To improve the efficiency of companies working with OSS, Debricked has developed a tool, Open Source Health (OSH) to perform this kind of quantitative analysis on different OSS components. This tool outputs numerical values indicating the activity and sustainability of the community surrounding an open source package. The project's purpose is to explore the selection process of open source components and to develop a design proposal for a business valuable tool with high usability and good user experience to help organizations and end-users improve their selection process of open source software. The target group for this tool is experts in software development and software security. It's intended to be usable for different roles and use cases in software development, such as software developers, software architects, tech leads or CTOs.

1.2 Research Questions and Goals

The main focus for this empirical investigation is the intake process of OSS from a company's point of view, with the goal to develop an OSS evaluation platform with high usability and good user experience that provides value for individuals in different roles and organisations. To reach the purpose of this project, the research questions (RQ) is formulated as:

> RQ1: What is the selection process for different use cases when adopting new open source software?
>
> RQ2: Which metrics are important for different use cases when evaluating open source software?
>
> RQ3: How can a user interface be designed to facilitate the selection process and to communicate a business value of Open Source Health?

1.3 Limitations

Since this project mainly aims to find the relevant factors and design concepts, the full front-end development was handed over to the Debricked development team. Also, the user experience (UX) design focus on exploring where users find value in the tool, not a full working user interface (UI). Due to the limitations in time, 20 weeks, no time was spent on responsive design [2] to adapt the tool to more than one platform, nor universal design [3] to make the tool accessible to all users. During our work with the thesis, Debricked has developed and implemented their design in parallel to be able to go live in early September 2021.

2 Background

In this chapter, all background information to this project is explained.

2.1 Open Source Software

In the contemporary world of software development, OSS has become ubiquitous. It has been estimated that OSS constitutes 80-90% of any given piece of modern software. It's common with a heavy reliance on OSS in nearly all industries, in both public and private sectors as well as in both tech and non-tech organizations. [4]

OSS refers to software with source code that anyone can inspect, modify and have the possibility to contribute to [5]. Some examples of well-known pieces of open source software are the operating systems Linux and Android. Open source software is also essential for our modern digital infrastructure. One example of this is the fact that two pieces of open source web server software, Apache HTTP server, and Nginx, are responsible for hosting over 65% of all websites on the internet, as of July 2021 [6].

2.1.1 Terminology

There are several core concepts that are unique for open source software. These definitions below explain some of them. Most of the explanations are taken from the Foundation for Public Code [7], GitHub's glossary [8] and the book *Roads and bridges: The unseen labor behind our digital infrastructure* by Nadia Eghbal [9].

Programming language: Programming languages are the communication backbone of software. They help different software components perform actions and talk to one another. Popular examples of languages include JavaScript, Python and C. [9]

Open source software: Open source software: Open source software (OSS) has the same technical definition as *free software*; software that is free to run for any purpose (commercial or non-commercial) as well as be studied, changed and distributed. Culturally, the term "open source" is a more corporate friendly alternative to the term "free software" and highlights the practical benefits of public software, whereas free software is a social movement. There are several different variants and acronyms, in addition to OSS. Free and Open Source Software (FOSS) is meant to be inclusive of both terms that refer to public software. Free, Libre, and Open Source Software (FLOSS) is the most inclusive definition. [9]

Version Control: Version control is the management of changes to source code and the files associated with it. Changes are usually identified by a code, termed the revision number (or similar). Each revision is associated with the time it was made and the person making the change thus making it easier to retrace the evolution of the code. Revisions of code can be compared with each other and it makes it easy to restore previous versions. [7]

GitHub: A commercial platform for hosting code. GitHub launched in 2008 and is currently the most popular platform for people to host and collaborate on open source projects. (One can also host private code on GitHub.) GitHub helped standardize open source development practices and bring open source to a wider audience. Projects on GitHub use the version control system Git. [9]

Source code: For the purposes of this report, source code is the actual code associated with an open source project. [9]

Codebase: Any discrete package of code (both source and policy), the tests and the documentation required to implement a piece of policy or software. This can be, for example, a document or a version-control repository. [7]

Code repository (repo): The location of source code needed to use a software project. For example, GitHub offers a place to host one's repository so that other people can find and use it. Also colloquially called a "repo." The collection of source code files themselves are called the "codebase." [9]

Software library: Software libraries are "prefabricated" pieces of code that make it faster to write software, just as a construction company might buy prefabricated windows instead of building them from scratch. For example, instead of a developer writing their own user login system for an application, they can use a library called OAuth. [9]

Contributor (open source): Someone who has made a contribution to a live open source project. Examples of contributions include writing code,

documentation, or managing support issues. A contributor may not have commit access to a project (i.e. someone else must approve their contributions before they are live). [9]

Maintainer (open source): Someone who assumes the responsibility of an open source project. The definition varies from project to project. Sometimes maintainers are formally named in a project, and sometimes they emerge de facto based on who is doing the bulk of the work. A maintainer likely carries the burden of holistic project management more than any individual contributor. They may or may not have authored the original version of the project. They will likely have commit access to a project (i.e. can make changes directly to the project). [9]

Dependency: A piece of software that is needed for another program to work. A reliance on another component, in this context oftentimes on other open source packages.

Package: See software library. The terms library, lib, package and dependency are often used interchangeably, including in this thesis.

Package manager: A package manager is a tool that is used to install, upgrade and configure pieces of software. In the case of open source software, these managers are often programming language specific. Some commonly used package managers are PyPI for Python, npm for JavaScript, Maven for Java and Composer for PHP.

Documentation (software): Written information that explains how people can use or contribute back to a software project. Documentation is like an instruction manual for software. Without it, a developer wouldn't know how to use the project. [9]

License: Legally binding statement of the terms of use, in this context of open source software. There are many different types of open source licenses used. Generally, they can be ordered into two categories: permissive and copyleft. **Permissive licenses** allow for all use of the software, including commercializing derivative works. Common permissive licenses are MIT and Apache licenses. **Copyleft licenses** also allows for open use of the software, but it also requires that all derivative works are also open for use by others. In other words copyleft licenses may not be compatible with proprietary products. A common copyleft license is the GNU General Public License, GPL. [10]

Policy: A policy is a deliberate system of principles to guide decisions and achieve rational outcomes. A policy is a statement of intent, and is implemented as a procedure or protocol. Policies are generally adopted by a gover-

nance body within an organization. Policies can assist in both subjective and objective decision making. Public policy is the process by which governments translate their political vision into programs and actions to deliver outcomes. At the national level, policy and legislation (the law) are usually separate. The distinction is often more blurred in local government. In the Standard the word 'policy' refers to policy created and adopted by public organizations such as governments and municipalities. [7]

Fork: There are two types of forks. A **project fork** is the historical definition of a fork, in which someone makes a copy of an open source project and continues to develop it separately. It is used politically; for example, when there is internal disagreement about the project's direction, or to force substantial changes to the original project (ideally, merging the fork back into the main project). A **GitHub fork** refers to temporarily copying a project to make changes, usually with the intent of merging those changes back into the main project. It does not carry the political and social weight of a project fork. GitHub repurposed this term to encourage a culture of lightweight "tinkering" that is now prevalent among modern open source contributors. [9]

Star: A bookmark or display of appreciation for a repository. Stars are a manual way to rank the popularity of projects. [8]

Commit: A commit, or "revision", is an individual change to a file or set of files. A record is kept of the specific changes along with who made them and when. [8]

README: A text file containing information about the files in a repository that is typically the first file a visitor to a repository will see [8]. Often contains a description of the repository, guides for installation and usage and potentially links to further documentation.

Pull Request: Pull requests are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators. [8]

Issues: Issues are suggested improvements, tasks or questions related to the repository. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators.[8]

Merge: Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another. This often happens as a "pull request", which can be thought of as a request to merge. [8]

Vulnerability: A weakness in the code found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability. Mitigation of the vulnerabilities in this context typ-

ically involves coding changes, but could also include updating or removing dependencies. [11]

Framework: Software frameworks provide a structure for applications. They can be seen as a blueprint for an entire application, laying out how a program can look or how information gets stored for example. [9] One example is the frontend framework react.

2.1.2 Common Usage in Practice

OSS often takes the form of packages that are imported to implement additional functionality to a programming language. [9] This can be anything from a package containing code for a scroll bar in JavaScript to a library that can train deep neural networks in Python. These packages are language-specific and are often imported using package managers. Two commonly used package managers are PyPI[12] for Python and npm[13] for JavaScript.

To bring together many of the terms described above, the authors interpretation of a standard workflow utilizing OSS in practice is presented here: Developer A creates a project that implements some functionality, for example, a log-in page. They want to share this with the world, so they upload their *codebase* on *GitHub*, specify a *license*, for how others can use the code and makes it available for others on a *package manager*. Developer B later wants to create a log-in page. Instead of doing that from scratch, they find developer A's project. They look through the code and decide that it's good quality and that they trust developer A. They then use a *package manager* to install that library and use the code that A has already developed.

Arguably the biggest advantage of OSS comes after that process though. OSS projects are almost always developed collaboratively, and everyone can contribute. If developer B finds a bug in the login page, they can fix it themselves and make that fix available for everyone by suggesting code changes in the project. Or they can contribute to the project in other ways, for example by implementing new features or writing guidelines for how other people can contribute to the project. It's this crowd-sourcing of time and effort and collective development that makes today's digital ecosystems so diverse and thriving.

2.1.3 Pros and Cons of Open Source Software

A 2021 report from the Swedish Public Employment Service about strategic choices of e-archives, [14], identified several pros and cons of open source solutions in general.

Pros:

- Discovery of bugs and errors
- Contribute to creating a better product
- Collective development that spreads risks and increases the effectivity of contributors
- No user licenses, the organization can increase number of users themselves
- No vendor lock-in
- A transparent it-solution that doesn't keep any secrets for the organizations that chooses to use them
- No need for procurement, except for potential support and upkeep
- Consultants can be called-off via regular framework agreements
- More effective system documentation, since it is developed centrally and adopting organizations only document that which is unique for their usage

Cons:

- Risk of community "dying" and then being stuck with an it-solution that is no longer sustainable over time or able to be supported
- Risk of influence from foreign power if for example the community is focused or localized within a dictatorship

2.1.4 Sustainability in Open Source

As mentioned before there are several advantages with OSS, but there are also some risks, e.g. "risk of community dying". Evaluating the health of the project can minimize these risks. Johan Linåker, Postdoctoral Researcher in the Software Engineering Research Group from Lund University mainly looking into OSS within the public sector, stated at a webinar event together with

the Open Source Observatory (OSOR) community that ”health and sustainability of OSS are essential to a secure and robust digital infrastructure” [15]. Linåker divides sustainability within OSS into three parts [16];

- Productivity - activity within the community, for example the number of commits and contributors.
- Robustness - resistance to crises and defectors, improved by a good knowledge distribution within the community.
- Openness - the community is open to new ideas, changes and welcoming new members.

To evaluate community health in an OSS project, certain checklists and guidelines can be used. Both Red Hat, an American software company, and CHAOSS community, a Linux Foundation project aimed at defining and evaluating open source community health, provide complete checklists. The more checks you get on the checklist, the healthier the project is likely to be [17]. CHAOSS provides about 50 metrics and Red Hat about 80 metrics [18]. These metrics are divided into groups such as ”Governance”, ”Code Development Activity”, ”Communal Value”, ”Infrastructure”, ”Documentation”, ”Outreach” and more. These checklists are best suited for very large and active projects, many metrics are not plausible for small and medium-sized packages. The full checklists can be found at Red Hat's web page [1], and CHAOSS' web page [2] respectively.

Linåker described during an online call (2021-07-05) that the degree of maturity within OSS in Sweden is generally relatively low. There are considerable knowledge gaps within the evaluation metrics. Today a lot of the evaluation is about gut feeling. In order to increase the maturity and knowledge about OSS, this gut feeling has to be well-founded with both qualitative and quantitative metrics.

2.2 Debricked

This master thesis is a collaboration with Debricked AB. Debricked is a Malmö based startup company founded in 2018 that works within the cybersecurity field [19]. Their main product is a tool to scan open source projects within a company codebase to verify security, license types, and vulnerability level. Debricked is currently working on a project called Open Source Health (OSH), which this thesis is a part of.

[1] http://www.redhat.com/en/resources/open-source-project-health-checklist
[2] https://chaoss.community/metrics/

2.2.1 The Debricked Open Source Health Model

The Debricked Open Source Health model computes quantitative metrics on the activity, longevity, and developer quality of any given OSS project using machine learning (ML) techniques. The model takes a quantitative approach, using data from GitHub to evaluate the OSS projects. It is built as an hierarchical model of weighted summation, divided into three main layers, *features*, *practices* and *metrics* together with a normalization layer. The hierarchical model structure is shown in figure 2.1. Each layer has a sequential connection to the next in a one-to-many relation, i.e. a practice consists of one or more features, and a metric consists of one or more practices. Features use raw open source data to calculate values that bare signal to Practices. The output is a scalar value per feature/project, which gets normalized through projection to a normal distribution and scaled between 0 to 1000 in the normalization layer. Practices represent latent variables that are attractive to know as an open source consumer. Each input to the practice layer is scaled with weights and summarized to an output score. Metrics aggregate practices that process similar areas of interest into one abstract representation of one certain dimension of an open source project, such as popularity, contributors, and security. Figure 2.2, 2.3 and 2.4 shows the model structure for these three metrics.

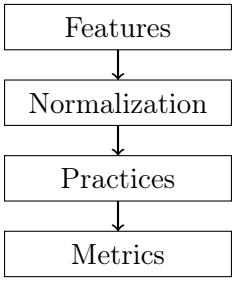


Figure 2.1: The hierarchical structure of the OSH-model. Features are based on raw data and are, through the different layers building up a metric.

Since the OSH project is under development, only two metrics, with corresponding risk areas, are implemented in the model, popularity and contributors. The security metric is still in the research phase.

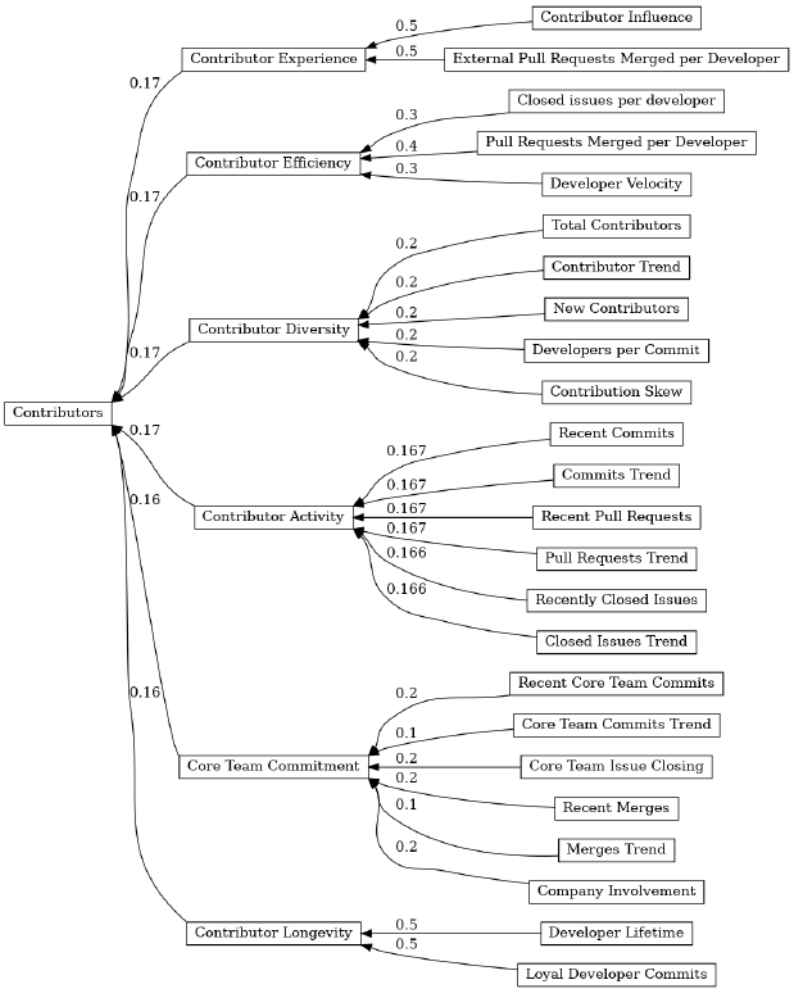


Figure 2.2: Contributors metric. Open Source projects consists of contributors. When deciding what open source to bring in to your software, it is important to inspect and analyse the contributors of a project. The features to the right sums up to the practices in the row in the middle. The practices are then summed up in the metric to the left.

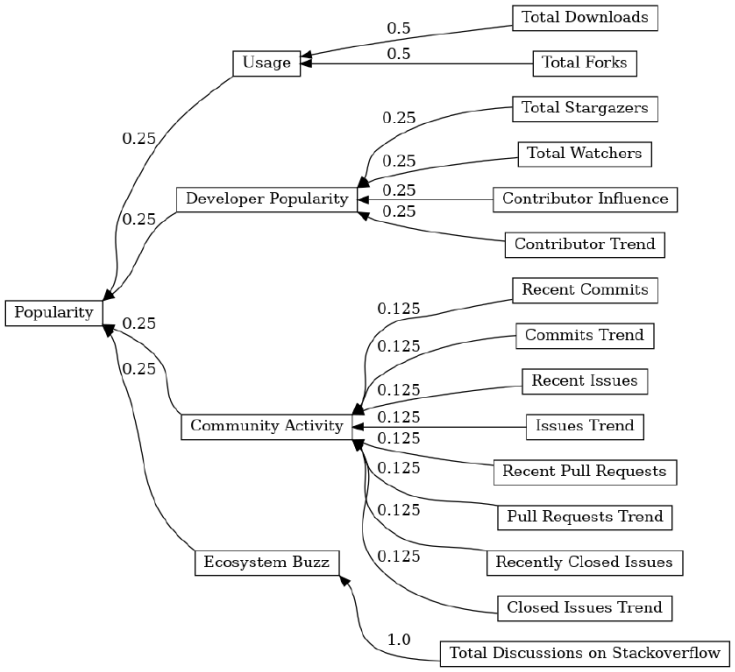


Figure 2.3: Popularity metric. The popularity of a repository is a rather crucial indicator of the health of a project. It signifies interest from both developers and users alike, pointing towards viability and continued development. The features to the right sums up to the practices in the row in the middle. The practices are then summed up in the metric to the left.

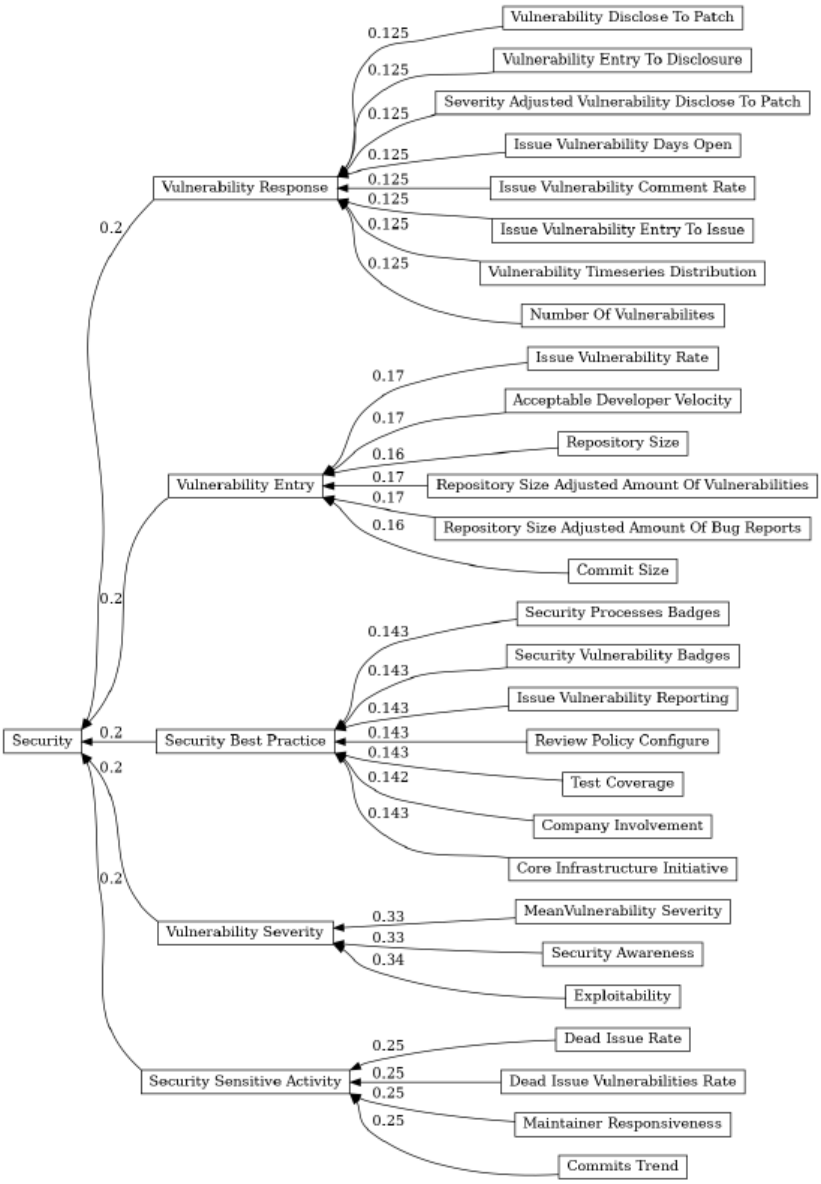


Figure 2.4: Security metric. The security of an open source project is important in many aspects of the software. This metric is impactful to the risk associated with an open source project, as it measures both indicators of vulnerability entry risk, and past vulnerability response performance. This metric also takes into account the risks associated with anomalies in development speed, as well as risks associated with halts in project activity. The features to the right sums up to the practices in the row in the middle. The practices are then summed up in the metric to the left.

3 Theoretical Background

In this chapter the main theories within interaction design and user experience which this thesis rely on are presented.

3.1 User Experience

The definition of User Experience (UX) is described by Don Norman and Jacob Nielsen as ""User experience" encompasses all aspects of the end-user's interaction with the company, its services, and its products." [20]. Good UX is to meet the exact needs of the user, giving the users even more than they know they want or need. Good UX is simplicity and elegance, the product should be easy and fun to use. Different disciplines, like engineering and design, should be seamlessly merged. It is important to keep UX and user interface (UI) separate. A good UI is not enough if the UX does not meet the user's needs. In the same way, it is important to distinguish UX and usability. Usability is a quality attribute of the UI, but UX is an even broader concept. [20]

3.1.1 Nielsen's 10 Usability Heuristics

The 10 usability heuristics are ten general principles or rules of thumb for interaction design originally formulated by Jakob Nielsen in 1994. These heuristics can be used as a framework for evaluating user interfaces to find usability problems [21]. The 10 heuristics are described in Appendix A.

3.2 The Design Process

The design process used in this thesis is based on The Double Diamond Process, Human Centered Design and Nielsen's 10 Usability Heuristics. They are

each described in detail below.

3.2.1 The Double Diamond Process

The double diamond process is a framework developed by the British Design Council [22]. This framework can be used to convey a design process and aims to improve convergent and divergent thinking. Divergent thinking meaning to explore an issue more widely or deeply, for example through idea creation (also known as ideation), and convergent thinking meaning to take focused action. The name, double diamond, illustrates this process of convergent and divergent thinking which can be seen in figure 3.1.

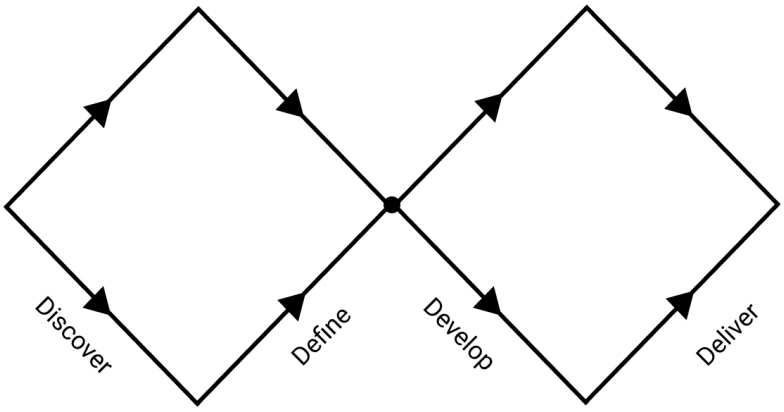


Figure 3.1: The double diamond design process.

The process contains four different steps; *Discover, Define, Develop* and *Deliver*. **Discover** is the phase where the designer should understand the problem and gathering insights about the users and the user needs. **Define** is the second phase in the research diamond and during this stage is the findings from the *Discover* phase are formulated to suit the project goal. **Develop** is the first step in the second diamond, the design diamond. In this phase is the design work started, different solutions are developed and tested within the company during several iterations. **Deliver**, in this last phase the final design is made to be ready to be launched. [22]

3.2.2 Human Centered Design

Human Centered Design (HCD) [23] is a design approach that puts human needs, capabilities and behaviour first, and the designs to accommodate those needs, capabilities and behaviours. One major principle of HCD is to avoid specifying the underlying problem for as long as possible, and instead make repeated approximations of the problem and iterate solutions for those approximations. This is done through rapid tests of ideas, and after each test evaluating the results and modifying the problem definition and approach.

HCD consists of four activities:

1. Observation
2. Idea generation (Ideation)
3. Prototyping
4. Testing

These four activities are repeated iteratively within the double diamond process. In **Observation** the nature of the underlying problem and need is researched. The interests, motives and true needs of the intended users are investigated. **Ideation** is the process of generating ideas for possible solutions. **Prototyping** is a method of quickly and cheaply implementing a basic version of a solution in order to get user feedback. This feedback is gathered in the **Testing** activity, where the prototypes are tested by people in as much of a natural setting as possible. This generates real experience in seeing how real users interact with- and interpret the solution. [23]

4 Method

This chapter describes the methods used in this thesis.

4.1 Discover

In the *Discover* phase, the context of the problem area is explored and insights about users and user needs are gathered. In terms of HCD, this is the *Observation* activity. This was carried out by conducting a literature review, investigating existing solutions for the problem statement and by interviewing potential end users and an expert in open source community health.

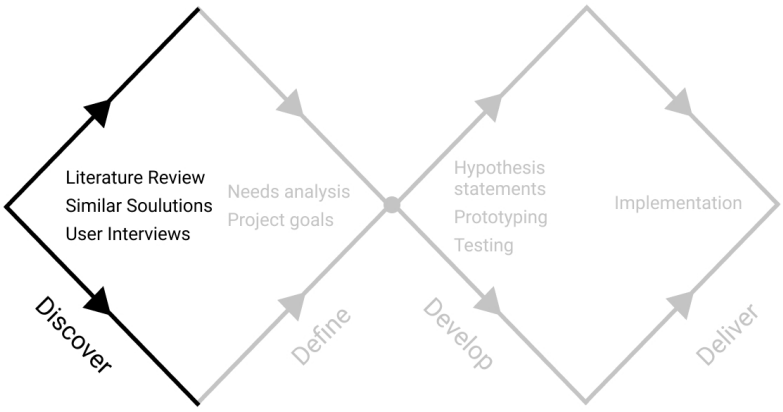


Figure 4.1: The *Discover* step in the double diamond design process and the activities conducted in this stage.

4.1.1 Literature Review

In the beginning of the development process, a literature review were conducted to get an overview of the field and to get the necessary context for the product. The main focus were literature regarding the processes and factors influencing the choice between different open source software. The review was mainly conducted using Google Scholar with the search terms: ”Open Source”, ”Open Source Software, ”Open Source selection”, ”Open Source intake”, ”Open Source health, ”Open Source software adoption” and ”Open Source evaluation”. Both backwards- and forwards citation search was used to find more relevant papers. The results from this review are presented in section 5.1.

4.1.2 Similar Solutions

Similar solutions to problems related to ours were investigated to get a feel for the current state of the art and inspiration. These were found during the literature study, the user interviews, and by searching OSS-related works. The results are shown in section 5.2.

4.1.3 User Interviews

One central part of the design process is to gain information and feedback to improve the design. This can be done with user interviews, at any stage of the design process. The purpose of the interviews is to understand the user's experiences, opinions, motivations, attitudes, and behavior concerning products and services. The intentions of the interviews could be different depending on where in the design process the interview is performed. Interview guides can be structured, semi-structured, or unstructured. The different setups will give a different kinds of information and answers from the interviewee. The interview guide can be tested using pilot interviews. [24]

We conducted semi-structured interviews during our process. This method was chosen since it is common in human centered design and a good way to understand the user in different situations. Since we are not the end-user of this product and we do not have a lot of experience in this topic, semi-structured interviews allowed us to explore the OSS selection process further. The interview guide was prepared in advance using Google Forms. The questions were divided into the sub-areas; process, factors, and source of information. The questions can be found in Appendix C. To evaluate the guide, pilot interviews were conducted with participants from the company. The pilot interviews can be seen as data points as well, but since the interviewee works at the company

the bias needs to be considered.

These semi-structured interviews were conducted in connection with prototype testing. They were conducted digitally via Google Meet. We booked the interviewees for a total of 45 minutes, where approximately half of that time was used for interviews and the other half for prototype testing. Both of the authors were present, one conducted the interview and the other wrote notes. The meetings were recorded.

Before each interview, an informed consent document was sent to the participant. This is to ensure that the interviewee is fine with the meetings being recorded and informed regarding what data was collected and how it was handled. The informed consent document can be found in Appendix C.1.

The participants were found both in the Debricked customer base and in our circle of acquaintances. The goal has been to get a good distribution in age and experience. All interviewees are listed in table 4.1 below. Along with a participant id, PX, current role, and company are shown. As mentioned before, pilot interviews were conducted, the participants in these interviews, who are employees at Debricked, are shown in table 4.2.

Participant	**Role**	**Company Size;** Employees	**Programming Language**
P1	CTO	25	JavaScript
P2	Co-founder, Developer	10	Python
P3	Principal Architect	700	Go
P4	Developer	60	JavaScript, Elm
P5	Architect	15	JavaScript
P6	Developer	7	Ruby
P7	Product Owner, Software security	150 000	-
P8	CTO	20	Php
P9	Senior Architect	2500	C#

Table 4.1: Participants; interviews

Content Analysis

All interviews were analyzed using direct content analysis, in accordance with Hsieh and Shannon's theoretical framework [25]. In this approach, also known as deductive analysis, key categories and concepts that we were looking for before conducting the interviews are identified. This synergizes well with a

Participant	Role
Pilot1	Data Scientist
Pilot2	Back-end Developer
Pilot3	Senior Lecturer

Table 4.2: Participants; pilot interviews

semi-structured interview approach as the participants can discuss concepts unprompted and then be asked to elaborate on topics that are especially relevant for our research questions. The key categories we were looking for were based on our research questions: metrics, process, benefits and drawbacks with current sources of information.

4.2 Define

The *Define* phase of the design process aims to understand the users, how the users' needs and the problem align, based on the findings in the *Discover* phase. This is distilled into goals for the project, presented in section 6.4.

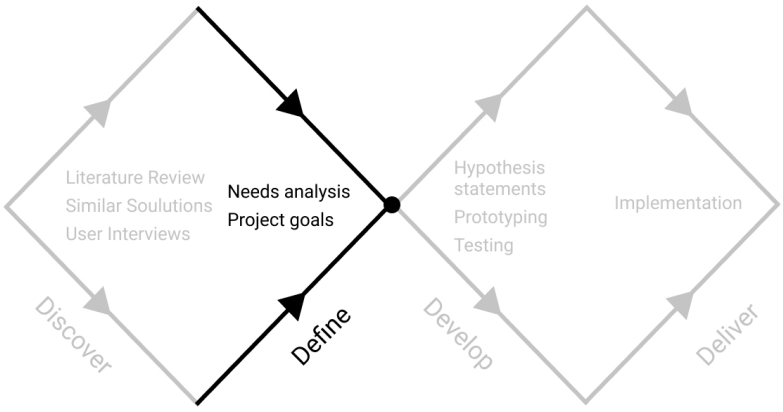


Figure 4.2: The *Define* step in the double diamond design process and the activities conducted in this stage.

4.2.1 Needs Analysis

Need analysis is a method to gain a better understanding of and qualitative information about the user. A need is defined as a user's goal-oriented behavior and way of acting. Needs remain while solutions are timely. The information gathered from the literature study and user interviews were converted into needs. The focus was to find the actual needs, not specific solutions. Focusing on solutions can limit the process. [24] The needs analysis was performed by brainstorming.

4.3 Develop

In the *Develop* phase different solutions are developed and tested iterativelty. A prototype-driven working method was used. *Develop* is divided into three iterations where one prototype is developed and tested in each.

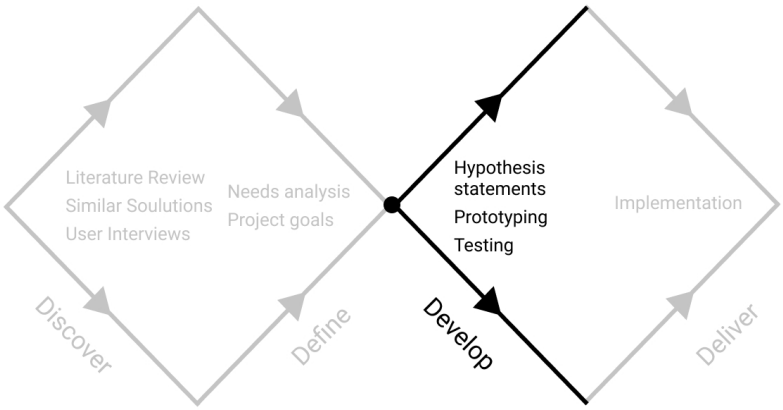


Figure 4.3: The *Develop* step in the double diamond design process and the activities conducted in this stage.

4.3.1 Hypothesis Statements

Gothelf and Seiden [26] present hypothesis statements as a tool in UX-design for expressing assumptions in testable form. It is a systematic way to work with building and testing prototypes by framing them in the context of testing and validating these assumptions.

A version of Gothelf and Seiden's framework of hypothesis statements were used as a systematic approach for developing and testing prototypes. The statements consists of several elements. **Hypothesis:** An assumption of what is believed to be true, directed at specific areas of the product or workflow. **Test:** The signal we seek to help validate or invalidate the hypotheses. **Result:** The result of the test. **Design adjustments:** Changes in the product that are believed to drive wanted outcomes, based on the assumptions.

In practice, several hypotheses were formulated based on our assumption of customer needs and principles of interaction design. Prototypes were then designed to test these hypotheses.

4.3.2 Prototyping

Prototyping is a method used in the designing process to help finding any design issues and test the practicability of the current design by developing an inexpensive and scaled down version of a product. There are two different levels of prototyping, *Low Fidelity Prototyping* and *High Fidelity Prototyping*. The low-fidelity prototype is a quicker and inexpensive while the high-fidelity prototype is closer to the final product, both in functionality and appearance. [27]

In this project, only low fidelity prototypes were developed. This was chosen mostly due to the limitations in time and previous research.

Low Fidelity Prototype

The prototypes are developed in Figma, an interface design tool. The prototype is improved every iteration according to the feedback received in the users. These prototypes were partly clickable, and populated with placeholder and example data.

4.3.3 Testing

User Testing

The prototypes were validated through user tests. These tests were conducted on the same occasion as the user interviews described above. The user performed the tests after having discussed their current procedures and workflow for selecting OSS packages. During these tests, the user was told to use the tool to find and evaluate an OSS package to solve a certain problem. In the second iteration of the prototype, this problem was evaluating the front-end JavaScript framework *react*. The participant was told to think out loud and

show us how they would interact with the tool.

Expert Testing

The prototypes were also validated through expert tests. A domain expert in Open Source communities, Johan Linåker from Lund University, was shown a version of a lo-fi prototype. He was mainly asked to give feedback and ideas regarding which metrics are relevant and helpful for evaluating the health of open source communities.

Heuristic Evaluation

Heuristic evaluation is an evaluation method for user interface design, where a small set of evaluators judge the design's compliance with general usability principles [28]. This design evaluation was performed with the design lead at Debricked on the final prototype.

4.4 Deliver

In the *Deliver* phase the final prototype is implemented and presented.

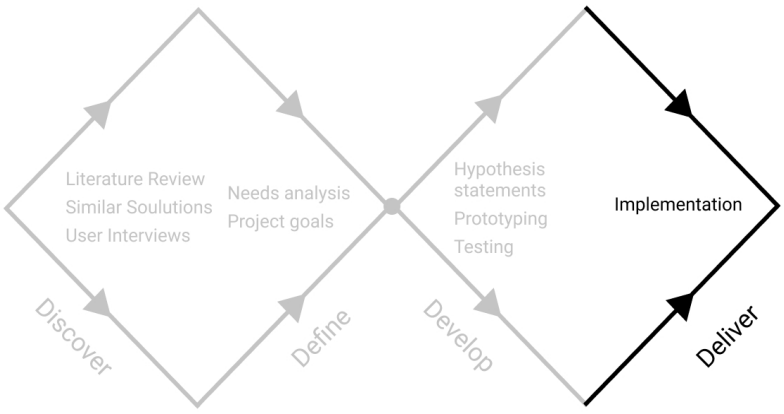


Figure 4.4: The *Deliver* step in the double diamond design process and the activities conducted in this stage.

4.5 Applied Technologies

Miro

Miro is an online whiteboard that can be used for workshops, planning, mind mapping etc. [29]. In this thesis, it was used to facilitate discussions, perform brainstorming and to visualize the selection process and the needs analysis.

Figma

Figma is an online design and prototyping tool based on vector graphics [30]. It was used to create partially clickable, low fidelity prototypes in iterations 1, 2 and 3.

5 Discover

This chapter includes all findings from the literature review and the user interviews, along with an analysis of the UX designs of competitive solutions. This chapter aims to mainly answer research question 1 and 2. The analysis of similar solutions can be seen as a pre-study for research question 3.
RQ1) What is the selection process for different use cases when adopting new open source software?
RQ2) Which metrics are important for different use cases when evaluating open source software?
RQ3) How can a user interface be designed to facilitate the selection process and to communicate a business value of Open Source Health?

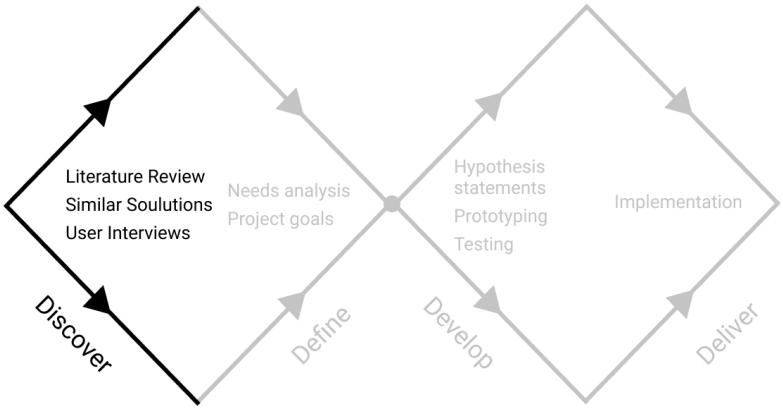


Figure 5.1: The *Discover* step in the double diamond design process and the activities conducted in this stage.

5.1 Literature review

OSS evaluation and community health is an active research area. Several relevant articles were released just during the time it took to write this thesis. Most articles found are focused on tracking or prescribing metrics for evaluating OSS, often from an organizational perspective.

RQ1: What is the selection process for different use cases when adopting new open source software?

The overall process for open source selection is not as actively researched as which metrics are considered. Some articles do mention selection processes, though it's not the main research question.

A 2021 interview survey by Jansen et. al. [31] interviewed software engineers with different roles and backgrounds about trust in software development. One question was about their selection process when importing third-party components. They found that there was no universal procedure that all participants used. Most of them had an informal process where the common sense of the individual developer was a main deciding factor, with some informal discussion in the team as a sort of social security. A minority of participants, people working in sectors such as defense or finance, had formal procedures with defined roles and responsibilities when selecting third-party software components. The paper included some quotes from participants. We were able to get some idea of what the participants prioritized by analyzing the language used. For example, "So the most important one is that it is compliant with your current framework." is a clear indication of what one participant considers the most important factor. Analyzing all quotes this way indicated that a common process was to first identify requirements, such as compatibility and license, then identify relevant candidates, and finally to evaluate those candidates based on several metrics.

A relevant factor when describing the selection process is where developers look for information. A 2021 survey of 23 experienced developers by Li and Moreschini et. al. [32] asked this question, amongst many others. Their results are shown in table 5.1.

The information gained from reviewing relevant articles was compiled to a process flow chart in Miro, seen in figure 5.2. The main process we identified was 1) Identify a problem 2) Identify requirements for an external solution 3) Identify potential candidates 4) Evaluate candidates. In the flowchart, there are also examples of specific factors that are commonly mentioned for each

Source of Information	total mentions	% of mentions
Version Control Systems:		
GitHub	23	100
GitLab	1	4.3
SourceForge	1	4.3
Bitbucket	1	4.3
Issue tracking Systems:		
GitHub Issues	19	82.6
Jira	1	4.3
Question and Answer Portals:		
StackOverflow	12	51.2
Reddit	12	51.2
Forums and Blogs:		
Medium	5	21.7
Hackernews	5	21.7
Security		
NVD	14	60.9
CVSS	2	8.7
CVE	1	4.3
CWE	1	4.3

Table 5.1: Sources of information reported by participants in a 2021 paper by Li and Moreschini et. al., table 3 [32]. The table is recreated from data in the same paper.

step, in the lighter yellow boxes. For example, in the step Identify candidates, asking colleagues, searching the web, and looking at forums are frequently mentioned tools.

RQ2: Which metrics are important for different use cases when evaluating open source software?

Research in this area is very active. Amongst others, two different systematic literature reviews were published in 2020, one by Lenarduzzi et al. [33] and one by Sanchés et al. [34]. Both measured the number of papers that mentioned different factors. Some of the most commonly mentioned factors were: *Support, Compatibility with the current setup, Code quality, Reliability, Cost, License type* and *Community*.

Lenarduzzi et al.[33] identified 60 primary papers. They grouped the literature

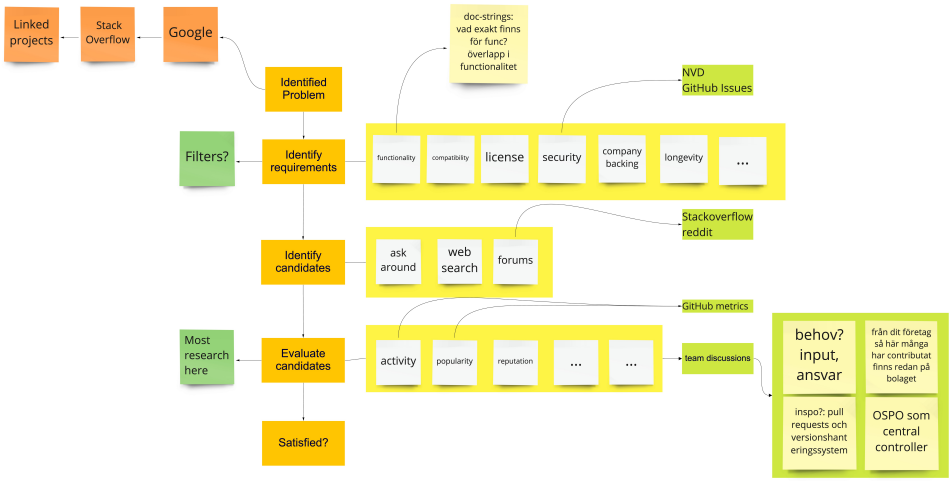


Figure 5.2: Brainstorming session about the selection process, based on findings in the literature review.

into two categories; model proposals and surveys/lessons learned. The results from the surveys and lessons learned are interpreted by Lenarduzzi et al. as representing what practitioners consider useful information. Data from the paper is shown in figure 5.3. By comparing the two categories, as shown in the figure, they found that the literature showed a discrepancy between what factors were considered useful by practitioners and what models proposed. They also found that it is a very time-consuming process to find data on the factors that are required to evaluate an OSS product.

Sánchez et al. [34] identified 54 primary studies. They grouped the identified factors into three categories: technological, organizational, and economical factors. Here, organizational refers to internal factors in the organization that is adopting OSS. Data from the paper is shown in figure 5.4. They found that technological and organizational factors were considered equally important, as they were mentioned in 91% and 93% of all papers respectively. Economic factors were mentioned less, in 60% of studies. They also concluded that IT managers are neither using any tool nor procedures that allow them to evaluate the adoptions of OSS solutions.

A 2021 survey exploring factors and measures to select open source software by Li and Moreschini et. al. [32] was released after both these systematic literature reviews. They interviewed 23 experienced developers. They found 8 main factors and 46 sub-factors. Each interviewee considered between 1 and 21 factors, reporting an average of 2.35 factors per interviewee. The most commonly mentioned factor was *License*, and the factors considered

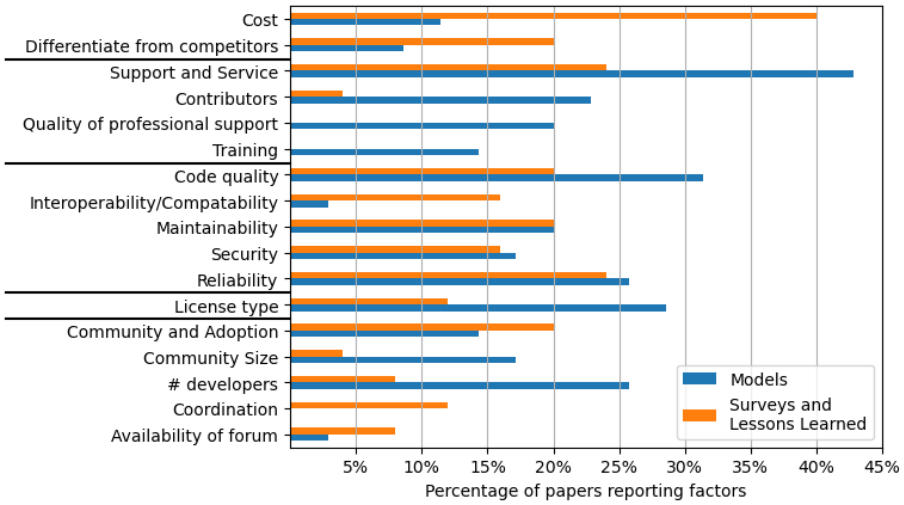


Figure 5.3: Percentage of papers analyzed by Lenarduzzi et al. [33] reporting factors. Figure created by the authors based on that paper. Some additional factors examined by Lenarduzzi et at. that were not considered relevant for this thesis by the authors are not shown here.

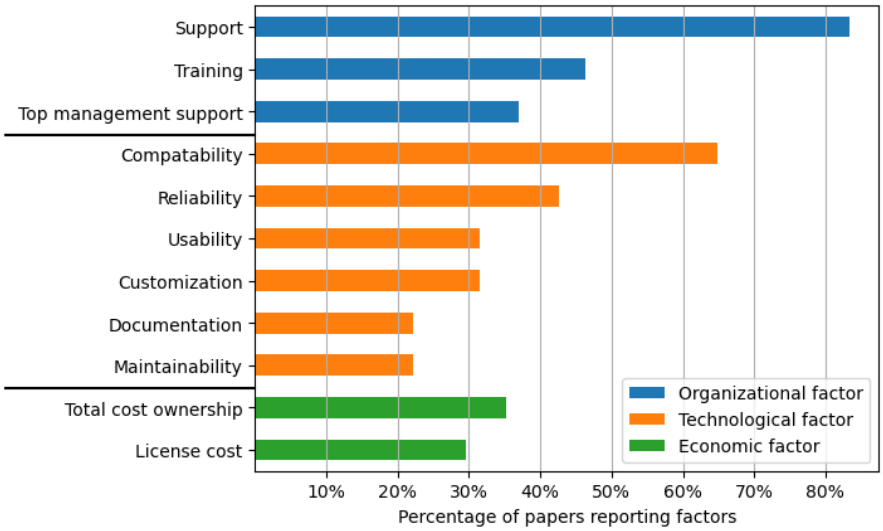


Figure 5.4: Percentage of papers analyzed by Sánchez et al. [34] reporting factors. Figure created by the authors based on that paper. Some additional factors examined by Sanchéz et at. that were not considered relevant for this thesis by the authors are not shown here.

most important were: *Community Support and Adoption, Performances* and *Perceived Risk*. None of the participants mentioned economic-related factors. In total, the participants mentioned 110 different measures to evaluate these factors.

5.1.1 Main takeaways

Based on the user interviews, we propose a general intake process with the following steps:

1. Identified problem
2. Identify requirements
3. Identify candidates
4. Evaluate candidates

GitHub appears to be the main source of information for developers when evaluating OSS. Many developers also look at the NVD (National Vulnerability Database), StackOverflow, Reddit, and other forums and blogs.

Some commonly mentioned metrics are: *Support* (especially in the adopting organization), *License*, *Community and adoption*, *Compatability* and *Code quality*.

There seems to be a discrepancy between what factors evaluation models propose and what factors are reportedly being considered in practice. There are a very large amount of factors, subfactors, and measures mentioned in the literature, and it is mentioned that it would take a lot of time and effort to find data for these metrics.

5.2 Similar Solutions

There are several solutions available that are similar to the OSH tool. The approach varies a bit between the different solutions. In this section, the main competitors will be analyzed from a UX point of view as well as what unique value they add to their users. They were investigated to get a feel for the current state of the art and inspiration. Snippets of the different graphical user interfaces (GUI:s) can be found in Appendix B.

RQ3: How can a user interface be designed to facilitate the selection process and to communicate a business value of Open Source Health?

npm.js

Node Package Manager, npm, is a center of open source packages in JavaScript and the largest software registry in the world with more than 1 million packages. [13] The main point of the website is to facilitate the installation and usage of specific packages rather than to analyze and compare them. But they do show some metrics, which are probably chosen very carefully for their impact. The main metric shown is a number and graph showing weekly downloads.

The design is reminiscent of GitHub with general information shown in a column to the right of the page and a use of tabs to structure the page.

Open Source Insights - deps.dev

deps.dev is a website to the underlying project ”Open Source Insights” by Google. The goal of the project is ”to help developers to better understand the structure, construction, and security of open source software packages” [35]. The website was launched in June 2021. The focus of the project is dependencies and dependents, where a full, detailed graph of the dependencies and their properties is generated.

The deps.dev design is minimalist and clean. The content is divided into five different tabs, Overview, Dependencies, Dependents, Compare, and Versions. This is a good way to split information into different categories and at the same time keep down the amount of information that is viewed at the same time. When searching for packages, you have to search by name and it is not possible to search by functionality. There are no filter functions for the search except for which package manager, which does not have to be specified. This makes the user able to search broader.

Libraries.io

Libraries is a project from Tidelift that "helps you find new open source packages, modules and frameworks and keep track of ones you depend upon" [36]. A snippet from Libraries.io can be seen in figure 10.3.

There is a lot of good information presented, such as statistics, installation information, and links to documentation. One unique piece of information is a list of popular users of the package at the bottom right. The SourceRank score is some sort of quality indicator that takes several different data points

as input. It is also a source of confusion, as the score is not contextualized. It is hard to interpret whether the score of 34 for react is good or bad for example. One redeeming factor is that the score can be explained and broken down by clicking on it which clearly explains what it is based on.

Synopsys Black Duck Open Hub

Black Duck Open Hub, formerly Ohloh, is a website that aims to index and show data about open source projects. It was originally launched in 2004. A snippet from Open Hub can be seen in figure 10.4.

It has a lot of good information and great contextualization. There are several interesting design choices, which in our opinion are great ideas. For example: presenting some essential pieces of information in snippets of common text under ”in a nutshell...”, clearly explaining what a certain license means in practice, and having clear and interactable graphs to show how metrics have changed over time.

There are intentions made to make it a community hub where users of Open Hub can show their ratings and usage of projects. There is not a lot of activity there, for example React having only 12 Open Hub users, which gives the website a sense of being outdated and not widely adopted by the developer community.

Snyk Advisor

The Snyk Advisor tool is a tool to find and evaluate open source packages. Currently, the tool is in a beta version which has been available since October 2021. The main focus of the tool is security issues and vulnerabilities. A snippet from Snyk Advisor can be seen in figure 10.5.

The overall design is nice and clean. The page content is divided into different categories; Popularity, Maintenance, Security, Community, and Package. This kind of view makes it easy for the users to navigate and find the information they are looking for. In the popularity and Maintenance category, nice and informative graphs are shown. Although it would have been nice with some more graphs and time series. At the top of the page, an overview of the project is shown. This overview includes a description of the package, links to package manager and GitHub, license information, a list of similar packages, and a health score. The search functionality is limited, it is not possible to add filters and you have to specify which package manager you use, which is good in most cases, but sometimes this will complicate the *Discover* process for the user. Also, the search is only for direct matches, i.e. you have to search for the name of a package to find it.

stackshare.io

StackShare is a service that aims at facilitating general tech stack discussions and comparisons. In the context of the problem area of this thesis, they are mainly focused on comparisons between similar packages and on showing opinions of real users. A screenshot is shown in figure 10.6.

StackShare really focuses on usage statistics and user sentiment. One unique and interesting feature is showing "Decisions about ..." on the page. This section has stories of real tech stack decisions written by real users. This gives some very practical insight into someone else's decision process and sometimes experience with the component. Another unique feature is the "Pros" section, where users can write and vote on what they think are specific strengths of a package. With the visible votes, this is one way to collect and show some sort of developer consensus regarding a package.

This user-driven approach is great if the service is widely used and for popular packages. But there is a big risk in having little to no data on not so commonly used packages.

npms.io

npms, which stands for npm search, is an explicit attempt to improve the search functionality of npm. The main improvement is to assign each package a score and ranking the search results by that score. The score is calculated based on three submetrics: Quality, Maintenance, and Popularity. [37] The fact that this much effort has been put into improving this aspect of npm indicates that there is a large need and demand for this sort of tool.

In terms of design, the website is very clean and simple. It's mainly an aggregator, so clicking a package links directly to its npm or GitHub page. It gives a very clear score, both an overall score and for the submetrics. It's clearly indicated where the user can find out exactly how the scores are calculated, and the interested user can get a detailed explanation and even look at the source code.

5.2.1 Main takeaways

In summary, this is a rather active area, with a lot of competing solutions. Even Google recently released a product in this space. Many of the solutions have their own niches and some great ideas for informative metrics. Combining ideas from these solutions is probably a sensible start for a new tool.

I terms of design, many of them are pretty similar. The majority has an

interface that uses tabs to structure the page. This implies that this is a common design choice that the intended audience is familiar with, increasing the usability.

There seems to be an unmet need for good search functionality. Most of the existing solutions only support search queries for the exact names of packages. The fact that npms exists shows that there is some level of demand for improvements here.

5.3 Interviews

The interviews were carried out during the whole *Discover* phase. The results are presented in conjunction with research questions 1 and 2.

RQ1: What is the selection process for different use cases when adopting new open source software?

This part of the interview investigates the selection process today. Mainly at the interviewee's current company but sometimes also in past jobs and for personal projects.

Pilot Interviews

This is the interview result from Pilot1 and Pilot2. Pilot 3 did not participate in a full interview and is therefore not included in this summary. While reading the results from these interviews, bias needs to be taken into consideration since the Debricked team knows about the OSH-model and problem area.

In the selection process today, all pilots stated that the first thing they do is to search on Google. Pilot1 usually wants to find all possible solutions and use several forums, like Stack Overflow and Medium, to evaluate the projects according to activity and popularity, and then look at the project's GitHub page. Pilot2 wants to know if the current package is good and looks at GitHub to get a brief idea about the project. According to a team policy, an imported package shouldn't have any known vulnerabilities and it should work with the current setup, without crashes.

When importing smaller components, Pilot1 and Pilot2 described that each developer takes care of the selection process and makes all decisions. For bigger architectural decisions, research is done within the team and discussed afterward, according to Pilot1. Pilot2 describes that some discussion might

be during sprint planning or code reviews.

Both pilots see advantages in the current way of selecting OSS, especially in the development speed with no hierarchical decisions. Pilot1 mentioned that one disadvantage is the security risks and future problems a non-well-grounded decision can lead to. The process could be improved by a well-defined policy/requirements or checklist to help each developer make a good decision, according to Pilot1.

User Interviews

The most common process, six out of nine interviewees answered, is not to have a formal process for selecting new OSS. P6 described the overall process like: "Just choose and install the package. At code review a peer will review the changes".

During the interview, eight of nine interviewees described that the selection process starts at Google to search for the specific problem. P6 stated that if the problem is generic, you can assume that someone else has had the same problem and hopefully solved it. P8 mentioned that package pages are not based on problem/functionality, which makes them wrong in the search phase of the selection process, and that P8 sometimes browse by authors to get inspired. Some authors are more active in a particular community. P9 tries to find how other known companies are solving the same problem. P4 described that there is a difference in the general quality of projects depending on the programming language. JavaScript has a massive amount of packages but is often of low quality. ELM, on the other hand, has fewer but of higher quality. This affects the amount of research needed before choosing a package and, therefore, the process.

Five of nine participants described that after searching on Google, they read forums or blogs, like Stack Overflow, Reddit, and Medium, to find a solution to their problem. Where to look seems to vary a bit between different languages. P8 said that the main thing to look for is if the package has the desired functionality and solves your problem.

When evaluating the packages, the main factors are activity and popularity, which five of eight participants mentioned. What factors to look for when evaluating these are described in the next section below. The primary sources for evaluation data are GitHub, project documentation, package manager, and forums/blogs. In many cases, the evaluation step is about the gut feeling. P1, P3, and P6 did all talk about the selection process as a way to understand the possibilities and get a sense of the different opportunities. This part of the process involves different types of questions; Does the package work? Is the

solution suitable for our needs? Will it be maintained over time? Do we need to put resources ourselves in developing? P1, P3, and P6 also mentioned that they look at the code and the APIs to evaluate their quality.

During some of the interviews, the difference between smaller and bigger imports was discussed. The most significant difference between different kinds of imports is the decision-making. P5 described that there is almost always some discussion in the team, often connected to code reviews.

Most of the interviewees, seven of nine, stated that each developer makes the decision. P1 described that for components that will not affect the architecture or the whole team, you just install and test the component. If you like it, you are done. In P2's case, no comprehensive evaluation of the package is done, the only important thing is that the package solves the problem and that some standard metrics look okay. This is similar to P3 and P4, who mentioned that for smaller imports, they use ad hoc processes are that are up to each individual, based on their judgment. P5 reviews the code directly, looking at the complexity, functionality, and how understandable the code is. P6, who just choose and install, stated that sometimes there might be a need to motivate a choice. P8 mentioned that each person does their research, and the intake will then be verified in the pull request review.

When selecting OSS for bigger architectural decisions, more research is done and there are more discussions within the team. In P1's and P2's teams, decisions are made through consensus, after some research and possibly some group discussions. P3, who works as principal architect, described that the architects discuss and try out a few candidates before deciding. Officially, the chief architect has the final word when making a decision, but the whole team is forming a consensus. P4 stated that there is a collective team effort. The team members do their research, often split into different responsibilities, then discuss together and form a collective decision. In P5's case, some preliminary research is done for critical imports, and sometimes an external tool is used to assess the risk of vulnerabilities in the code.

The most commonly mentioned advantage of current processes is that there is no bureaucracy. Six participants mentioned this. P8 described that the most significant advantage of the current process is that the teams can be autonomous. When looking at disadvantages, the most commonly mentioned is the risk of making wrong decisions and adopt vulnerable components. This might lead to extra work and a new intake process. P2 described a risk with the current process that you can miss alternatives and potential problems as it is only up to one person. P1, who is working with frontend, said that there might be a risk if you do not look up the number of dependencies because of

speed reasons for web apps.

Two of the interviewees described that a security department checks all external software. P7, who works within security at a global company, described that developers in their team make their own decisions and that there are no overall restrictions or particular guidelines regarding OSS. The security team works continuously to make developers aware of the different risks with OSS, and they check all third-party code that is adopted to the codebase. This check is mainly performed using Synopsys Black Duck to find and solve vulnerabilities, check licenses and dependencies. The security team is working on a checklist for the developers to use when selecting new OSS. The goal of the checklist is to help the developers with the evaluation without controlling them too much. Also, P7 stated that some kind of tool would be valuable for the developers in the selection process to perform automatic analysis, component checks, and some dependency analysis. But, there are a lot of requirements that have to be satisfied. For example, developers do not want to leave GitHub; this kind of extra tool has to be compatible with the current workflow. It has to be convenient, easy to interact with, and easy to understand what is checked or not. P7 thinks that Snyk has made a good approach here. The main thing that you want to know from this kind of tool, according to P7, is; what is the problem, and how do we solve it? Most of the risks in this kind of third-party component are pretty small and don't affect larger companies in the same way as smaller companies. As a developer today, it is hard not to create any vulnerabilities in the code you write.

The information gained from the interviews were used to modify the process flow chart created after the literature review, seen in figure 5.5.

RQ2: Which metrics are important for different use cases when evaluating open source software?

In this part of the interview, important metrics are discussed and what specific factors the participants consider in their process. Along with what sources of information are commonly used to find these metrics.

Pilot Interviews

In this section, the interview result from the three pilot interviews is summarised. While reading the results from these interviews, bias needs to be taken into consideration since the Debricked team knows about the OSH-model and the problem area.

Pilot1 starts by evaluating the functionality of the package and if it solves

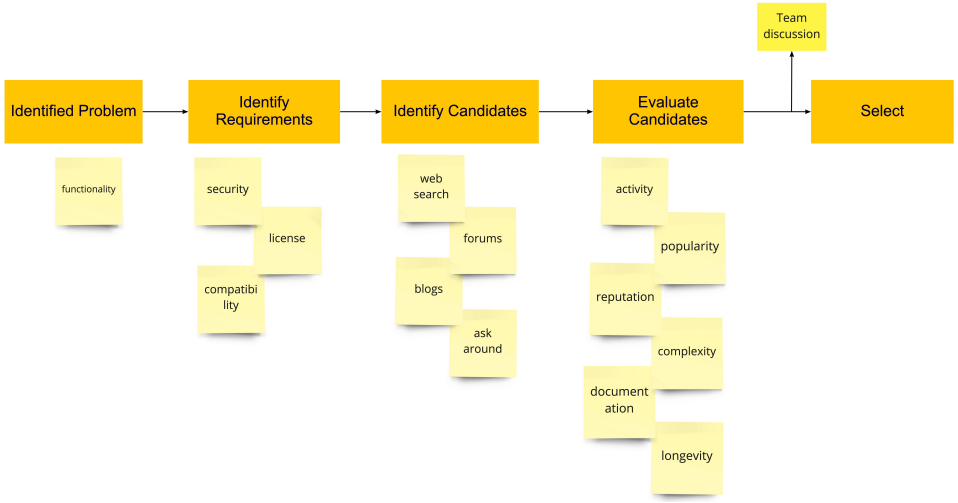


Figure 5.5: The selection process based on user interviews and literature review, created during a brainstorming session. Main process in darker yellow text boxes.

the problem. Pilot2 looks if the package is compatible with the current setup and at the package's functionality. These factors have to be green to even consider the package. When evaluating the package, Pilot1 looks at metrics indicating the activity in the project. These metrics are often time since the last commit and number of merge requests. If the community is non-active, the package requires a deeper investigation, checking the maturity of the project. Pilot1 wants to know if the package is known among the dev community. Pilot2 focuses on the popularity of the project by looking at the number of users. Also, Pilot2 likes to see some code examples to help to understand the functionality of the component, but this can be hard to find. Pilot3 stated that the most important is to convert data into information, this would make it easier for the user to interpret the data and make a more precise evaluation of the package.

The main sources of information used during the selection process are Google, GitHub, and Stack Overflow, which are mentioned both by Pilot1 and Pilot2. In addition, Pilot1 uses Medium, pip documentation for Python packages, Reddit, and documentation. The documentation can sometimes be hard to find.

User Interviews

Most participants stated that they look at 1-4 specific metrics. These metrics have been placed into overarching categories by the authors; popularity,

activity, license, quality, and others.

Many participants emphasized that evaluating OSS components are a complex problem with a lot of influencing factors. As mentioned before, three interviewees (P1, P3, P6) explicitly stated that they go mostly by gut feeling based on some sort of metrics when making decisions. P6 for example looks at all resources that are available, such as activity, opinions of other users, issues, quality, usability, and age of the project. P3 has a similar approach and described that they don't look for hard metrics specifically, especially when developing personal projects. Instead, they try to get more of a sense of the community consensus regarding the packages. This is easier for languages and ecosystems where they are more familiar. In those cases, they might recognize specific authors, know where to look in forums, and recognize good or bad design patterns of the code. Several participants indicate that context is important. P1, working mostly with web applications in JavaScript, said that the size of- and the number of lines of code in a package is important, but mostly for code in production. P3 states that they prioritize different metrics for personal projects compared to professional use. For professional use, the popularity and expected longevity of a project is important. For personal use, they prioritize based on soft metrics, like the sense of community consensus and potentially recognition and opinion of the author of a package.

Many interviewees mentioned that they often start with looking at metrics that indicate package popularity. A list of all specific metrics mentioned is shown in table 5.2. P2 stated that they start with looking at the number of users as that speaks for a lot of other factors. The underlying need is to know if a package will survive and keep being developed. P3 said that if there are several alternatives, they choose based on popularity.

Indications that a package is actively being developed were also mentioned frequently. P8 mentioned that their underlying need is to know if a package is actively maintained. Last update was the single most mentioned specific metric. Many participants look at the GitHub issues of a project.

P6 mentioned that the number of issues in themselves is not that interesting, but the contents and the time it takes to be dealt with are. P2 looks at open issues on relevant features and if there are very many inactive issues. P4 and P7 stated that if a project has been inactive for too long, both said two years as an example, they won't consider that package.

About half of the participants specifically mentioned going into the source code or API to gauge the quality of it, especially for smaller packages where this is more viable.

When asking the question of if there is a ”dealbreaker” - or some single factor that would make them not consider a package at all, licensing was the only factor mentioned by any participant. It was mentioned by P1, P2, P3, and P7, who stated that a commercially viable license is a must. P3, who works at a large company, stated that they are very keen on keeping track of the licenses of installed components.

Metric	Mentioned by
Popularity	P2, P3, P4, P5, P8
Number of users	P2
Number of contributors	P2, P8
Number of maintainers	P4
Weekly downloads	P5
Number of GitHub stars	P8
Activity	P2, P4, P6, P7, P8
Last update	P2, P4, P7, P8
Open issues	P2, P6
Closed issues	P6
Speed of bug resolutions	P4, P6
Responsiveness of questions in community	P4
Quality	P1, P3, P4, P6, P8
Code	P3, P4, P6
API	P1, P6
Functionality	P4, P8
Usability	P6
Understandability	P9
Availability of test cases	P9
Complexity of code matches functionality	P1
License	P1, P2, P3, P7, P9
Security	P3, P5, P7
Other	P1, P2, P3, P4, P6
Size	P1
Lines of code	P1
If on standard repository page	P2
Long-term complexity	P3
Expected longevity	P3
Dependencies	P4
Age	P6

Table 5.2: Metrics mentioned by participants. The categories (e.g. ”Popularity”) have been interpreted and assigned by the authors.

Sources of information:

The participants were asked which sources they used to find information to find and evaluate OSS packages. The results are shown in figure 5.6. One

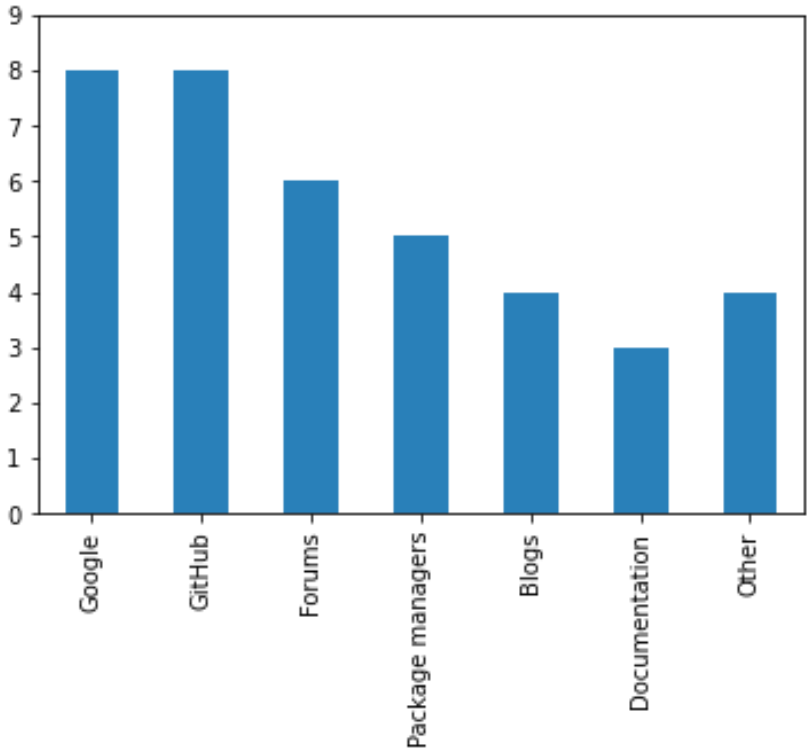


Figure 5.6: Mentions of specific sources of information when looking for and evaluating OSS packages. Out of a total of 9 interviewees.

outlier in these results is P6, a product owner of software security at a large company. The interview with this participant was in the context of general tools and metrics they recommend to other developers at the company. As such, the metrics mentioned are more geared to specifically helping security and licensing issues. They only mentioned Snyk and Black Duck and was the only interviewee to mention those tools. As such, they were also the only participant not to mention Google and one of two who didn't mention GitHub.

The rest of the interviews were more in the context of their process. Google was mentioned by all of them, in the context of where they start their search for new packages. npm, PyPI and Packagist are all package managers for different languages. Thus, 6 participants mentioned a package manager relevant to their language, making that the third most common source of information in these

interviews.

5.3.1 Main takeaways

The interviews indicate that there are a lot of variations in the selection process. To summarize the process, the most common way to find OSS is to search on the current problem or the desired functionality on Google. Everyone who participated described that some sort of evaluation of the package is done. The main part of the evaluation is to get a general gut feeling about the package based on different metrics. Some of the interviewees described that the evaluation includes comparing a few packages. Which metrics are taken into consideration varies a lot between different users, depending on factors like the programming language, current role, and kind of import. The most considered categories are popularity, activity, license, and quality. Table 5.2 shows all mentioned metrics, divided into these categories. The most common sources of information when searching for and evaluating OSS are Google and GitHub. Many participants use different forums and blogs as well. All mentioned sources are shown in figure 5.6.

6 Define

In the Define phase insights from the Discover phase are analyzed and a problem statement and design goals are formulated.

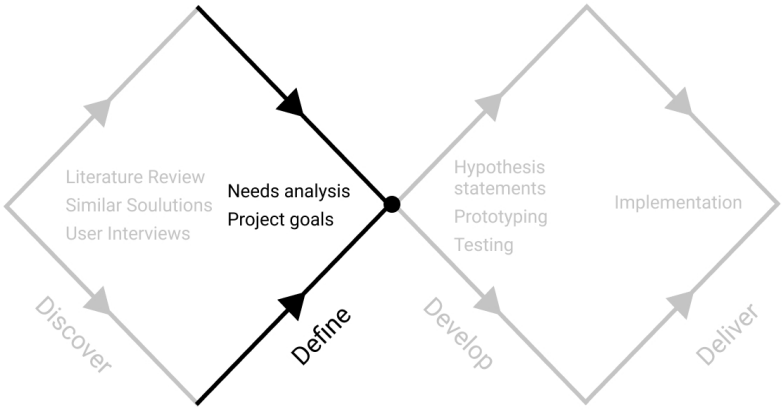


Figure 6.1: The *Define* step in the double diamond design process and the activities conducted in this stage.

6.1 Defining User Group

The OSH tool is mainly intended to be used by developers, developer's team managers, and open source program offices (OSPO). It is emphasized that this is an expert tool intended for use in a very specific domain. The end-user is therefore assumed to be well acquainted with core concepts and terminology from software development, open source projects, and version control systems.

6.2 Defining Test Group

In order to get relevant feedback from the user testing, participants that could be potential users were chosen in the same way as the user interviews. In table 6.1 all participants are listed. The participant id, Px, corresponds to the same id as in the interviews.

Participant	Role	Iteration
P1	CTO	1, 2
P2	Co-founder/Developer	1, 2
P3	Principal architect	1
P4	Developer	1, 2
P5	Architect	2
P6	Developer	2
P7	Product owner/Software security	2
P8	CTO	2

Table 6.1: Participants in the prototype tests. "Role" is the current role at the current company. "Iteration" is which iteration they have participated in.

6.3 Needs Analysis

The needs analysis resulted in figure 6.2 seen below. In the figure, the needs are shown in round green text areas, with some examples/areas listed around. The green box to the left shows needs that are specific to a certain user group.

Efficiency There is always a need of making things more efficient. This need covers a lot of different needs but is in our analysis efficiency corresponds to development speed within the team, the decision process regarding OSS, and how the company is working with OSS.

Usability The usability need covers the need for flexibility within the tool, the ability to implement it in the current workflow, and no addition of extra steps in the process. Since the OSS communities look different for different languages and the need for a tool is different depending on the current stack, company size, and user role, the tool has to be flexible.

Minimizing the risks with OSS As mentioned before, there are some risks with adopting OSS. One need of this tool is to help the users minimize these

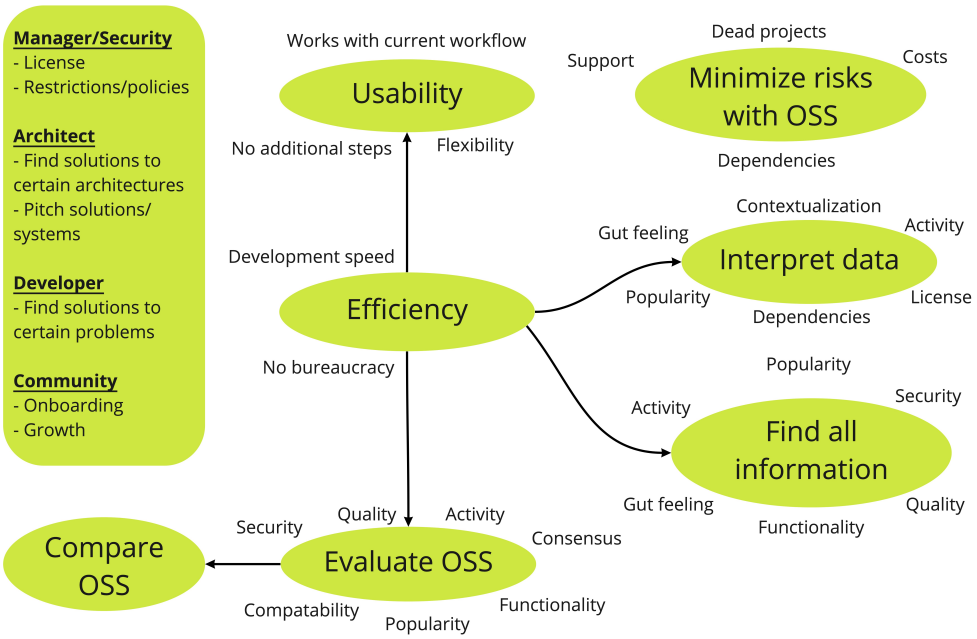


Figure 6.2: Needs analysis. This figure shows the result from the need analysis brainstorming session. The green bubbles are the overall needs with some examples of the certain need around. The big green box to the left describes specific needs in different user groups.

risks. These risks may include dead projects, costs, insufficient support, and long dependency lists.

Evaluate OSS This need is a part of the need of minimizing the risks with OSS by evaluating the OSS components. The evaluation is necessary to answer the questions: can I use this now and does it solve this specific problem?, and is this safe to use in the future? For the future, it is important to know if the community is stable and will be active in the future as well and if there are some critical dependencies.

Compare OSS The users have described that they want to know that they have looked at all possible solutions in order to find the best one. To examine which project is the best there must be an easy way to compare different alternatives.

Gather all information in one place To get a good and deep knowledge of a package, without spending too much time on searching for the information.

Interpret data To help users understand the meaning of the data and put several data points together, the evaluation process of OSS will be much faster and easier. The interpretation of the data will hopefully provide a rough gut feeling of the community health.

Find solutions based on problem This need is mainly for the developer role. When having a problem within software development, there are mainly two ways of solving the problem. Either you construct the solution yourself, or you see if someone else already solved it. For the second one, the need is to an efficient way to find these solutions. Since you already know the problem, the most common way to find it is to search for this problem.

Find OSS to a certain architecture or system As a software architect, you design the architecture and design of the product, which includes different systems, software, and platform choices. To make these choices, the architect has a need to, first of all, find these tools and then evaluate it and compare it.

Pitch certain packages/systems When importing bigger OSS packages that will affect the whole team, it is important that everyone likes to work with a certain solution. This creates a need of pitching the specific solution to the user.

Growth and Onboarding The whole OSS ecosystem relies on the possibility and will to contribute to projects. If there is too complex to join a project, no external developers will join the community.

6.4 Project Goals

Based on the information collected in the *Discover* chapter, the aims and design goals of this project are defined in this section. This project aims to investigate how new open source components are selected in companies and what factors affect the selection. The goal is to develop a proposal for a user interface that can improve that process and build trust from users.

There are several key goals that are considered vital for such a UX. One of the most important factors is to correspond to the current workflows. RQ1 of this report aims to investigate how that process looks, and the tool should facilitate that process for the users. All participants that were interviewed stated that they would not like to use any tool that introduced bureaucracy into their process, so it's also crucial to maintain that autonomy.

The main point of this tool is to show information. Consequently, the choice of what information to make available, and how to prioritize that information is fundamental to the value of this tool. The results from RQ2, what metrics are important when evaluating OSS, should therefore influence those choices. In addition to what metrics users consider today, this project also aims to evaluate innovative metrics that users haven't considered yet. This novel information is from the Open Source Health model. One major challenge of this design project is finding a balance between giving users what they are accustomed to and having them trust and utilize this new model. One important step in this process is for the tool to be transparent in how it operates in order to build understanding and trust in users.

Other design goals that are considered important based on the research phase include providing a flexible tool. The tool should aid in the selection of large frameworks as well as smaller packages. In order to promote both these uses, another goal of the tool is to show all information needed for users to make a quick decision for a small package as early as possible. Another aspect of flexibility is that the tool should be usable and valuable for users in different technical roles as well as managerial and administrative roles. In addition to this, it should also be usable considering different users' individual preferences, and the specifics of the ecosystems surrounding different programming languages.

In their systematic literature study, Sánchez et al. [34] found that internal support in the organization adopting OSS was the factor that was mentioned the most often. In that sense, it was concluded to be the most important factor reviewed. Based on this, another important goal of the tool is to show the availability of internal support and competence for an adopting organization.

In a broader sense, the goal is to provide contextualization for organizations using this tool. This is to not only provide value for individual developers, but also for businesses and organizations.

Based on the needs analysis and competitor analysis, it is concluded that there is an unmet need for improved discoverability when looking for OSS. 8 out of 9 interviewees stated that they start their search process by using google. Many of them mentioned that their search always starts with the problem that they are trying to solve. Yet, most of the analyzed similar solutions' main way of finding packages are by direct name search - which means that a user must already know what exact package they want to evaluate. Based on this, a goal of the tool is to provide ways for a user to discover relevant packages based on what they want it to do, not only based on its name.

In summary, the overarching design goals of the tool are:

- Correspond to the current workflow.
- Show relevant information, both information that users expect to see and new valuable information that is unique to this tool.
- Transparency in OSH-model interaction.
- Show all information needed to know if a package is compatible and healthy at first glance.
- Provide a flexible tool that can be used by different technical roles and for different kinds of OSS imports.
- Discovery based on functionality.
- Provide contextualization, such as showing the availability of internal support and competence, for organizations.

7 Develop

In this chapter is the Develop process described. Develop is divided into three iterations, where one prototype is developed in each of them. This chapter also includes the user validation and user testing of the prototypes.

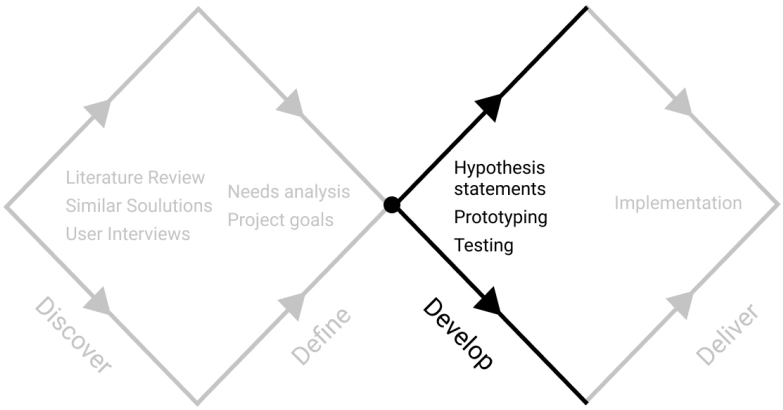


Figure 7.1: The *Develop* step in the double diamond design process and the activities conducted in this stage.

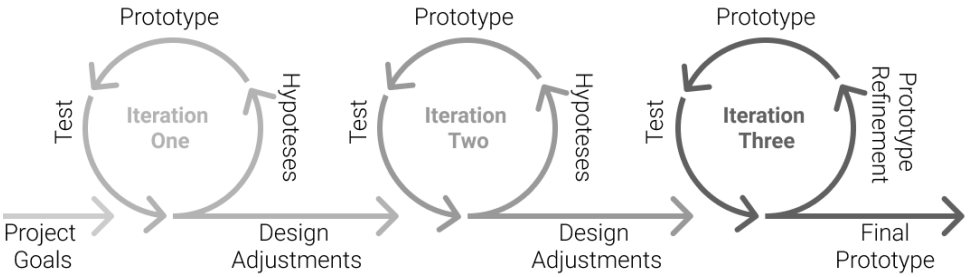


Figure 7.2: Process of the *Develop* phase. The figure shows the different actions taken during the different iterations.

7.1 Iteration 1

For the first iteration, a wireframe prototype was created in Figma. The background for the prototype was the literature review and the study of similar solutions. The main focus in this iteration was the OSS selection process. Several hypotheses were formulated, based on the findings from the literature review and user interviews. We intended to use this prototype to test these hypotheses by seeing how real users interact with the prototype. The hypotheses are:

H1.1: Transparency in how things are calculated is important.

H1.2: It's important to be able to search for packages by functionality.

H1.3: Users want to be able to go to primary sources (GitHub, package manager).

H1.4: It's important to be able to compare packages in a simple way.

H1.5: Contextualizing packages for companies is a nice feature.

7.1.1 Prototype 1

In this section, the main design and functionality of the first prototype are described. The full prototype can be seen in Appendix D.1.

Start

When entering the tool, the user will find the *start page*, figure 7.3. The main function of this view is the search functionality, therefore is the search bar located in the middle of the page. The user can search for both specific package names and the desired functionality. To make the search process easier and more precise, a *filter* functionality is available. This is found right under the search bar. Up in the left corner *navigation breadcrumbs*, a graphical control element, are available to simplify the navigation in the tool. In the right corner, a *list* functionality is available, the lists can be used to save projects in the list. This functionality is further described in connection to the list view. The list button is recurring in every view of the prototype.

Search Results Page

After performing a search, the user enters the *search results page*, figure 7.4, where all relevant packages are shown in a list/table. Each cell should contain enough information about the package to get a brief idea about the package's functionality and community health. To the left in the cell, a "more/less"

Figure 7.3: Start page

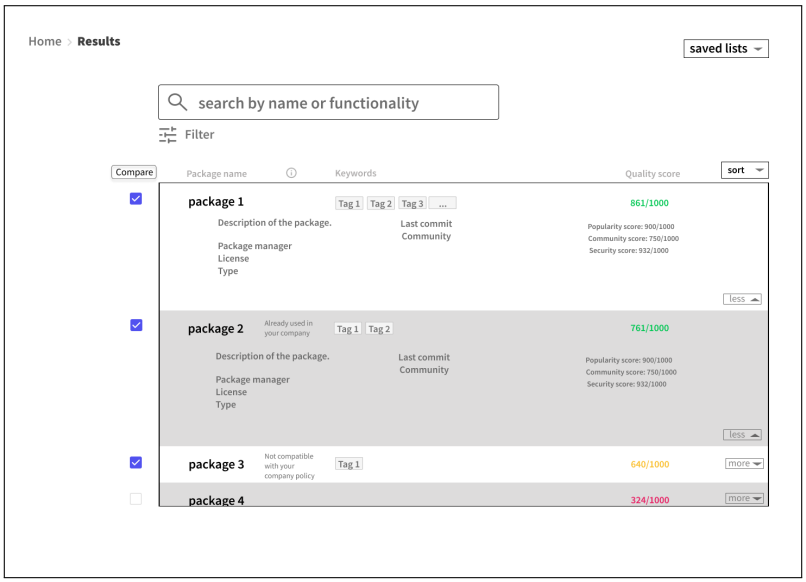
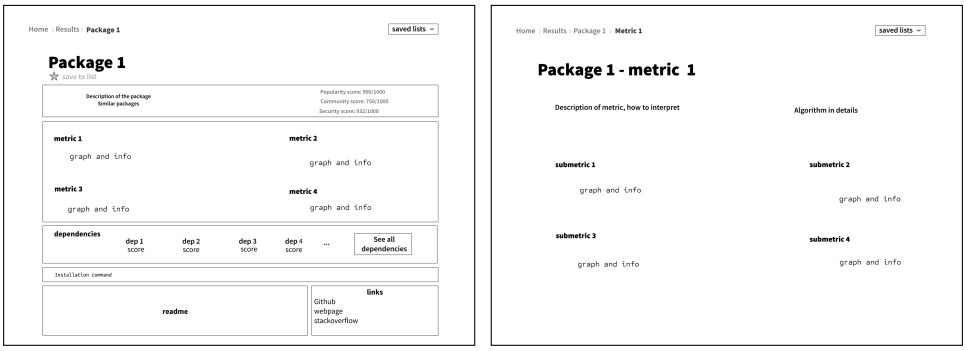


Figure 7.4: The search result in the extended version.

option is available. This extends the cell to view more information about the package. This can be used to do a quick comparison between the package in the list or just evaluate the packages that are interesting for further investigation. To find more information about the package, the user should click the desired

package name to enter the *package page*, which is described below. Above the list of results, to the right, it is possible to sort the search results in the list, e.g. the most popular, the most healthy, best match, etc. On the left side of the list, a *compare button* and some checkboxes are available. This is used to compare packages in the list. To compare packages you check the boxes for the packages you want to compare and then click *compare*, this will navigate you to the *compare page*, which will be described later. At the top of the page, a search bar is placed to enable the user to search again, for example if the result was not what the user was looking for. Under the search bar are filters available, they can be used when performing a new search or to refine the current search result.

Package Page



(a) Package page for package 1, start view.

(b) Package page showing the deep analysis options.

Figure 7.5: Package page for package 1.

The *package page* is used to evaluate a certain package. This page should contain all information about the project and the community. This view is divided into two pages, the *main page* in figure 7.5a and the *deep investigation page* in figure 7.5b. The *main page* contains placeholders for a description of the package, the possibility to save the package to a list, explore similar packages, view OSH model scores, list the dependencies of the package, copy installation command, read the README file, and for external links to primary sources. The biggest part of this page is the placeholder for different metrics describing the community health. The metrics are described by graphs or data points, and the user can click a metric to investigate it further in the *deep investigation page*. This page shows a description of the metric and its sub metrics.

Figure 7.6: The compare functionality with the comparison between package 1, 2 and 3.

Compare Page

To enable a deeper comparison of packages the *compare page* was designed. This page shows several relevant metrics side by side for the selected packages.

List Page

The *lists* are an easy way to share evaluated projects, facilitate team discussions or just save projects for later, like a watch list. The user should be able to activate email notifications, which will send an email if something critical happens with a package in a list.

7.1.2 Prototype Testing

This section presents the result from the prototype testing during iteration 1. The feedback is divided into sections, one for each page in the prototype.

Start Page

Figure 7.3. All users got going quickly from the *start page*, there was no confusion here. About half of the participants started with looking for a filter for their results, commenting that they wanted to filter mainly for language and package manager.

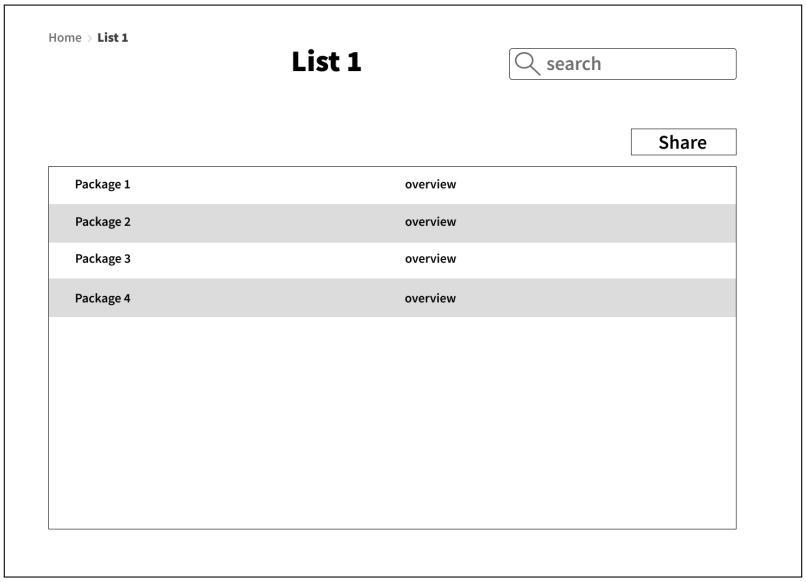


Figure 7.7: List with saved packages.

Search Results Page

Figure 10.11. All 4 participants wondered what the ”Quality score” meant and what it was based on. P1 wanted a less subjective term for the score, and proposed something like ”Business readiness” instead. They also wanted to see other objective metrics, such as a description of the package, number of GitHub stars and number of open/closed issues. P2 thought that it would probably be easier to interpret the score if the subcategories making up the ”Quality score”; popularity, contributors and security, were presented directly instead. In addition to getting a thorough explanation of the ”Quality score”, P4 would like to be able to set their own weights in how the overall score is calculated. This would allow them to prioritize different factors, such as security or popularity, after their own preference and situation. Some participants didn't click the ”more” button to show more information about the packages until after being prompted.

Package Page

Figure 7.5. There were differing opinions on how much information was interesting for the participants. P2 would like to see the regular metrics that they are used to from GitHub and their package manager here, such as number of stars and downloads. They could also see a value in having more and contextualized data. P1 thought that having sub-metrics might be too detailed information to be useful in regular usage, and had a hard time seeing what

data could be interesting at that deep of a level. P4 liked the option of doing a deep analysis, and also appreciated having links to first-hand sources like the GitHub page.

The participants had similar ideas of what they wanted to know about a package in a broad sense. Both P1 and P4 mainly wanted to know if the project is actively being developed and looked after, and if its well-known and popular among its users. All participants agreed that information about dependencies was important, but secondary to the main package. Most of them also stated that there are often too many dependencies to analyze all of them. Instead of trying to show all dependencies, P1 suggested to show just the number of dependencies. P2 and P4 both suggested to highlight just critical dependencies.

Compare Page

Figure 7.6. 3 out of 4 participants thought the *compare page* was very similar to the *search results page* and possibly redundant. P2 thought it could be useful when pitching package within the team.

List Page

Figure 7.7. None of the participants were interested in the list functionality. P1 would rather just use bookmarks in their browser. P4 thought that the domain is so active that there is no real point to save a list for later as there are likely new packages available by then.

General Feedback

All participants wanted more explanations of how the scores were calculated. Most of the participants also wanted to see different datapoints that they were used to see in their regular sources of information. P1, P2 and P4 all stated that having a great search function would be valuable for this tool and set it apart from similar solutions. They would like to be able to explore packages by functionality and what problems they can solve rather than just by names. P3 would also see a lot of value in some sort of expert curation or community driven recommendations of packages.

I general, the tool was sometimes a bit hard to navigate, as many participants took a while to progress from some pages. This was probably not helped by the very low fidelity and placeholder names in the prototype.

7.1.3 Expert testing

The following feedback is from Linåker, Postdoctoral Researcher in the Software Engineering Research Group from Lund University.

According to Linåker, it is hard to evaluate an OSS package using the metrics on the *search results page*. Most of the developers do not know what they need. To help developers evaluate OSS, all data should be converted into information. For example, last commit by itself does not describe the community activity but with additional information about the commit type, who committed, and a commit trend. Open issues are interesting if you know what kind of issues there are. From a community health perspective, lines of code are not interesting at all. GitHub stars do not correspond to popularity or community health. The star function does not necessarily mean that a package is actively maintained and popular among developers. It is interesting to read about known vulnerabilities, but you want to know more than just that they are known. The number of unique contributors and the number of contributions during the last six months tell the user that the project is active. The developer should be able to find out about productivity, robustness, and openness.

7.1.4 Hypotheses and Design Adjustments

H1.1
Hypothesis: Transparency in how things are calculated is important.
Test: No descriptions or infoboxes, only show placeholders and some data in the prototype.
Result: Everybody commented that they want see more details.
Design adjustments: Add more explanations of information.

H1.2
Hypothesis: It is important to be able to search for packages by functionality.
Test: Text in search bar.
Result: No feedback on the search bar text. Three out of four said that they would use the tool if the search worked "like Google for OSS".
Design adjustments: Add example data in prototype 2 based on this hypothesis.

H1.3
Hypothesis: Users want to be able to go to primary sources (GitHub, package manager).
Test: Package page, links to external sources.

Result: Good support.
Design adjustments: Add a isolated area with several links to primary sources. Also, add links to secondary sources, such as community forums or chats to meet the exploration need.

H1.4
Hypothesis: Users wants to compare packages in an easy way.
Test: Results page, primarily in the extended view and the compare page.
Result: The information on the compare page was felt redundant but might be good when pitching packages for others. Mostly important in large, architectural decisions, but usually appreciated for smaller decisions as well.
Design adjustments: Add more information to the compare page. Enable graphical comparisons.

H1.5
Hypothesis: Policy and contextualizing packages for companies increase the business value.
Test: Results page, showing if packages are already used within the company and if compliant with company policy.
Result: Some positive responses, not thoroughly tested.
Design adjustments: Implement a pop-up component where you can see exactly what the company policy is, and who to contact for more information. Also, implement the functionality for editing company policy.

7.1.5 Main Takeaways

There are several key points that are concluded from this first iteration of prototype testing. One is that it's vital that the Open Source Health scores are explained clearly to the user. Another conclusion is that there is only so much that is testable with placeholder data, especially to evaluate how metrics are interpreted in practice. One actionable insight is that it would be beneficial to change how the dependencies of a package are presented. A majority of the participants wanted to see less but more essential information, such as only showing critical dependencies.

There is good support for that the information flow of this tool fits into the participants' normal workflows. It can be streamlined and designed further to be more navigable and intuitive.

7.2 Iteration 2

Iteration two rely on the results from iteration one, the hypotheses and the user testing. The prototype for this iteration is more detailed than the first one and is partly populated with data. The focus in iteration two has been to investigate what information is to be given in the different parts of the tool. In this iteration we are consider the following hypotheses;

H2.1: Transparency in how things are calculated is important.

H2.2: Open source health, community health, is an important factor, and users can trust a numerical calculation of this.

H2.3: The user do not want to go between too many pages, the process should be relatively short.

H2.4: Contextualisation will improve the business value for organizations.

H2.5: Seeing other companies and projects that use a package helps build trust in that package.

H2.6: Being able to explore similar packages is appreciated.

7.2.1 Prototype 2

In this section, the main design and functionality of the second prototype are described. This prototype is built on the first, several pages are similar to iteration 1. The focus will be to discuss the changes. The full prototype can be seen in Appendix D.2.

Start Page

The first page of the tool is seen in figure 7.8, the *start page*. This view is similar to the one in prototype 1. A search button is added to the right of the search bar and a link to advanced settings under it. The advanced settings are used to update the OSH model weights before performing a search. On the top of the page a button, ”Admin Tools”, is added. This button is used to view and edit the company policy. This is only available when the user is logged in, and the goal is to in an efficient way help developers to navigate in ”bad” and ”good” projects.

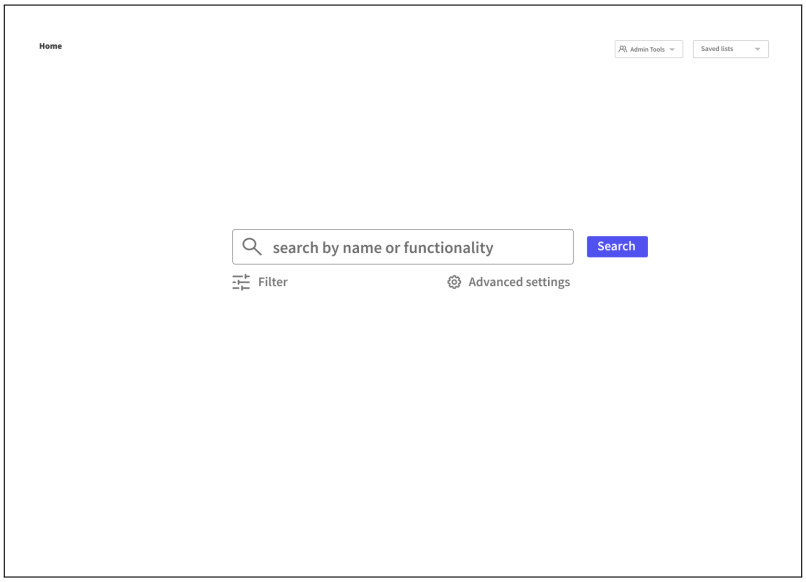


Figure 7.8: Start page iteration 2.

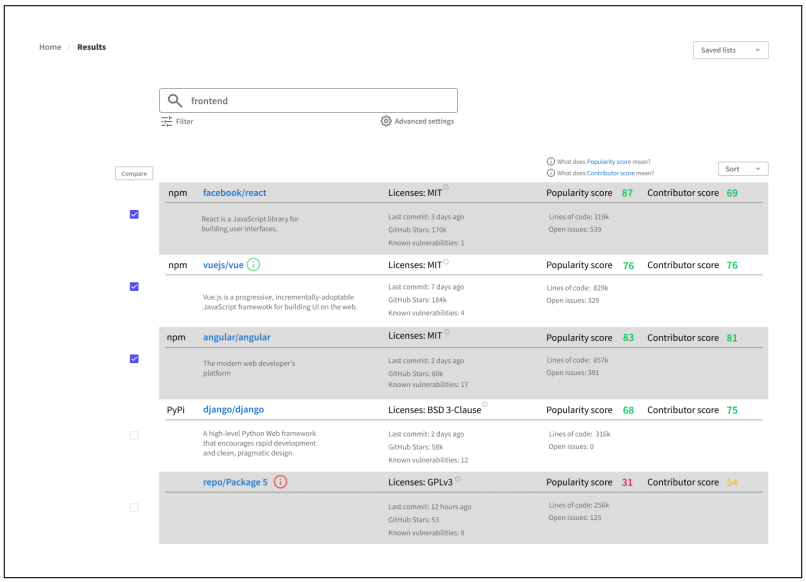


Figure 7.9: Search result with search word ”frontend”.

Search Result

When a search is performed, the user will be linked to the page seen in figure 7.9, the *search results*. This prototype is partly populated with actual data. In this case, with the search work, ”frontend” generates a list of frontend

frameworks. The compare and sort functions are the same as before. The search bar is slightly changed with the ”Advances Settings” underneath. Right above the list, two information texts can be clicked to show a description of the OSH scores. In the list with results, the cells have been modified since prototype 1. In this iteration, the cells are always in the extended version. This is because the user test showed that it was a bit hard to find the ”more” button to extend the view and that it was hard to understand that the package itself could be clicked to get to the package page. The metrics that are shown in each cell have been modified according to the user feedback. If the user hovers over the license type, a description of the license will appear. When clicking a package name, the user will be linked to the corresponding package page.

Package Page - Overview

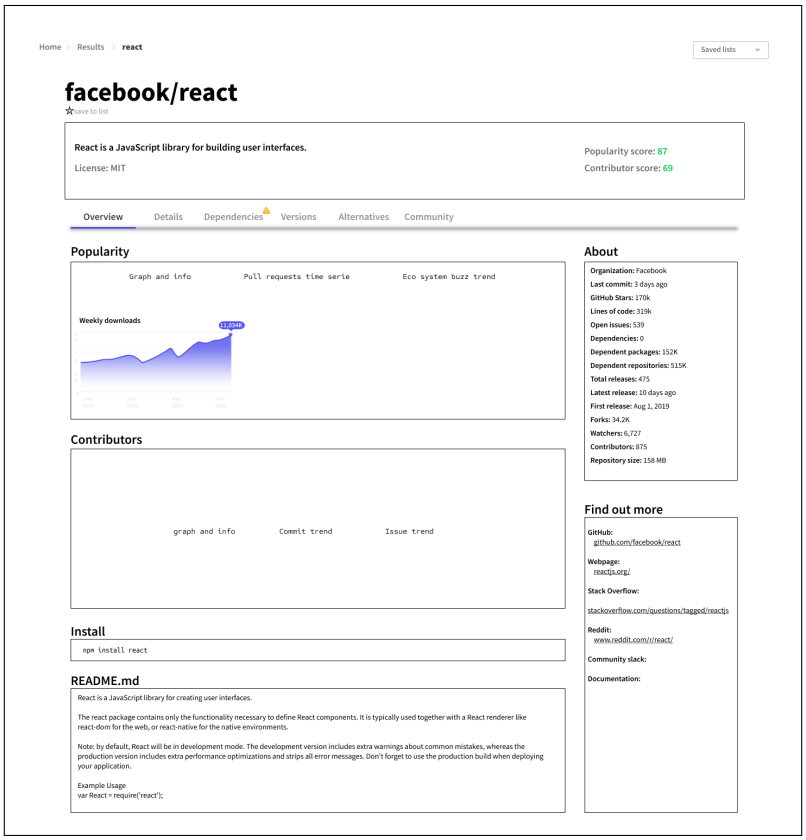


Figure 7.10: Package page for React package, overview.

In figure 7.10, *Package Page - Overview*, is the frontend framework React's package page seen. In this iteration, tabs have been added to the package

page to simplify the navigation of the page and to split the information into different categories. The available tabs are; *Overview, Details, Dependencies, Versions, Alternatives* and *Community*. At the top of the page, there is an overview consisting of a description of the package, *OSH scores*, and license. This box is available in all tabs. If the user hovers over the scores, a description of them will appear. The *overview tab*, which is seen in figure 7.10, contains placeholders for the OSH metrics *Popularity* and *Contributors*. In these, each metric is described by graphs and a time series of its submetrics. At the button of the page, the installation command and README are available like in iteration 1. At the right side of the page, placeholders for raw data and external links are available. The raw data was added since most test participants liked to verify the *OSH scores* with some raw data that they usually look at to build trust for the scores. The box with links was available in prototype 1 as well, but in this prototype, some secondary sources are added.

Package Page - Details

In the *Details* tab, in figure 7.11, the spider plots of the OSH metrics are available, along with the description of the metrics. In the plot are each of the submetrics/practices, that the metric consists of, visualized with its score. Each practice has a description, this was added to build trust and to enable the user to understand the meaning of the scores.

Package Page - Dependencies

Figure 7.12 shows the layout for the *Dependencies* tab. This tab contains a list with dependencies of the current project. The list is split into two parts. One with *critical dependencies*, like packages with critical vulnerabilities or low OSH scores. The other part is viewing *all dependencies*. In the design, the user is able to see the *OSH scores* of the dependency and to click each dependency to visit its package page.

Package Page - Alternatives

The *Alternatives* tab is seen in figure 7.13. This page contains a list of similar packages. Each package in the list has the same information as in the *search results* page are added. The goal with this page is to enable the user to verify that all alternatives have been considered, and to find replacements of packages that are currently in use.

Package Page - Community

The last tab in the package page view, is the *community* tab seen in figure 7.14. This tab contains information about the project's ecosystem. The aim

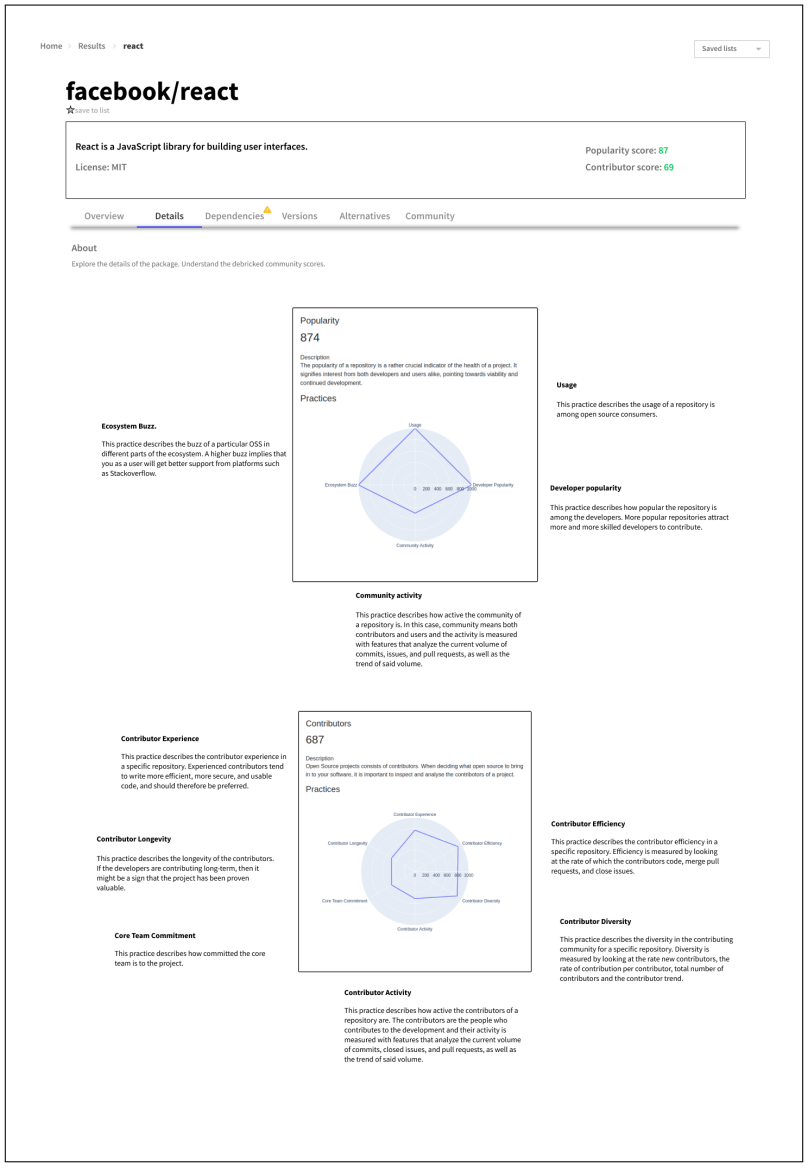


Figure 7.11: Detail view for React package.

of the page is to find out what other companies that use this package, how many times are the package mentioned in different forums or to find links to external pages.

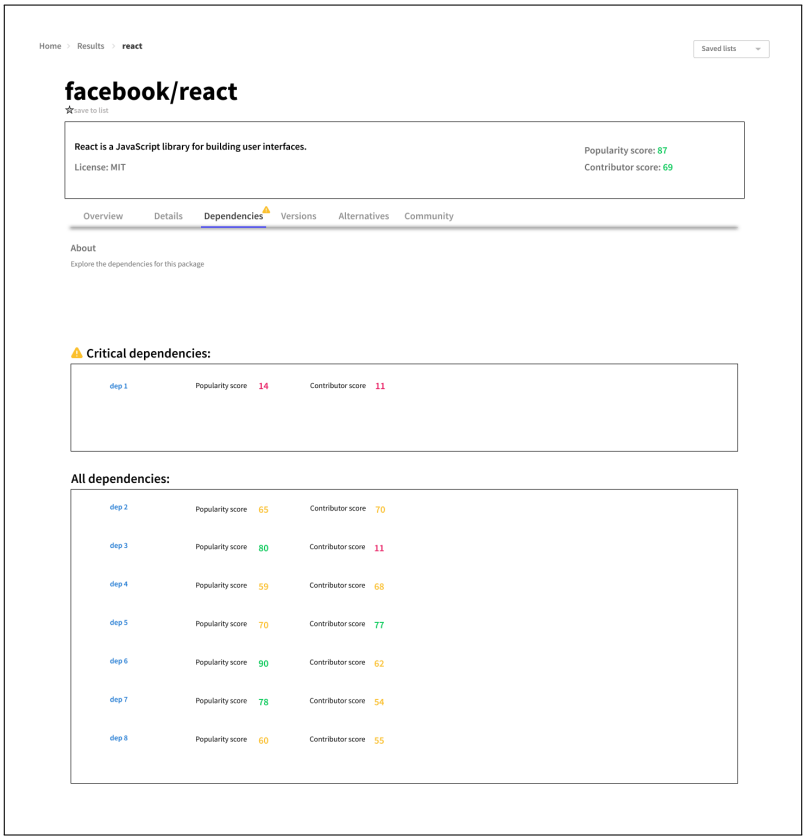


Figure 7.12: Dependency view, dependencies of React.

Compare Page

In figure 7.15, the updated view of the compare page is viewed. In this prototype, the comparison is performed between the two frontend frameworks React and Vue. In this iteration, the user is enabled to perform a graphical comparison with overlayed spider plots for the *Popularity* and *Contributor* metrics, and different submetric plots, e.g. *Weekly Downloads*. Below the graph section, the user can compare raw data side by side for each package.

List Page

The list view, seen in figure 7.16, has the same layout and functionality as in iteration 1.

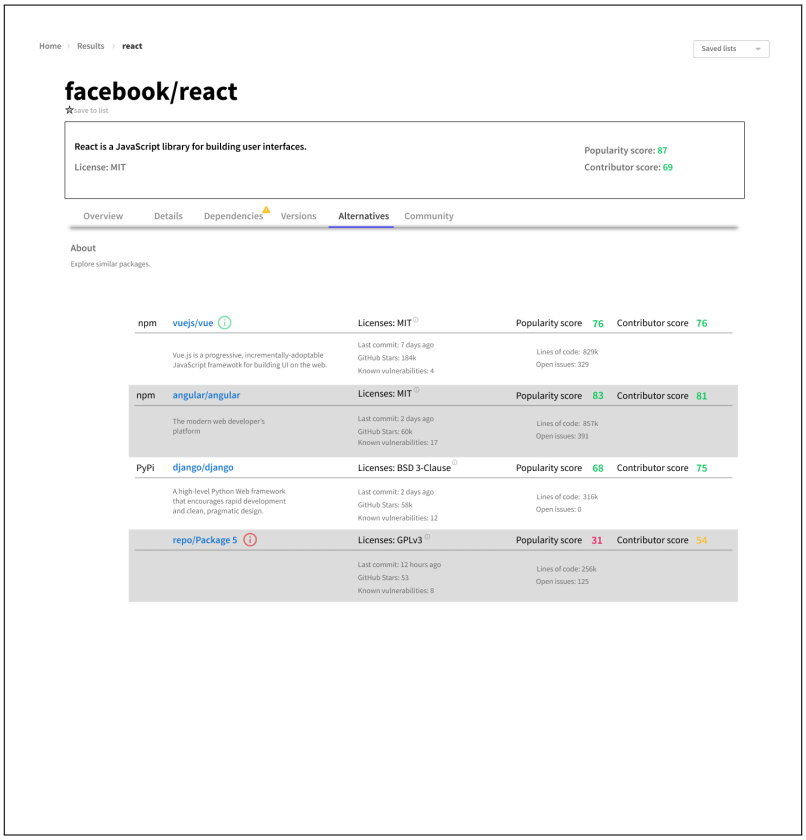


Figure 7.13: Similar packages to React.

7.2.2 Prototype Testing

Search Result

There was a lot of information in the results page, and the participants had mixed opinions of what was relevant for their evaluations. For example, P8 stated that the number of GitHub stars for a package was the most important metric, while P1 said that they are not that interesting for them. Overall, having a mix of raw data and calculated scores was appreciated. P1 stated that "Some raw data is nice to have to be able to verify the scores and create your own perception of the package". Four out of the six participants in this iteration mentioned appreciating at least parts of the raw data presented.

There were several mentions of a want for more informative metrics, other than just the plain numbers presented in this iteration. P1 and P7 were interested in viewing trends, for example if the number of open issues are increasing

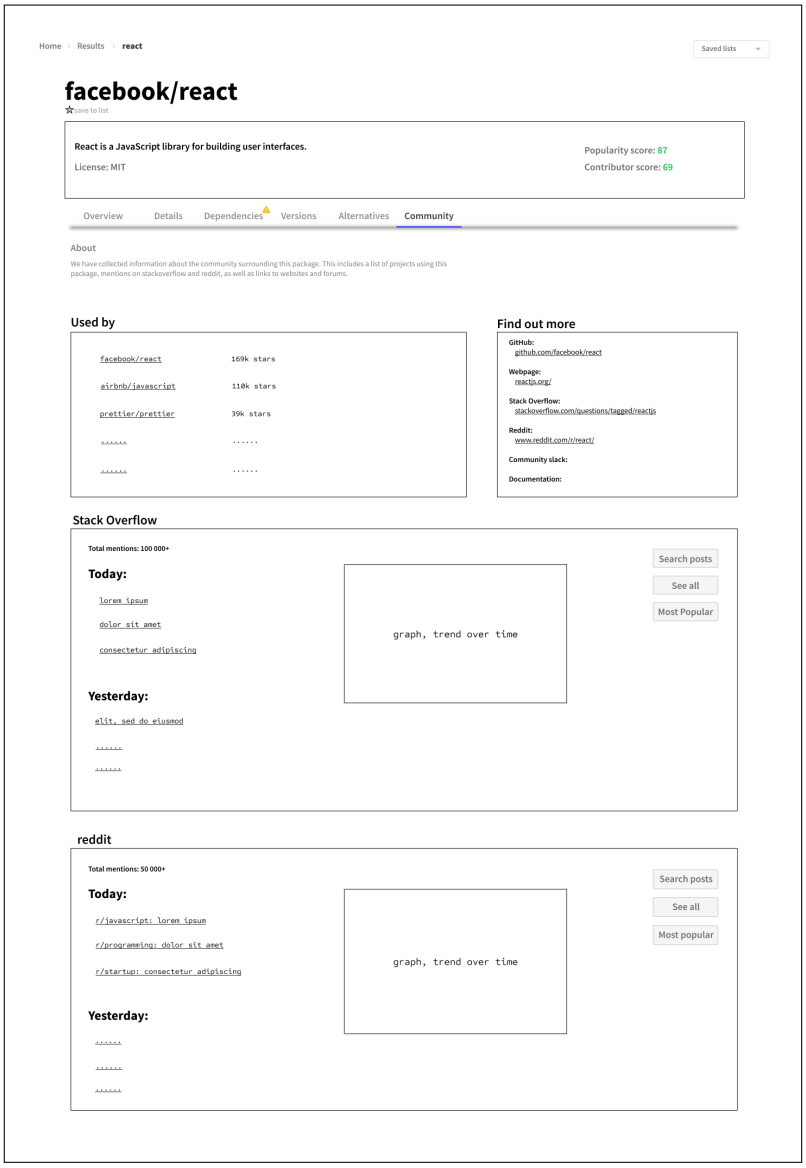


Figure 7.14: Exploring the React community.

or decreasing. P2 and P7 wanted to be able to get more information about vulnerabilities, for example exploring the details of known vulnerabilities for a package. P1 on the other hand did not care about known vulnerabilities for a package, as they had not had any issues with that in the past.

Three participants explicitly stated that the scores from the Open Source Health model, *Popularity* and *Contributors*, could be useful for their processes.

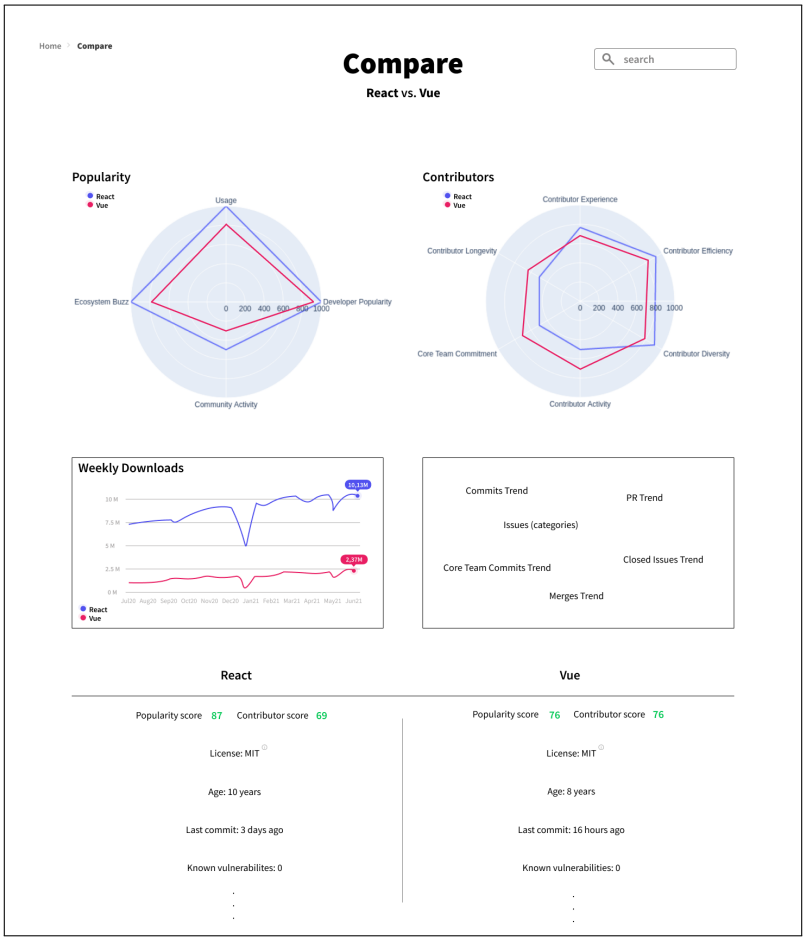


Figure 7.15: Compare functionality, comparison between React and Vue.

P2 mainly saw use in using those score to make a first screening for interesting package candidates, and then going in-depth in other parts of the tool to make the final decisions. P1 also mentioned that in general, the information on this page is enough to sift packages, but they need more information to make final decisions.

Package Page - Overview

Three participants, asked for more information about the OSH-model scores at this point. One wondered what the maximum score is, and all of them wanted a better understanding of the model and how to interpret the scores.

Once again, there were a lot of comments about the choice of what data to present. Four out of the six participants talked about the importance of the

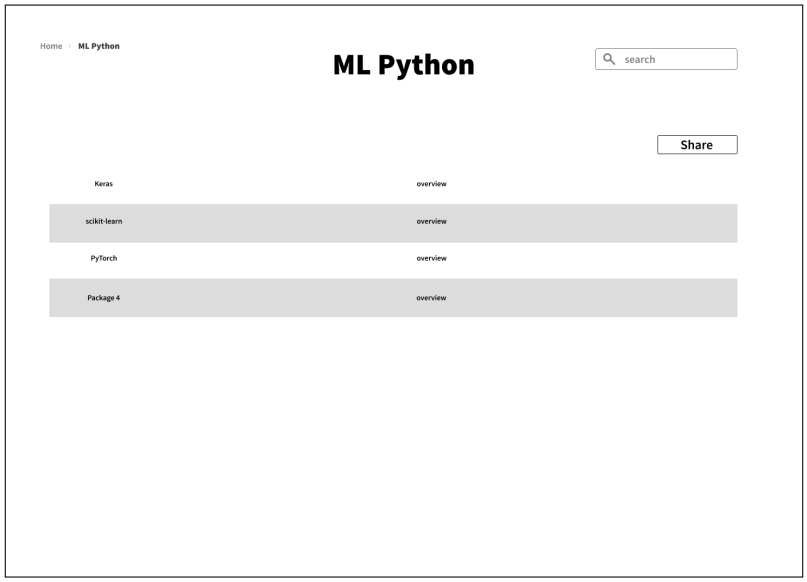


Figure 7.16: List with saved machine learning packages in Python.

metric ”Weekly downloads”. When asked if any of the presented data was unnecessary for them, both P2 and P5 replied that all the information in this view is relevant, maybe with the exception of a hyperlink to reddit.com. Two other participants considered the number of issues to not be very informative in itself. They both mentioned that the number of issues is not as relevant as the trend or types of issues. Once again, there was interest in viewing trends, or time-series, of the data. P1, P2, and P8 all mentioned that this would be more informative for them.

Package Page - Details

This page was overwhelming and contained too much information for most of the participants. Three of them said so explicitly and one navigated away quickly without giving any comments. P8’s first reaction was ”Woah. This seems complicated”. They suggested to have bigger text and to only show the information boxes when hovering on the plot to make it cleaner and more readable.

There was a lot to take in, and a lot of new concepts in this page. Several participants appreciated a lot of the information here. P7 said that they have not looked at OSS health in this way before. P2 said that the main advantage of this page would probably be to pitch packages to other people, by using the visual aid here. P6 thought the metrics were really cool. They thought that some of them were intuitive to understand what they were calculated on but

others, like ecosystem buzz, were more unclear.

Two of the participants reacted to the scores shown. They both expected the package React, which is a really big platform and community would have higher scores. One of them, P6, thought that is could be because it is hard to evaluate community activity purely based on GitHub (which is the OSH-models main data source). That community is more active in other forums.

Package Page - Dependencies

All participants except for one stated that this was a good and important page. P5 said that it was a good way to explore dependencies of a package, which might be overlooked in the selection process today. P2 thought that the warning for critical dependencies would be very useful, and that it could be even more visible to make the user aware of it. P1 and P5 said that showing just the popularity and contributor scores was good since users would eventually be familiar with those score, and that is was enough information to get a brief overview.

P7 wondered about how to show the full chain of dependencies, since there are often layers of them. They also said that it is hard to do something about dependencies since it is too complex, and the further you go in the layers the harder it is.

Package Page - Alternatives

Five out of the six participants had a positive to very positive reaction to this page. P5 said that it might be the best feature of the tool. Usually they just want to choose the most popular and maintained option, which is easy to find and compare in this view. Three other participants said that they would probably use this in practice as well. They saw themselves using it to scan the possibilities in the area. P2 would like to see even more information, such as see the features of the packages and trends of what alternatives are becoming more or less popular.

P6 and P8 were positive to the idea, but thought it could be hard to implement in practice.

Package Page - Community

Three participants liked this page, and one didn't really understand the aim of it. P1 thought that this had good information and interesting links that not everyone is thinking about. It's not the first thing they look for, so it's good that its in its own tab so that it doesn't disturb the flow. P2 would like to have this kind of feature today, as it had good links and could be a good indication of what kinds of problems are common in practice. P7 liked the

idea of showing if the package is used by a company. P5 didn't see the point, and expected to see more raw data describing the activity on forums.

Compare Page

Once again, three users liked this view and one thought it was unnecessary. P1 thought that it didn't provide much more value when you can compare pages directly in the search results and can get a lot of information in the package pages. P2 thought it would be very helpful and said that most of the time it is enough to compare quite simple metrics. They would like a description of the packages in this view as well. P8 would probably not use this every time but could see the value of it, especially when evaluating libraries for bigger imports.

List Page

Only three participants had time to view the list. P1 wouldn't work with lists, and would rather just use something like bookmarks in their web browser. P2 would like it to work like a watch list, to keep track of libraries for some time before making decisions. P7 thought is was a nice feature.

The OSH Model Page

Only two participants had time to visit this view. Both of them, P2 and P8, liked it. They liked having a description of the model for interested users, and thought it would help them understand and maybe trust the model more.

Advanced Settings

This functionality was explained poorly in the prototype. Two participants entered this view before getting any context about the OSH-model in other parts of the prototype and had no way of getting information about it here.

Three participants thought that being able to manually set the weights for how the OSH-model scores are calculated would be a nice feature, but none of them could see themselves using it. They thought it would take too much time, and require too much understanding of the model for it to be worth it for them as users.

Policy

All five participants that had time to discuss this liked the concept of being able to set a policy for what packages are acceptable to import. P1, P7 and P8 especially liked being able to set a policy licenses. P2, P5 and P8 thought it could be valuable to set policies for the popularity and Contributor scores. P5 thought it could be hard to know what good limits would be when starting to use the tool though.

P1 thought that it would be good to not have to teach everyone at the company about good and bad OSS, but has not had a problem with people choosing bad OSS yet. P2 would like the tool to still show packages that are not compliant with a set policy, so that users can still find all packages. P7 would like advanced policies, such as not allowing developers to use Python to open files.

General Feedback

The tabular structure of the package pages had positive responses. Only one participant had trouble finding how to advance to new pages, and they found their way on their own eventually. P1 specifically mentioned the benefit of having pieces of information that they didn't consider important in a tab that they can choose whether they want to navigate to or not.

In a sense, there were two different groups that envisioned different uses of this tool. P4, P5 and P7 talked about the benefit and competitive advantage of having a great search functionality, for example being able to search for the functionality of packages. P2, P6 and P8 on the other hand saw their main usage of the tool probably being if being linked from Google when searching for packages there. P6 thinks it is extremely hard to compete with Google for a tool to discover new packages.

Two participants mentioned that they it would be really nice to have this kind of evaluation integrated in another tool. P6 said that it would be interesting to see this information on a projects GitHub page, but that they don't see themselves going to another website just to see it. P8 wanted an integration with Composer, a package manager for PHP. They said that their main problem is having too many packages, and they could see this information helping them remove bad packages.

P2 would in general appreciate more recommendations from the tool, as it is hard to know what to do with all the information. P8 would also like to receive some sort of opinion for how many security vulnerabilities are normal, and when there are too many, as just a number is pretty meaningless for them. Otherwise, the participants in general liked having a mix of opinions from the tool (the popularity and contributor scores) and raw data. Both P5 and P6 mentioned that they would build an own view of how to interpret and trust those scores by seeing how they correlate with their own opinions based on the raw data.

7.2.3 Hypotheses and Design Adjustments

H2.1
Hypothesis: Transparency in how things are calculated is important.
Test: Page describing the OSH-Model. Info boxes and pop-ups with information. Possibility to see and update model weights. Different levels of information in different tabs and pages, the user do not have to view all detailed information.
Result: Good support, but a lot of subjects have trouble finding and contextualizing that information. Some participants like the idea of being able to set their own weights, but don't see themselves using that functionality in practice.
Design adjustments: Make the information even more visible and understandable. Clarify the connection between the underlying model and graphical user interface (GUI).

H2.2
Hypothesis: The community health of an open source project is an important factor. Users can trust numerical values describing this factor.
Test: Popularity & Contributor scores.
Result: Mixed results. Nobody trusts this blindly, everyone want to know what the scores are based on. Might build trust with time.
Design adjustments: Clarify the model description and how the scores are calculated. Make the descriptions available in all parts of workflow.

H2.3
Hypothesis: The user do not want to go between to many pages, the process should be relatively short. By streamlining the information flow and by using a page layout using tabs, inspired by GitHub that all users are familiar with, people will have an easy time navigating the tool.
Test: Add information to the pages in drop-downs. Show all relevant information at the same page. Enable the compare page and make packages comparable directly in the search result table.
Result: No participant commented on these aspect or seemed confused by the information flow. We interpret this as good support for this hypothesis.
Design adjustments: -

H2.4
Hypothesis: Policy and contextualizing packages for organizations will improve the business value.
Test: Enable organizations to set policies for which packages they allow. Show if a package is already used in the organization. *Result:* Good support. Important not to interrupt normal workflow or add bureaucracy to the process.

Design adjustments: Make this information more visible.

H2.5
Hypothesis: There is a value in gathering info data from secondary sources, such as forums, and showing what other organizations are using a package.
Test: Community tab.
Result: Design adjustments: Keep only links to primary sources, make more pronounced in design.

H2.6
Hypothesis: Being able to explore similar packages is appreciated.
Test: Alternatives tab.
Result: Overall good support, P7 didn't see value in it.
Design adjustments: Describe the how the alternative packages are computed and hoe it can be used.

7.2.4 Main Takeaways

The design using tabs was intuitive to use for all users. All of the package pages and most of the data shown seems to be valuable to at least some users. The most important feedback on prototype 2 is that the description of the OSH model needs to be easily accessible in every part of the tool. The model description could be less subjective and the connection to the raw data could be clearer.

There is a lot of interest in ways to present and contextualize the data better. Showing trends of how metrics have changed over time and contextualizing the issues of packages were requested a lot.

7.3 Iteration 3

In the last iteration of the wire-frame prototype, focus have been to finalize tested concepts to deliver a complete design suggestion and *Nielsen's 10 usability heuristics* described in the section 3.1.1. All heuristics can be found in Appendix A. Since prototype 3 is not a full working prototype, all heuristics won't be considered.

7.3.1 Prototype 3

In this section, the main design and functionality of the third and last prototype are described. This prototype is built on prototype 2, and several pages have not changed and are therefore not shown in this section. The focus will be to discuss the changes. The full prototype can be seen in Appendix D.3.

Search Result

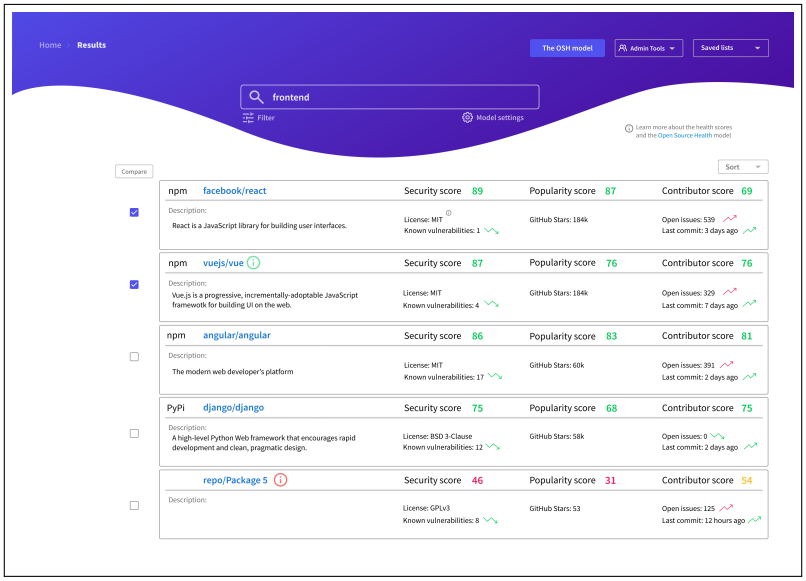


Figure 7.17: Search result with search word ”frontend”.

Figure 7.17 shows the *search results* from prototype 3. The list of search results has been cleaned up, and some metrics have been removed. A trend indicating arrow has been added to some of the metrics after receiving user feedback about it. At the initiative of users, the *security score* is added to the list. On the top of all pages, an *OSH model button* has been added. This links to the *OSH model description page*, which is the same as in prototype 2.

Package Page - Overview

The *package page* of the frontend framework React in prototype 3 is seen in figure 7.18. In this prototype is the spider plots from the details tab in prototype 2 are moved to the *overview page*. This is because the spider plots give a better overview than specific plots and time series, as in prototype 2. The *installation command* is moved to the top of the page to make the process quicker if the user wants to install the package directly.

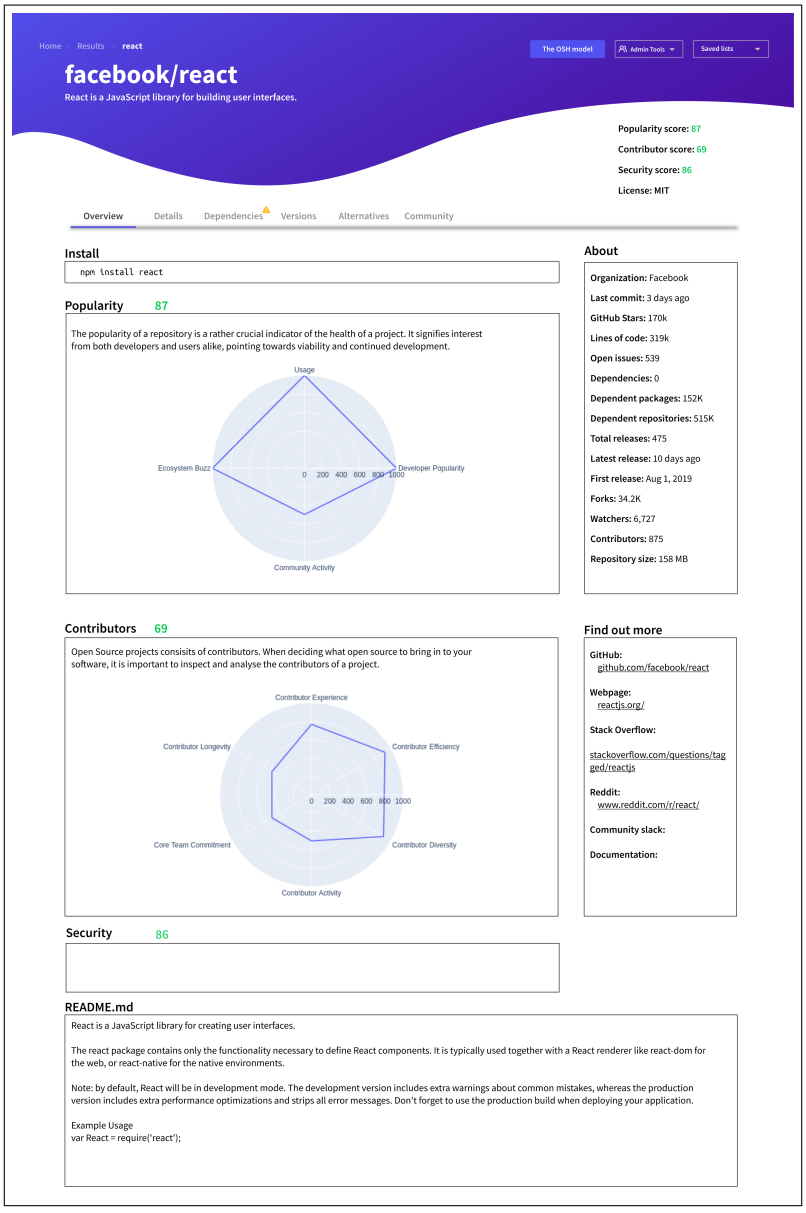


Figure 7.18: Package page for React package, overview.

Package Page - Details

As seen in figure 7.19, the plots from the *overview* tab in prototype 2 have been moved to the *details* tab. The plots are divided into three categories; *popularity*, *activity*, and *contributors*. In the *popularity* category, the user can explore weekly downloads of the package and the number of stars on GitHub.

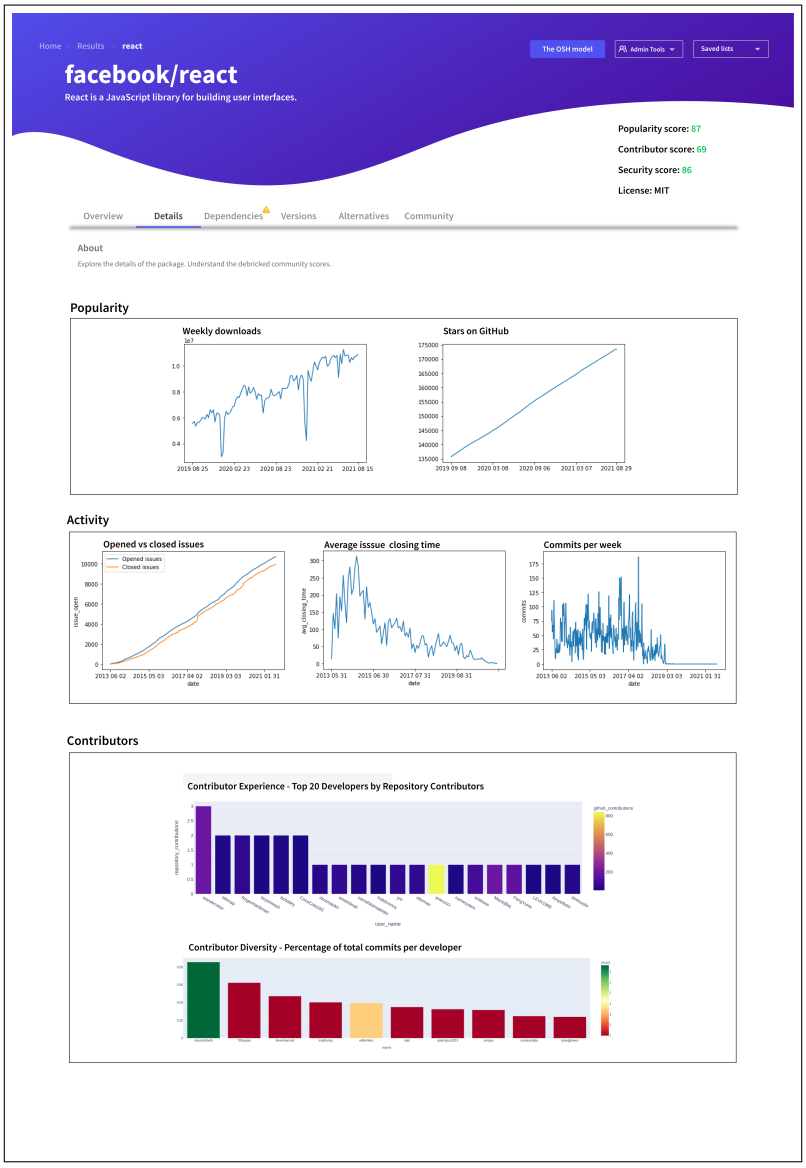


Figure 7.19: Detail view for React package.

To evaluate the *activity* in the community, the user can look at the number of *opened issues* in relation to *closed issues*, the *average closing time for issues*, and the *number of commits per week*. In the *contributor* category, diagrams of *contributor experience* and *diversity* is shown.

7.3.2 Heuristic Evaluation

In this section the feedback from the UX-designer at Debricked presented. The focus in this feedback are the *usability heuristics* [38].

Visibility of System Status

The *visibility of system status* aims to give the user feedback and information about what is going on while interacting with the system to help the user understand the product and build trust.

One example of this heuristic in the prototype is the breadcrumbs that show the user where they are in the tool. Another example is the tab structure on the *package page*. There is no doubt about which tab is currently viewed with the greyed-out tabs and purple underline.

To improve the *visibility of the system status*, a description of the current search and the model could be added to the *search results page*. When editing the company policy, there is no button for save or apply, like in the filter function. This would also improve the *visibility of system status*.

Match Between System and the Real World

The language used in the design should match the language that the user usually uses to make the design more intuitive.

This heuristic was not commented on during the heuristic evaluation. There was no problem understanding the functionality, and therefore this heuristic is assumed to be fulfilled. Note that the evaluation was performed by someone with knowledge about the OSH project.

Consistency and Standard

This heuristic aims to make sure that the same words are used for the same things to meet the customers' expectations.

There are good examples of this heuristic in the prototype, but it can be improved in several cases. For example, on the *package page - Details*, a box called "activity" is placed, which has not been mentioned together with the OSH metrics before. By changing the layout of or removing the "activity" box, the consistency heuristic will improve. The "About" text under the tabs on the *package page* is good and should be implemented under the *Alternatives* tab to keep consistency. Links are usually marked in blue and underlined. The package name is marked in blue on the *search results page*, which makes it easy to find and understand that it is clickable. This blue color could be adapted to all links, for example, the links in the box "Find Out More" on

the *package page*, to fulfill the consistency.

Another example of a standard is the color-coding of OSH scores. Green indicates good, yellow; ok, and red; bad. This helps the user interpret the scores, and it follows the same pattern for all pages in the prototype. The same color coding is used for the trends of specific metrics on the *search result page*.

The red and green info symbol in the table on the search result page is used inconsistently. One is connected to the policy functionality, and one is just informing the user that the package is already in use in the company.

Recognition Rather than Recall

In this heuristics, the evaluator looks if all relevant information is visible and easy to retrieve in all design parts. The user should not have to remember the information or how something should be done.

On all pages of the prototype, there is a link to the OSH model description. This is a good way to give the user access to relevant information. The user does not need to remember the OSH scores between the different pages since they are visible on the *search results page* and in all tabs under the *package page*. The scores on the *package pages* are highlighted, which makes the user aware of them.

To improve this heuristic, it might be good with some additional information on each page with a short description of the aim or some examples of usage. For example, when entering the tool, it would be nice with a description of the tool, how the model works, and some examples of usage. To help the user understand the *Alternatives* tab, it would be nice to add some explanations about which packages are shown and in what order, together with some recommendations. The model settings shown under the search bar on the *start page* and on the search result page might be easier to understand with another name describing the aim of the functionality in a better way. This is used to set the model weights, but it is not obvious how this is done. Some documentation or examples of usage would have been nice.

Flexibility and Efficiency of Use

Flexibility and efficiency of use aim to fit the design to all levels of experience. This can be done by shortcuts that might be hidden from inexperienced users and personalized functionality.

There is a lot of information available, but to structure, the user can cater to the amount of information needed. The *OSH model page* is perfect for users that want to go deep into the model. In the *Dependencies* tab, it is nice that

there is a highlighted warning on the actual tab that there are some critical dependencies. It is nice that the critical dependencies are shown in a special list, but these should also be listed in "all dependencies". The *flexibility and efficiency of use* heuristic might be improved with a filter function instead where the user can filter on critical dependencies.

Aesthetic and Minimalist Design

The more information in the design, the more the relative visibility will decrease. The design should be focused on the essentials, and irrelevant information should not be visible.

There are several good examples of this heuristic. On most pages, the information is grouped into different boxes and placed in a hierarchical order. This makes it easier to navigate and to get an overview. The tab layout on the *package page* is a good way to divide the information into different categories.

On the *package page* - overview the information in the "About" box can be grouped into different categories to improve this heuristic. Also, all repeated information in the table, like the scores, can be put in a column header on the *search results page*.

It is clear that the compare button and the check boxes on the search result page are related to each other in a clean way. The compare function is nice. It is good with the overlaid plots and the side-by-side comparison. It is good that the color coding is consistent, but this might not be nice for more than three packages at the same time. It is not stated how many boxes you can compare at the same time. The header *Compare* is nice, but the *React vs. Vue* could have been clearer by changing the size or placement.

The modal used for viewing policy is nice. It makes the user focus on this specific information or case. This could be used in several places.

Help and Documentation

A good design does not need any additional documentation, but extra documentation can be necessary to help the users understand the structure and how to gain value from it. The documentation should focus on given tasks and be formulated in a concise way with concrete steps.

The *OSH model page* is really good to help the user understand and find more information about the underlying model. It could have been nice to divide the information into smaller sections, where each section focuses on one topic.

It is nice with tooltips and hovers functionality on the different pages. There is a good amount of information in the boxes, but it would have been nice

with a “learn more” option that links to the documentation. In some places, the user should click to view the information, such as the *package page* scores. This functionality can be hard to find, and it might be better to replace it with a hover functionality.

7.3.3 Main Takeaways

The overall usability is evaluated as being quite good. There is a good flow of information. There is a reasonable amount of data presented, and ordering and grouping the data improves readability. There are no major issues with the usability, but there are several aspects and details that can be improved. Some terms and icons can be more consistent within the tool. Some features like filters, setting model weights and comparing packages could be more descriptive. In general, there could also be more links to documentation in the tool.

8 Deliver

This chapter presents the final result of this thesis - a final lo-fi prototype populated with real example data.

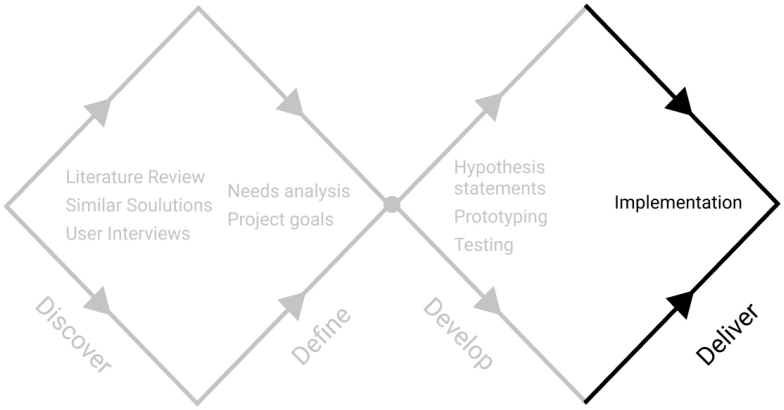


Figure 8.1: The *Deliver* step in the double diamond design process and the activities conducted in this stage.

8.1 Scope

The final prototype was a lo-fi wireframe implemented in figma. It was mainly concerned with showing an example flow for choosing a package. Hence, it is populated with search results for one example search query, and with detailed data for one package. The main goal of this prototype is to showcase how an entire selection process might look - from making a search, making a comparison of the search results and going into details of a specific package. The prototype also contains auxiliary pages containing explanations of the underlying model, as well as a policy functionality for organizations.

8.2 Result

The final version of the prototype is described and partly shown here. Images of all parts of the final version is available in Appendix D.3.

The user flow starts at a relatively clean page with the main focus being a search bar. It contains a prompt saying that it is possible to search both by name and functionality. Clicking on the search button makes the search results appear, shown in figure 8.2. This page contains short and concise information about the packages. Scores from the Open Source Health model are presented, along with some other essential and expected data. Arrows are included to indicate a rough trend, for example if the number of open issues is increasing or decreasing. The graphical profile shown in figure 8.2 is consistent

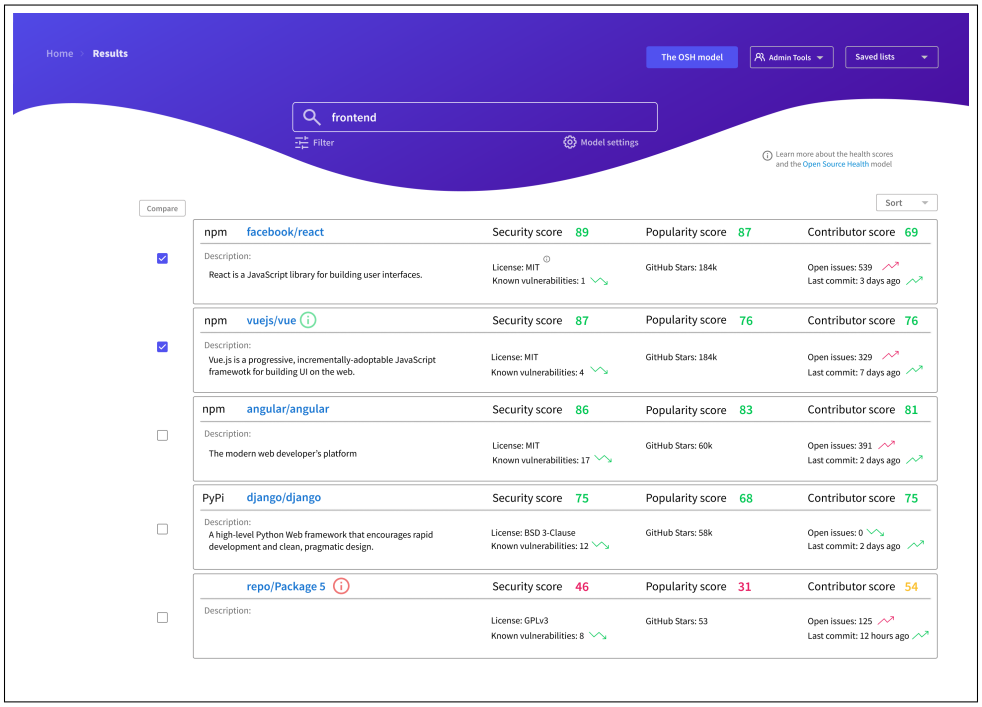


Figure 8.2: Search result with search word ”frontend”.

in the whole prototype. The breadcrumbs and buttons in the very top row are also visible and consistent in all pages. The breadcrumbs allow for visibility of where the user is and navigation to earlier pages. The ”The OSH model” button leads to a page explaining the underlying model in detail, similar to chapter 2.2.1. Clicking on the scores at any point in the prototype shows a more concise qualitative explanation of the relevant part of the model. The

interested user can navigate to the detailed explanation from there as well. The admin tools allow some users in an organization to set policies of what packages are allowed in that organization. Package 5 in figure 8.2 shows how a package that is against a set policy is presented. The package ”vuejs/vue” shows how a package that is already in use within an organization can be highlighted, further contextualizing the results for use in larger corporations and teams.

Clicking on the ”facebook/react” package bring the user to the page shown in figure 8.3. The OSH model, a potential niche for this tool, is highlighted and shown in more detail. It is presented together with a lot of general information and data about the package. The install command, readme and link to GitHub are presented in a way similar to what most users are accustomed to in their normal workflows, providing some familiarity alongside the new OSH-model.

The ”Details” tab is shown in figure 8.4. Here, several plots of raw data are shown. Time-series of total number of downloads and stars on GitHub show how the popularity is changing over time. The total number of open and closed issues are plotted against each other, providing information of how active the development is and if the number of issues are under control. The last two plots illustrate some data regarding the contributors to the package. The top one shows how experienced they are, by showing how many total contributions the most active developers have on their GitHub accounts. The bottom one shows the developer skew - how the number of total commits are distributed along the contributors. This can show if there is a small minority that is responsible for a majority of the work for example.

The ”Dependencies” tab shows information about the packages dependencies - i.e. other packages that this one is built on. They are presented in a list, showing only their names and OSH-scores. The triangle indicates that there is a critical dependency that could be a risk for the package. This could be because of it having open vulnerabilities or having signs of very poor community health for example. In the tab itself, critical dependencies are highlighted clearly.

The ”Versions” tab would show different versions of the package. This was not implemented in this prototype.

The ”Alternatives” tab shows packages that are providing similar solutions as the current package being viewed - providing a way to explore the options when trying to solve a problem. The alternative packages are presented in a list showing the same information as in the search results, as shown in figure 8.2.

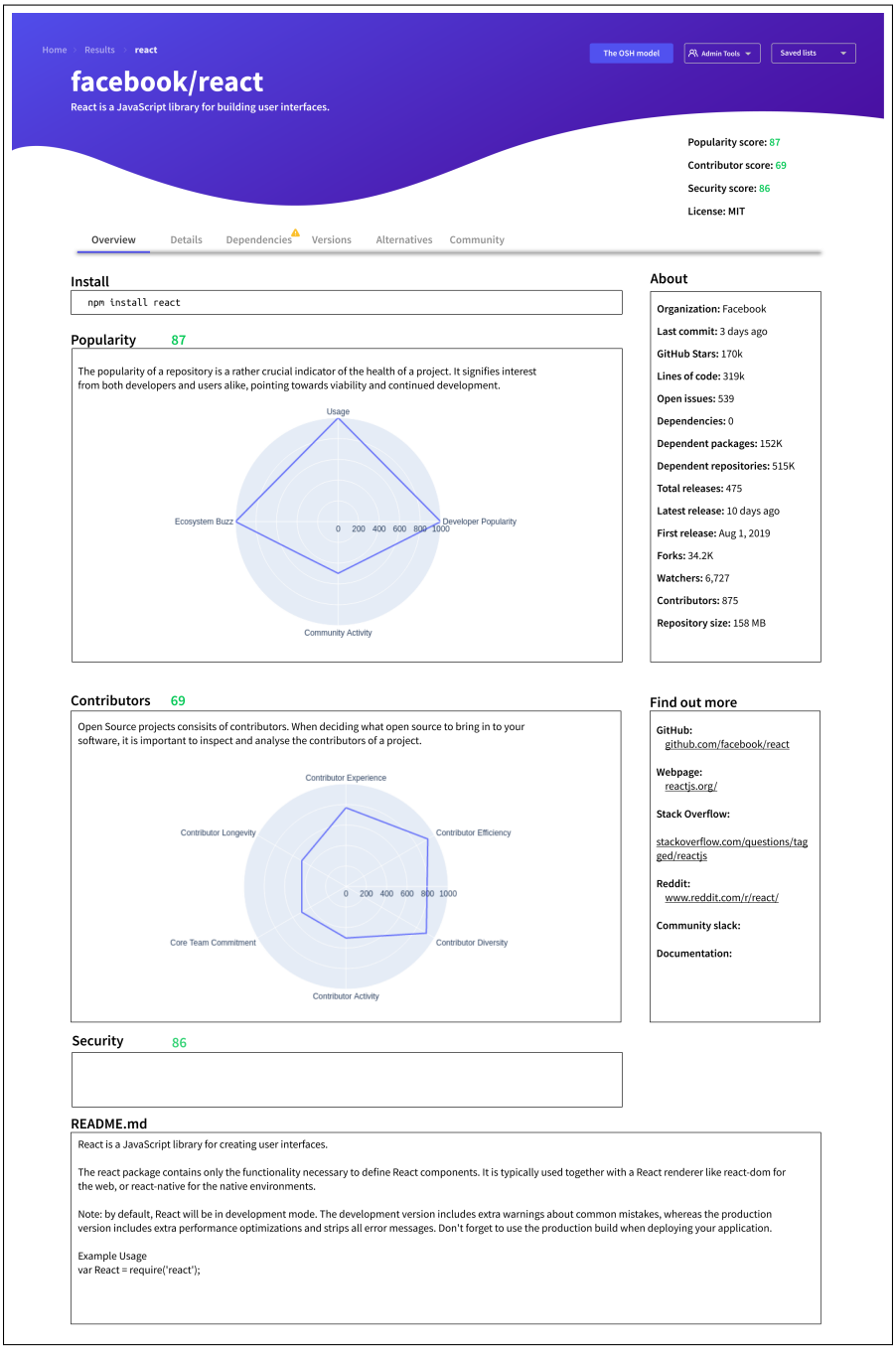


Figure 8.3: Package page for React package.

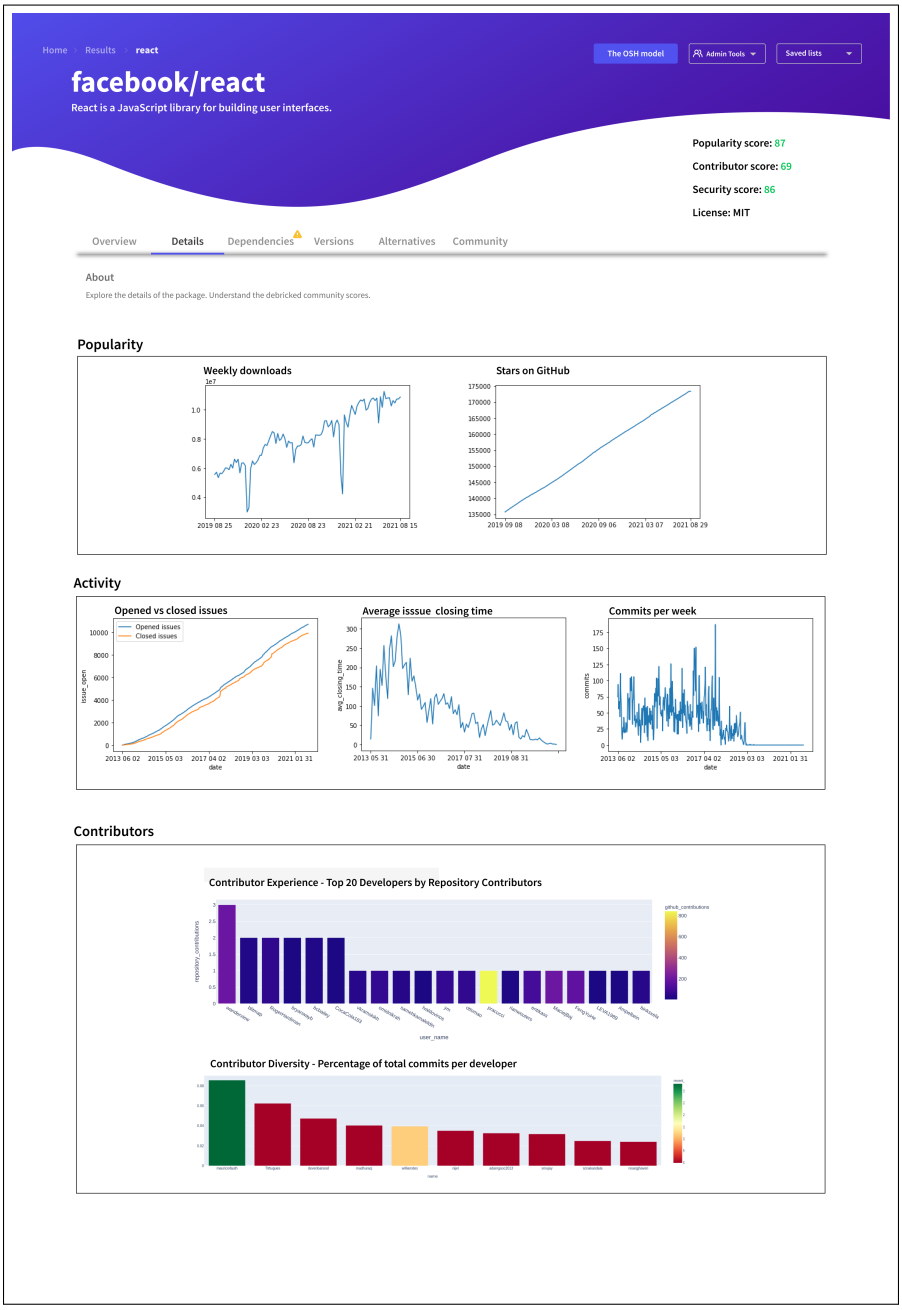


Figure 8.4: Detail view for React package.

The ”Community” tab collects mentions of the package on the forums StackOverflow and reddit. It shows some graphs of mentions over time to indicate

the buzz in the ecosystem. It also shows a list of the most popular repositories using the package. In the case of react, facebook and airbnb are shown to be using it for example. This provides some indication of how adopted the package is in practice, and if major companies use it.

9 Discussion

In this chapter we discuss how the findings can be interpreted and what factors have effected the design process. The selected working methods are discussed in section 9.1 Methodology Discussion.

9.1 Methodology Discussion

9.1.1 Design Process

The double diamond process combined with human-centered design has been the underlying working method during this project. The double diamond's four phases; *Discover, Define, Develop* and *Deliver* and concepts behind human-centered design are described in chapter 4 *Method*. It has worked well to combine these two methods since the *Double Diamond* process describes the overall structure and way of thinking with divergent and convergent phases. In contrast, the human-centered design describes specific activities. It has been helpful during the development to have the specific activities defined in order to find a manageable scope in the day-to-day work. And having the overarching structure of divergent or convergent thinking was helpful in providing a context for the what the overall goal of the activities are. The *Double Diamond* process was also conductive to keeping a balance between exploring different alternatives and still being able to deliver something concrete.

In theory, there is no difference between theory and practice. In practice, there is. In the ideal Human Centered Design process the problem area and intended users are researched extensively before working on ideas for solutions. Because of the limited time of this thesis, the idea generation and prototyping was initiated before interviewing real users. This brings a risk of focusing on solutions for a problem that is not that relevant in practice. It also brings a risk of having confirmation bias in the interviews, since there were already a suggested solution in mind during that process. But we believe that in

the circumstances of this thesis, it was worth it to be able to test something concrete early in the process.

The project was developed during three iterations to get feedback from both users and experts. This seemed like a good balance for the time and scope of this thesis.

9.1.2 Collaboration with Debricked

This project has been in collaboration with the Data Science team and the UX team at Debricked. Both of the teams have contributed with input during the work with the thesis. To not effect the project team's opinion, the literature study and first idea generation were made without input from the Debricked teams.

As mentioned in the section 1.3 *Limitations*, Debricked has, in parallel with this project, developed their version of this tool to be able to release the tool as soon as possible. This fact changed our focus on prototype development and the scope of delivery from a complete working tool in beta. The current delivery has allowed more testing and user evaluation of the prototype. To minimize the influence from Debricked, the project team has not participated in any meetings regarding this prototype, and no design sketches have been shown. Debricked has had complete insight into our work and findings to inspire their work. The most valuable part of this thesis for Debricked has probably been the material from the interviews and prototype testing.

9.1.3 User Interviews

The interviews in this project used a semi-structured interview guide, which is described in chapter 4 *Method*. During the pilot interviews, the semi-structured guide was evaluated and refined. The guide was then used during the first interviews. As more interviews were conducted, the structure tended towards more and more unstructured interviews. This is because we learned more about the area and what we were interested in, but also since the participants working methods and OSS knowledge varied a lot. By not using the semi-structured guide the interviewees had the opportunity to go more into detail in their thinking, and they had more opportunities to reason about their current processes and their thoughts on the problem area in general. This was great for exploratory interviews. One drawback of having more unstructured interviews is that since they are less systematic it is harder to compare and to find patterns in them. There is also a risk of over-analyzing

and over-designing based on few very specific data points. It also takes more time to analyze them. It was possible with the 9 interviews we conducted, but it would not be as feasible to have this approach if more participants would have been interviewed.

The target group for the tool is different kinds of experts within the software field. We as interviewers were very new to the area of open source software when starting this project. Our lack of experience might hinder the understanding of expert interviewees, especially the ones early in the process. During the interviews, key categories and particular concepts were identified, in unity with the *deductive analysis* approach described in section 4.1.3. This way of working may have affected our interpretation of the interview material, as our primary focus was on these key categories.

Most of the interview material in this report has been translated from Swedish to English. Translating information can lead to misinterpretation. The project team has tried to translate as carefully and precisely as possible. Due to the Covid-19 pandemic, all interviews were held remotely. Remote interviews might also lead to misinterpretations, both for the project team and the interviewee. All interviews have been recorded, and the material has been overwatched to minimize the risk of misunderstandings. But there were also benefits with the remote interview setup. For example, it was easier to record the meetings and the interviewees were in their regular working environments. This probably eased the discussion as they physically were in the same context as where they tackle the problems that were being discussed.

9.1.4 Needs Analysis

The needs analysis was performed as a brainstorming session based on the material from the literature review and the user interviews. As the topic of this thesis is a relatively new and unexplored area, especially in Sweden, and there is no similar tools established on the market, it was difficult to determine the needs of the users.

As stated in the literature, it's beneficial to focus on needs rather than solutions. Focusing on solutions can limit the process [24]. Whilst this mindset was intended for this thesis, the thesis is of course highly intertwined with an already existing solution - the Open Source Health model. The needs analysis process was intended to be agnostic to the existing solution, and hopefully validate that there is a market need for it. This could naturally have lead to some confirmation bias in the analysis. This risk was discussed extensively at the start of the project, but biases are of course hard to disregard completely.

9.1.5 Hypothesis Statements

The hypotheses were also formulated during brainstorming sessions. There were several more hypotheses formulated during the *Develop* process, but some were removed in order to get a more focused and coherent report. Having fewer hypotheses overall and narrowing the scope could probably have been beneficial for the project. It could have lead to more focused development and getting more focused feedback.

One of the intentions of using a hypothesis driven design is to be able to test different possible solutions to a problem. But there is of course a risk in getting to attached to ones ideas and instead achieving the opposite effect. The method in general was very helpful though, as it allowed a systematic approach to gathering and interpreting feedback.

Due to the fact that the interviews were conducted in the same session as the prototype testing, the hypotheses for the first iteration had to be formulated before the first interviews. This could possible lead to them influencing the interviews and limiting the interviews to exploring mostly those topics.

9.1.6 Prototyping

At the beginning of the prototype *Develop* phase, the project team generated ideas on their own to avoid being influenced by each other. These ideas were then discussed and merged into one prototype draft. This working method has worked well, but due to time limitations in the projects, the project team has only considered two design drafts. There are many possible design solutions to this problem that the project team has not evaluated.

The prototyping phase has been performed in three iterations. Only low-fidelity prototypes were developed to easily adapt user feedback into the prototype. The low-fidelity approach worked well and made the *Develop* process quick and easy to work with. Using a high-fidelity prototype for the third iteration was considered and almost completely developed. This prototype had a complete working UI and would be connected to the real database for all packages. This would allow a much more interactive testing of the tool. Also, the users would be able to evaluate how the proposed metrics correlate to their view of packages they know well. This hi-fi prototype was never finalized due to time constraints. It could have been finished in time, but we would not have time to perform and analyze tests with real users. And because Debricked worked on a similar beta-version in parallel, there was no reason outside of this thesis to finalize it either. So since the ultimate pro-

totype was also a lo-fi one, and the final results pivoted to testing usability factors instead.

One big difference between prototype 1 and 2 is the viewed data. The project team decided to populate prototype 2 with data from a real example to give the user a context to help them interpret the data and the OSH model. This approach resulted in feedback focused on specific metrics and the OSH model. Another method could have been to keep the placeholders from prototype 1. This could have resulted in feedback more focused on the workflow and page layout. This is a big lesson learned from this project - that what is shown in a prototype matters a lot for what sort of feedback is received. It could probably be beneficial in future projects to test very different prototypes in terms of scope and detail simultaneously in order to get feedback on a wide range of subjects.

9.1.7 Testing

User Testing

The prototype tests during iteration 1 and 2 were performed with potential end users. The users performed the tests after discussing their current procedures and workflow for selecting OSS packages, which may affect their mindset while giving feedback. This arrangement was made due to the limitations in time, but at the same time, get as much data as possible.

It can be hard for test users to give feedback tools they have never seen before, especially ones as complex as the prototypes in this thesis. Some of the participants participated in tests during both iteration 1 and 2. These tests generated a lot of valuable input since the participants where then more familiar with the problem area. Since there are no similar tools established on the market today, this way of working and talking about OSS can be rather new for the participants. This can affect their ability to formulate their feedback. This is especially apparent for the interviewees who stated that they go mostly by gut feeling - as they then aren't that accustomed to verbalize their process and reasoning.

During the tests, the user was told to use the tool to find and evaluate an OSS package to solve a specific problem. In the second iteration of the prototype, this problem was evaluating the front-end JavaScript framework *react*. This was chosen because *react* is a well-known framework and was considered a good example by the project team. Some users stated that it has been hard to think outside the example. Some of the feedback was about react in particular as well. Keeping the example data and placeholders in prototype two may have

resulted in more general feedback.

All user tests were recorded, and the material has been reviewed to minimize the risk of misunderstandings and to analyze the user's behaviors when navigating the tool. The quality of the recorded material was not as good as expected. The video lagged a lot, probably due to overload on the computer making the recordings, making it hard to analyze the user's behaviors when reviewing.

Heuristic Evaluation

The heuristic evaluation aimed to check the usability heuristics in the tool and to find usability problems. This evaluation method has worked well, and it was a good way to test the heuristics. The heuristics could have been a bigger part of the development, now they were only formally evaluated at the very end of the project.

9.1.8 User Group

The target group for this tool is experts in software development and software security. Finding participants for user interviews and prototype tests with a reasonable variation in roles, experience, age, and company size has been quite hard. Since Debricked only has been active for a few years, the customer base is limited. Our network is limited as well.

The limited accessibility within the target group has led to a predominant part of start-ups and smaller companies within the participant group. Commonly, these companies do not have fully developed routines, and they work quickly and flexibly. This may have affected our result as we have been looking for routines and patterns in the selection process. Most companies are based in Sweden, which can affect the results since the degree of maturity within OSS in Sweden is limited, as discussed with Johan Linåker during an online call (2021-07-05).

The variance in the participants' roles is acceptable. They were primarily developers, CTO's and architects. Architects and CTO's are often involved in bigger architectural decisions, while developers are more used to handle smaller imports and help libraries. This might affect how different participants look at the requirements in security and usability.

9.2 Results

9.2.1 Selection process

There was a lack of existing literature exploring the OSS selection processes directly. What was found indicated that it is rare with clearly defined procedures in practice, that it depends on the situation and that it is often up to the common sense of the developer. The results from this work further support these findings.

The selection process seems pretty similar at a high level, throughout all interviews and reviewed literature. Developers start by trying to solve specific problems, and then search the web for solutions. They often look through forums, where StackOverflow was most frequently mentioned by the interviewees in this thesis. The process often involves some sort of informal and unstructured discussion, often during code reviews. Most interviewees emphasized their want for autonomy and a fast process without unnecessary bureaucracy. This is probably influenced by the fact that most of our interviewees worked at start-up companies.

It was mentioned that most participants of a survey [31] had informal selection processes where the common sense of the individual developers was a main deciding factor. Some of the interviewees in this study expressed similar sentiments, and some even said explicitly that in the end they mainly go by gut feeling. The apparent fact that many developers don't have defined or planned processes makes it hard to design a solution fitting everyone, since there is no consistent standard in practice. It could also indicate that there is an unmet need for improved processes, where a tool suggesting a more systematic approach could potentially lead to better practices and more awareness in the OSS-ecosystem.

9.2.2 Metrics

Our results are somewhat similar to what was found in the literature, but there was also quite a bit conflicting data. For example, the most mentioned factors according to the two literature studies [33, 34] were *Cost* and *Support and service*. None of these were mentioned directly by the participants in this study. One probable reason is that our interviews were conducted in the context of choosing only OSS solutions. Cost, and to a lesser extent support, is a more relevant factor when deciding between OSS and proprietary solutions. So in our the context of our participants, cost was probably an irrelevant

aspect. Another reason could be that the participants in this study were mostly developers and in smaller companies. Li and Moreschini et. al.'s survey [32] solely interviewed developers. They also found that no interviewee mentioned economic factors, mirroring our results. Whilst *Support and service* wasn't mentioned explicitly in our interviews, a lot of participants mentioned factors that indicated the availability of support. *Speed of bug resolutions* and *Responsiveness of questions in community* are clear examples of this, and *Activity* and *Popularity* in general are quite clear indicators of a community where support is available.

One aspect that appeared in the literature as well as this study was the fact that people consider a wide array of different metrics, and that people can consider quite different factors. Li and Moreschini et. al. [32] found that 23 developers cumulatively considered 54 different factors, with each individual developer only considering an average of 2.35 different factors. As seen in table 5.2, there is a similar spread in our results. Our interviews indicate that our participants consider similar main factors but can use different specific metrics to evaluate that factor. For example: 5 participants mentioned *Popularity* in this study, but there were also 5 different ways of evaluating it, with only one metric, *Number of contributors* being mentioned by more than one participant. This indicates that there is a lot of variance in what pieces of data developers are requesting in order to evaluate OSS. This means that any attempt to collect this data for a broad audience must face a difficult balance between showing enough data to please a wide audience and not showing an overwhelming amount of data.

9.3 Future Work

As this design project has not delivered a fully developed tool for OSS selection, more research is to be done. We have considered several interesting topics for future work, in addition to the continued development of the prototype, both according to usability and functionality, and eventually implementation of the tool.

One topic discussed during the prototype tests is OSH score and how the relative score differs between different package sizes, categories and programming languages. As the OSH model is implemented now, the score is normalized for each programming language. A future research topic could be to investigate how package size or categories affects the relative score.

Another topic discussed during the prototype tests is how the similar packages

under the tab *Alternatives* are computed. Several participants mentioned that the list of similar packages would probably be the same as the search results. There are several factors that needs to be taken into consideration when computing. Should all components be written in the same programming language? Should the OSH scores be in the same order of magnitude? Should it rely on the package dependencies?

It would be interesting to look at how companies behind the OSS components and the OSS communities could benefit from this tool towards a more healthy community. How could the tool affect the different healthiness factors described by Linåker; *productivity, Robustness*, and *openness* [16].

One aspect that could benefit from further research is the metrics evaluating OSS-packages and communities. The most commonly used metrics in practice seem to be rather simple not that informative - for example number of downloads. Exploring more informative metrics and how to make them adaptable in practice could be beneficial for the whole open source ecosystem.

10 Conclusions

This chapter contains the main takeaways and most important results from this master thesis.

RQ1: What is the selection process for different use cases when adopting new open source software?

All the interviewees give a similar description of their process, which can be summarized in the following steps:

1. Google search, commonly searching by problem or functionality.
2. Reading at forums and blog posts
3. Look at GitHub and homepage
4. Install and test
5. Peer review in a pull request or group discussions within the team. Especially for larger imports.

It was rare to have a formal selection process with defined procedures, steps, and evaluation criteria. In practice, all participants in this study simply made their decisions based on the sensibilities that have been developed from their practical experience.

The main difference in use cases was in the sizes and importance of the packages being selected. For larger architectural decisions like frameworks, there was often some sort of group discussion and collective process. This was still a rather informal process for the participants interviewed in this study. It seems commonplace with an element of discussion in the use case of smaller, less important decisions as well. In those situations, it takes place in the form of a regular code review.

RQ2: Which metrics are important for different use cases when evaluating open source software?

The main factors that participants in this study wanted information about were popularity, activity, quality, and license. There were some differences in how the interviewees found this information and especially in what metrics they used to evaluate them. Many participants were mainly interested in the metrics that they were familiar with, which meant the metrics available on GitHub and package managers. The most commonly mentioned specific metrics were license and when it was last updated.

Many participants indicated an interest in more contextualized metrics, such as the trend of downloads, how fast issues are closed and how a new vulnerability was handled in the past. This could indicate that there is an unmet need for more specific information than what is available today.

RQ3: How can a user interface be designed to facilitate the selection process and to communicate a business value of Open Source Health?

The *Develop* phase of the design process shows one way of designing the user interface. During this project, we have found that one of the most essential things in the UI is to be transparent about the underlying OSH model. If the user does not understand the model or how things are calculated, there is a bigger step to adapt the tool in their current selection process since they then want to verify all numbers themselves.

Another key factor is to allow the users to perform their evaluation in the way that they are accustomed to by showing the data that they are familiar with interpreting. Many will not trust a new unfamiliar evaluation score but can be given the opportunity to find if it correlates with their own evaluation over time.

The main business value provided by the tool is more information for making less risky long-term choices of OSS. An additional way to increase the business value is to provide tools for organizations to contextualize the tool for them specifically. Two such tools are suggested: to provide information about which packages are already in use in the organization, and to provide a policy engine; where the organization can make organizational-wide decisions of what types of OSS are acceptable to import.

References

[1] *Overcoming Top Open Source Risks*. Aug. 18, 2021. URL: https://www.whitesourcesoftware.com/free-developer-tools/blog/overcoming-top-open-source-risks/.

[2] *Responsive Design*. Aug. 12, 2021. URL: https://en.wikipedia.org/wiki/Responsive_web_design.

[3] *Universal Design*. Aug. 12, 2021. URL: https://en.wikipedia.org/wiki/Universal_design.

[4] Frank N. et al. *Report on the 2020 FOSS Contributor Survey*. Tech. rep. The Linux Foundation & The Laboratory for Innovation Science at Harvard, Dec. 2020.

[5] *opensource.com; What is open source?* July 20, 2021. URL: https://opensource.com/resources/what-open-source.

[6] *Usage statistics of web servers*. July 13, 2021. URL: https://w3techs.com/technologies/overview/web_server.

[7] *Standard for Public Code; Glossary*. July 19, 2021. URL: standard.publiccode.net/glossary.html.

[8] *GitHub glossary*. July 20, 2021. URL: https://docs.github.com/en/get-started/quickstart/github-glossary.

[9] Eghbal N. *Roads and bridges; The Unseen labor behind our digital infrastructure*. 2016.

[10] *Snyk.io; Open Source Licenses: Types and Comparison*. July 20, 2021. URL: https://snyk.io/learn/open-source-licenses/.

[11] *National Vulnerability Database; Vulnerabilities*. July 20, 2021. URL: https://nvd.nist.gov/vuln.

[12] Python Software Foundation. *PyPi : The Python Package Index*. Aug. 3, 2021. URL: https://pypi.org.

[13] Inc. npm. *npm*. Aug. 3, 2021. URL: https://npm.js.

[14] Åkerlund M. and Lundstedt R. *Rapport om strategiska vägval för e-arkiv och digital informationshantering*. Tech. rep. AF-2021/0001 1174. Arbetsförmedlingen, May 2021.

[15] *Key takeaways from the OSOR event on the sustainability of open source communities*. July 7, 2020. URL: joinup.ec.europa.eu/collection/open-source-observatory-osor/news/osor-event-sustainability-webinar-takeaways.

[16] *OSOR Event Johan Linaker*. July 19, 2021. URL: joinup.ec.europa.eu/sites/default/files/inline-files/OSOR_Event_Johan_Linaker.pdf.

[17] *Checklist for measuring the health of an open source project*. July 8, 2020. URL: www.redhat.com/en/resources/open-source-project-health-checklist.

[18] CHAOSS Foundation. *CHAOSS Metrics*. Aug. 12, 2021. URL: https://chaoss.community/metrics/.

[19] *debricked.com*. Aug. 9, 2021. URL: https://debricked.com/.

[20] Norman D. and Nielsen J. *The Definition of User Experience*. May 20, 2021. URL: www.nngroup.com/articles/definition-user-experience/.

[21] Nielsen J. "Enhancing the explanatory power of usability heuristics". In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 1994, pp. 152–158.

[22] *The design council; A study of the design process*. June 9, 2020. URL: www.designcouncil.org.uk/sites/default/files/asset/document/ElevenLessons_Design_Council%20(2).pdf.

[23] Norman D. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[24] Ericsson Å. Wikberg Nilsson Å. and Törlind P. *Design; process och metod*. Studentlitteratur, 2016.

[25] Hsiu-Fang H. and Shannon S. E. "Three approaches to qualitative content analysis". In: *Qualitative health research* 15.9 (2005), pp. 1277–1288.

[26] Gothelf J. and Seiden J. *Lean UX*. O'Reilly Media, Inc., 2021.

[27] Dam R. and Siang T. *Stage 4 in the Design Thinking Process: Prototype*. May 25, 2021. URL: www.interaction-design.org/literature/article/stage-4-in-the-design-thinking-process-prototype.

[28] *Nielsen Norman group; How to Conduct a Heuristic Evaluation*. July 1, 2020. URL: www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/.

[29] *About Miro*. May 25, 2020. URL: miro.com/about/.

[30] *Figma (software)*. May 25, 2020. URL: en.wikipedia.org/wiki/Figma_(software).

[31] Jansen F., Jansen S., and Hou F. "TrustSECO: an interview survey into software trust". In: *arXiv preprint arXiv:2101.06138* (2021).

[32] Li X. et al. "Exploring factors and measures to select open source software". In: *arXiv preprint arXiv:2102.09977* (2021).

[33] Lenarduzzi V. et al. “Open Source Software Evaluation, Selection, and Adoption: a Systematic Literature Review”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 437–444. DOI: 10.1109/SEAA51224.2020.00076.

[34] Sánchez V. R. et al. “Open Source Adoption Factors—A Systematic Literature Review”. In: *IEEE Access* 8 (2020), pp. 94594–94609. DOI: 10.1109/ACCESS.2020.2993248.

[35] *About Open Source Insights*. July 22, 2021. URL: https://deps.dev/about.

[36] *About Libraries.io*. July 22, 2021. URL: https://libraries.io/about.

[37] Cruz A. and Duarte A. *npms - About*. Aug. 4, 2021. URL: https://npms.io/about.

[38] *Nielsen Norman group; 10 Usability Heuristics for User Interface Design*. June 16, 2020. URL: www.nngroup.com/articles/ten-usability-heuristics/.

Appendix A

Nielsen's 10 Usability Heuristics

The heuristics' descriptions are based on a 2020 update of the article *10 Usability Heuristics for User Interface Design* by Nielsen [38]. Since the prototypes are low fidelity prototypes and are not fully developed, all heuristics are not considered in this report.

Visibility of system status

The user should always gain appropriate feedback and information about what is going on while interacting with a system. This will help the user to understand the product and build trust.

Match between system and the real world

The language used in the design should match the language that the user usually use. This will make the design more intuitive for the user.

Consistency and standards

Make sure that the same words are used for the same things, be consistent. This will more likely meet the customers' expectations.

Recognition rather than recall

All relevant information should be visible and easy to retrieve in all parts of the design. The user should not have to remember the information or how something should be done. This is due to human's limited short-term memory.

Flexibility and efficiency of use

The design should fit all levels of experience. This can be done by shortcuts, that might be hidden from inexperienced users, and personalised functionality.

Aesthetic and minimalist design

Irrelevant information should not be visible. The more information in the design, the more will their relative visibility decrease. The design should be focused on the essentials.

Help and documentation

A good design does not need any additional documentation, but to help the users to understand the design and how to gain value from it, extra documentation can be necessary. The documentation should focus on given tasks, and be formulated in a concise way with concrete steps.

Appendix B

Similar Solutions

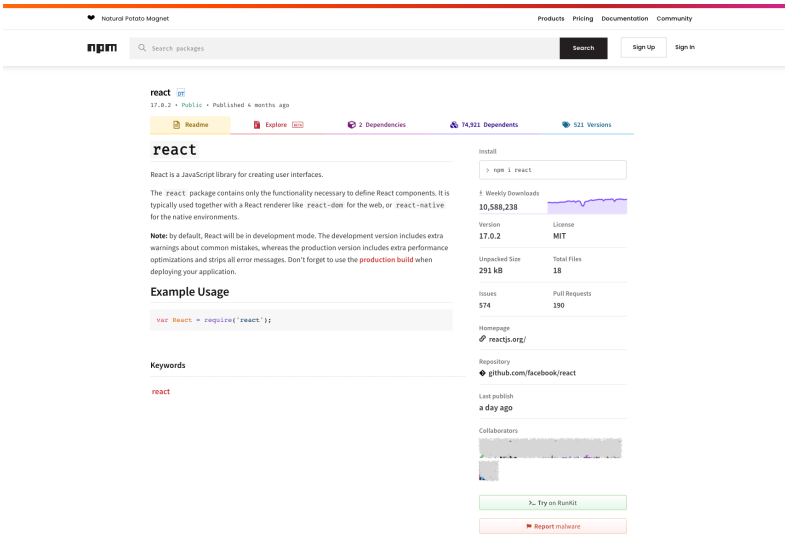


Figure 10.1: Snippet from npmjs, showing the frontend framework react. Identifying information has been blurred.

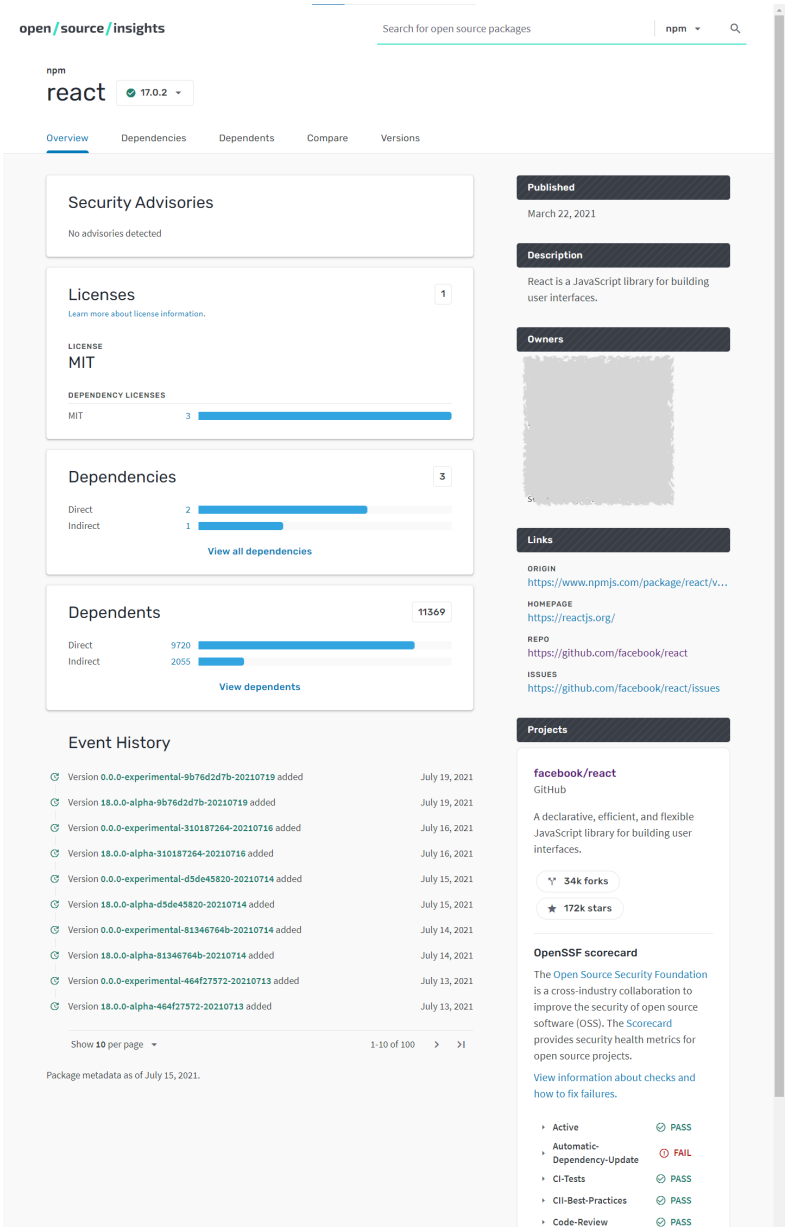


Figure 10.2: Snippet from deps.dev, showing the frontend framework react. Some identifying information has been blurred out.

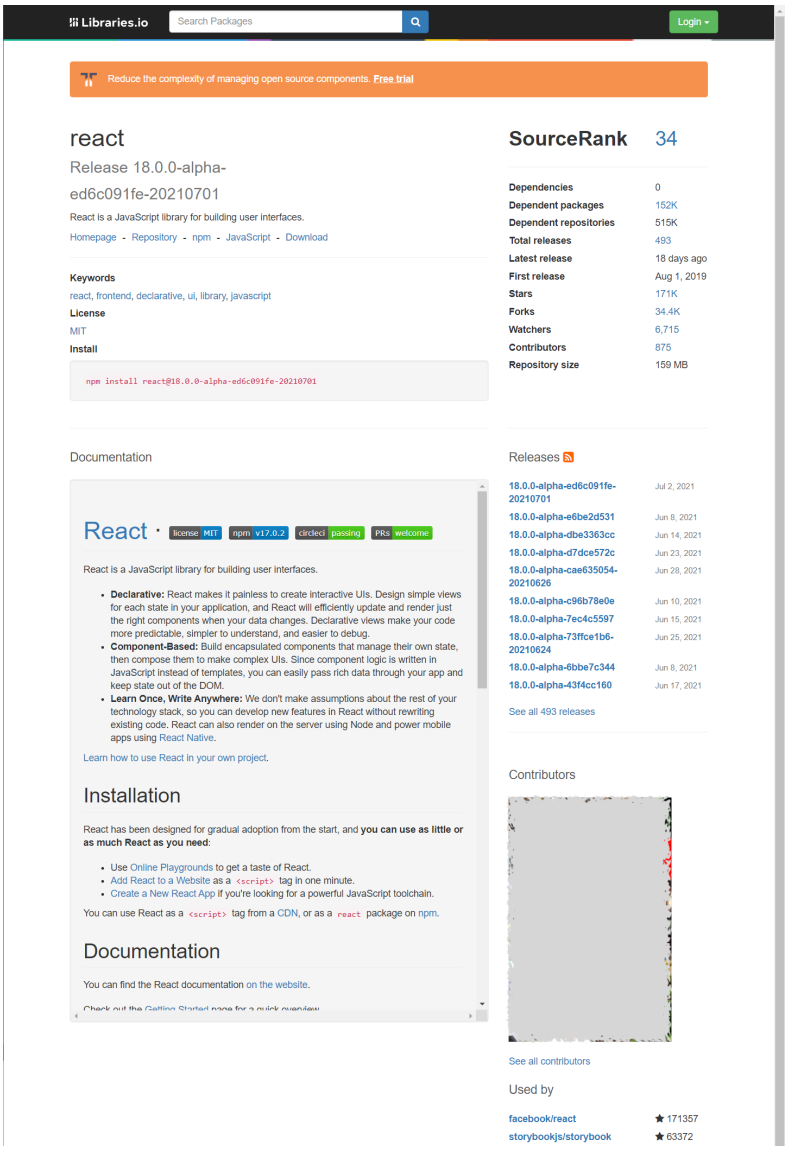


Figure 10.3: Snippet from Libraries.io, showing the frontend framework react. Some identifying information has been blurred.

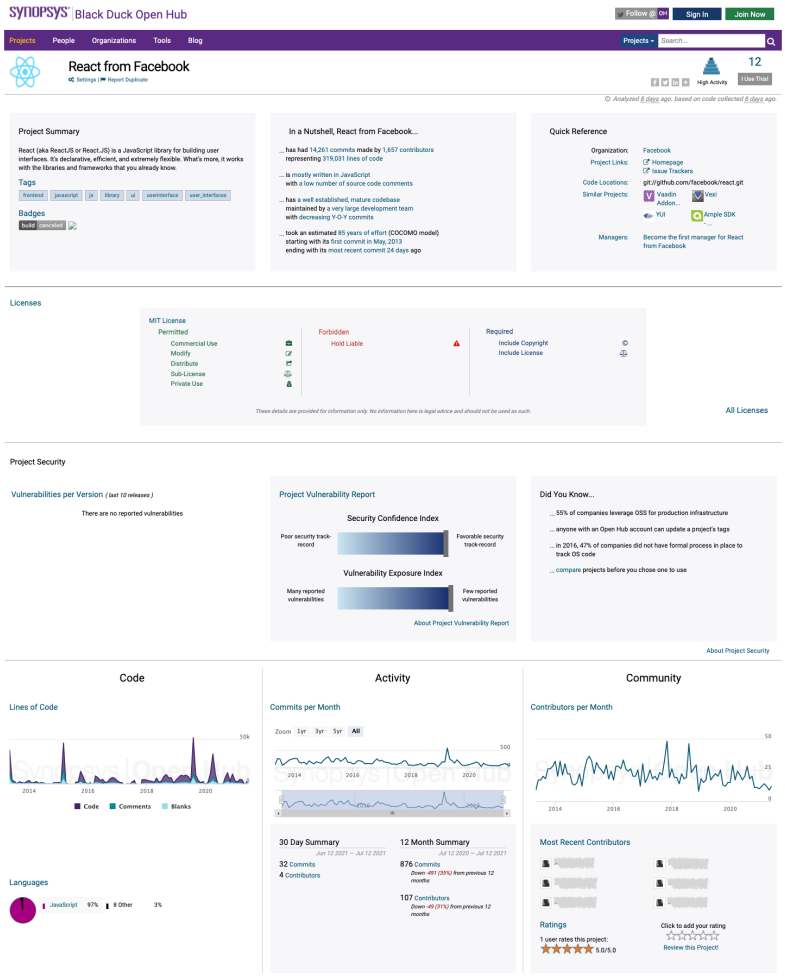


Figure 10.4: Snippet from Open Hub, showing the frontend framework react.

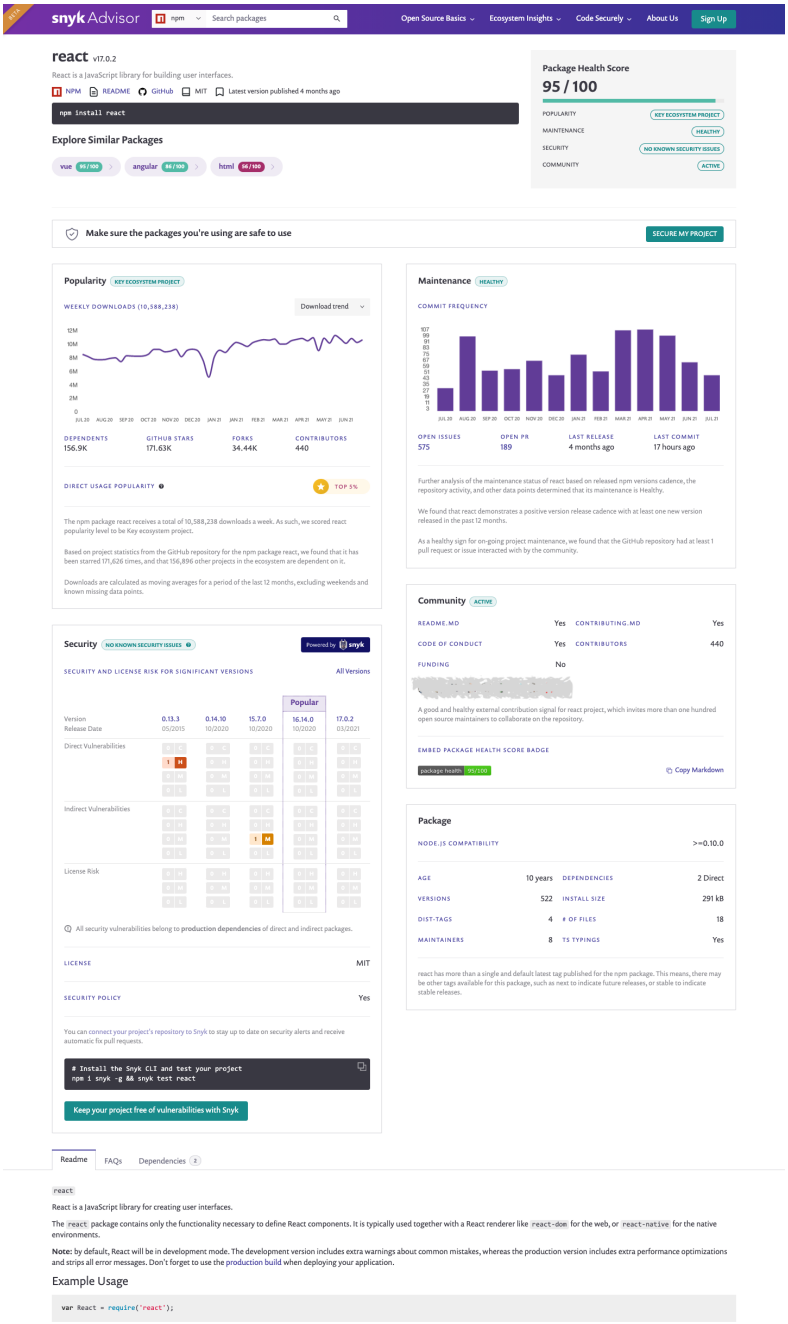


Figure 10.5: Snippet from Snyk Advisor, showing the frontend framework react. Pictures of contributors have been blurred.

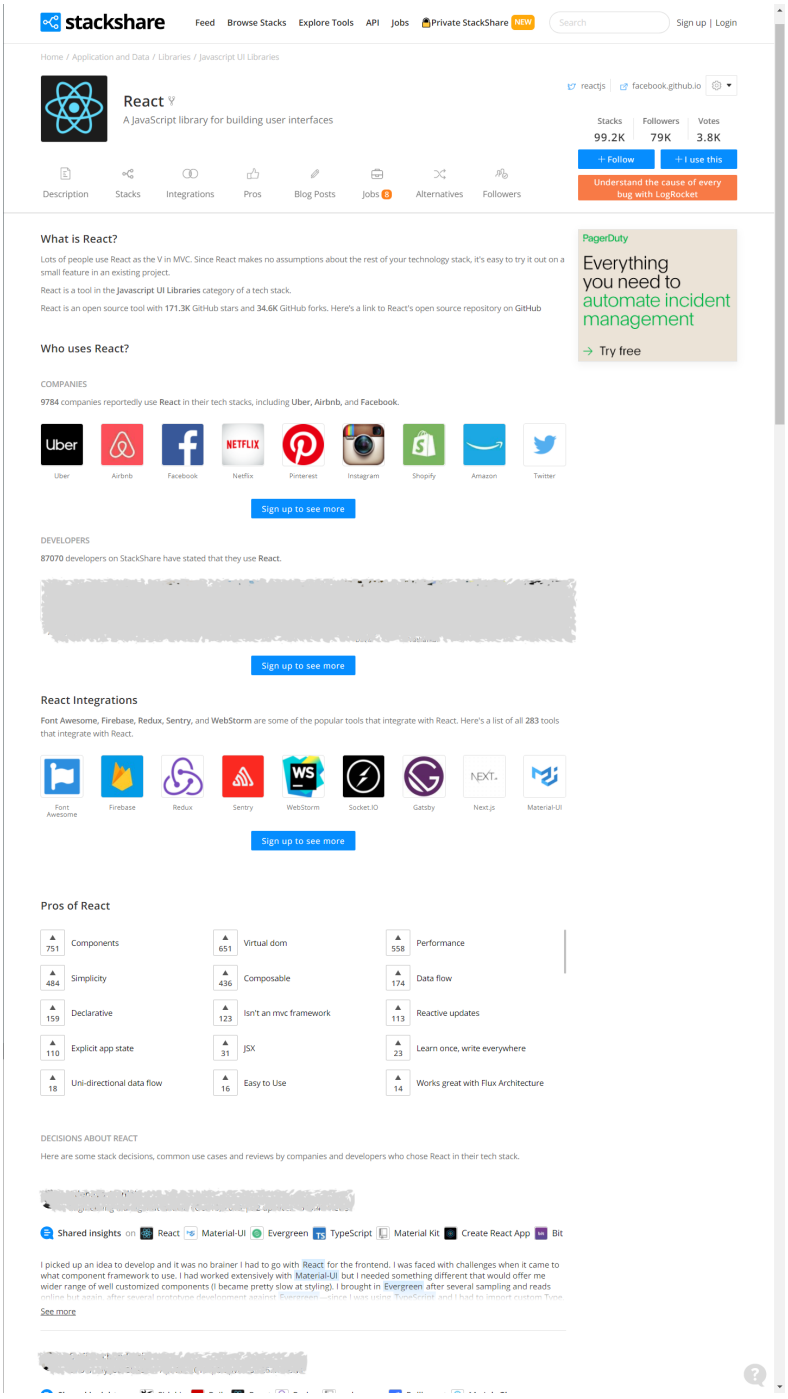


Figure 10.6: Snippet from Stackshare, showing the frontend framework react. Identifying information has been blurred. Source: stackshare.io/react

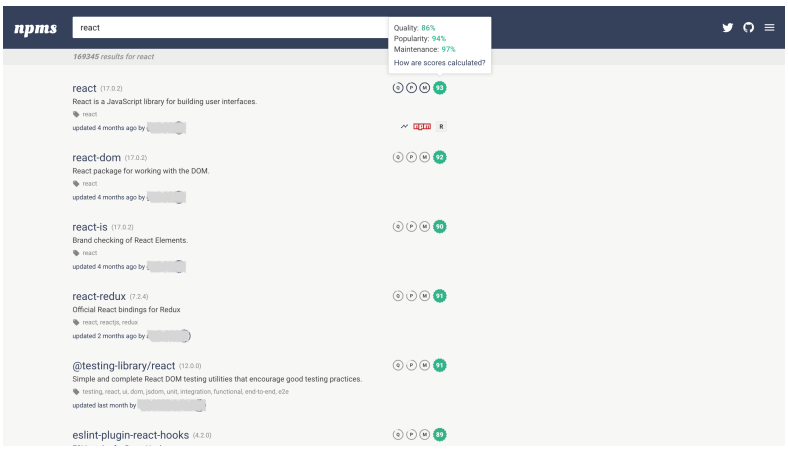


Figure 10.7: Snippet from npms.io, showing the frontend framework react.

Appendix C

C.1 Informed consent document

A document with this content was sent to all interviewees before their interviews. These documents were sent with an online signing tool so that they could be electronically signed by the participants.

What this study is about
The purpose of this study is to understand how people select open source software. Your participation in this study will help us make the product easier to use.

Your participation in this study is voluntary
You can take a break at any time. Just tell the researcher if you need a break. You can leave at any time without giving a reason.

Information we want to collect
We will ask you to show us how you use the product. We will watch how you do various tasks and we will ask you some questions. We will record the session and we will take notes to record your comments and actions.

How we ensure your privacy
People involved in the design of the product may watch the recording of your session in the future. These recordings will be treated as confidential and will not be shared outside our company. We will publish a master thesis that may include your comments and actions but your data will be anonymous. This means your name and identity will not be linked in our research reports to anything you say or do.

Your consent
Please sign this form showing that you consent to us collecting these data. I give my consent:

- For people to observe me during the research.

- For the session to be recorded.
- For people on the design team to watch the recording in the future.

If you want to withdraw your consent in the future, contact the person named below who will destroy any personal data we hold about you (such as the recordings). Otherwise, we will delete your personal data when the master thesis is finished.

Filip Bolling, fi2886bo-s@student.lu.se

C.2 Interview questions

Process
Finns det variationer inom företaget, dels inom roller men också i olika intakes.
Skulle du säga att ditt företag har en process när det kommer till att välja OSS?
Beskriv hur ni går tillväga?
Skiljer sig processen mellan små importer och större arkitekturimporter?
Har ni några rutiner/sätt att kolla vilken OSS som redan finns i företaget?
Har ni några riktlinjer eller krav inom företaget?
Diskuteras valet av OSS i teamet, eller är det upp till var och en?
Vem har sista ordet när det kommer till att välja OSS?
Fördelar med nuvarande process?
Nackdelar med nuvarande process?
Något som saknas i processen idag?
Finns visioner om att utveckla processen?
Hur skulle DU vilja välja OSS, utifrån egna preferenser?
Hur gör andra? Både inom team/företag och generellt.

Factors/information
Vad är det första du kollar upp om ett projekt?
Vilka faktorer är relevanta? + rangordning?
Finns det någon faktor som är "non-negotiable" / "dealbreaker"?

Source of information
Var hittar du info när du utvärderar OSS?
Vilken typ av info hittar du var?
Kopplat till faktorerna, hur bra är källan på att lyfta fram den informationen?

Fördelar med källan?
Nackdelar med källan?
Finns det andra källor som du slutat använda?
Är det något du vill veta men sällan/aldrig kan hitta?

Prototype testing
Var hade du velat se vilken information?
Saknar du något?
Vad var bra?
Stämmer flödet överens med din konceptuella modell?
Fungerar flödet för både stora och små beslut?

Appendix D

D.1 Prototype 1

Figure 10.8: Start page

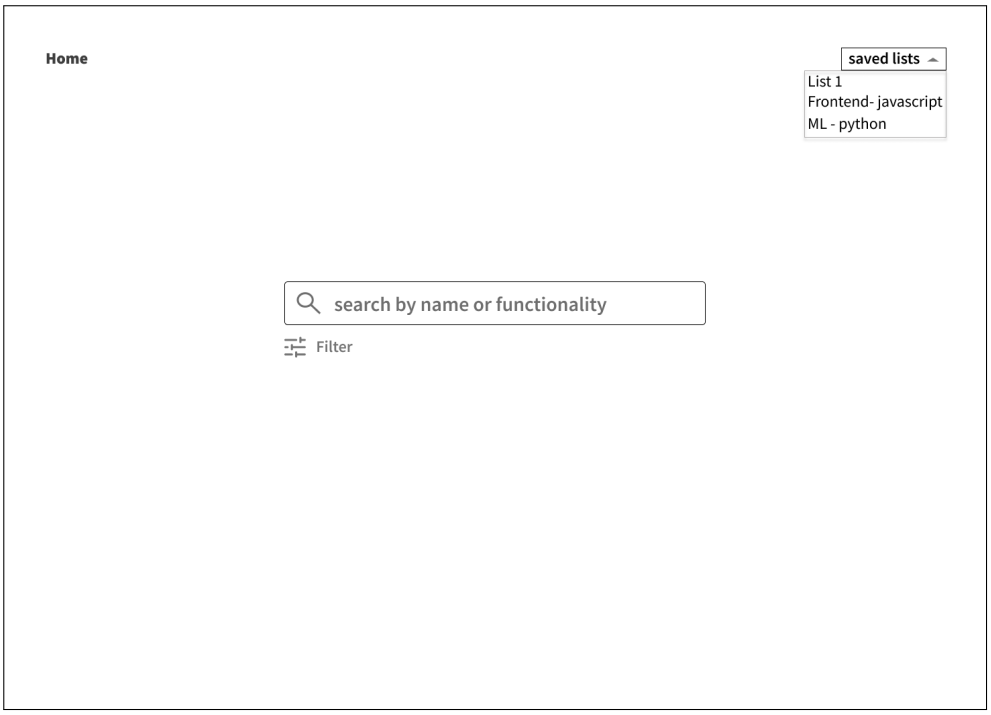


Figure 10.9: Start page showing the list functionality.

Figure 10.10: Start page showing the filter functionality.

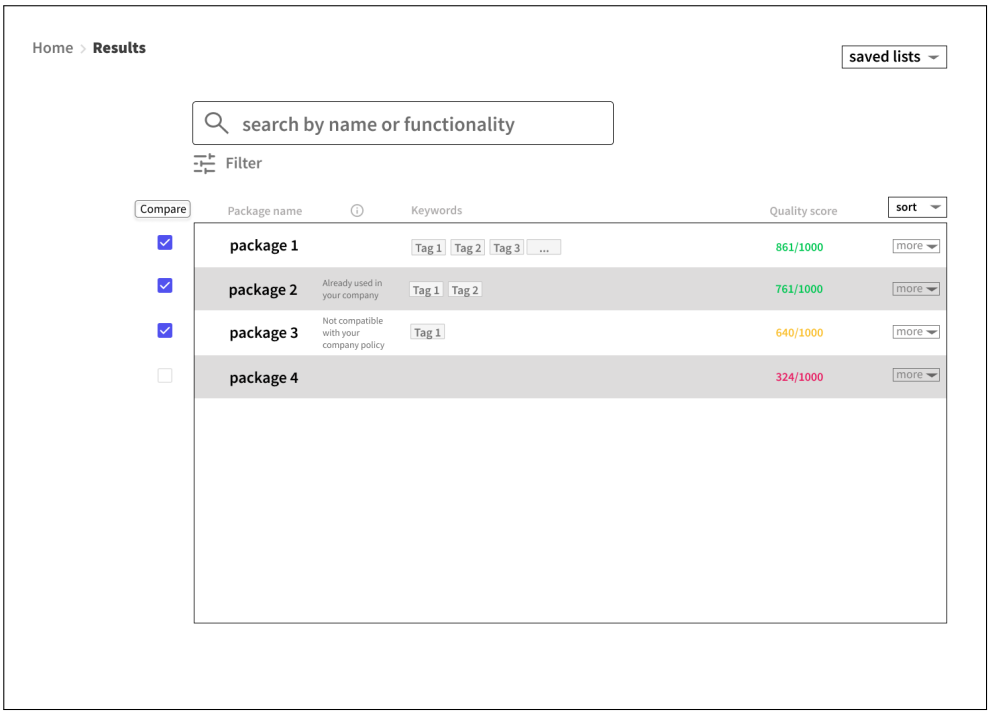


Figure 10.11: The search results from a random search.

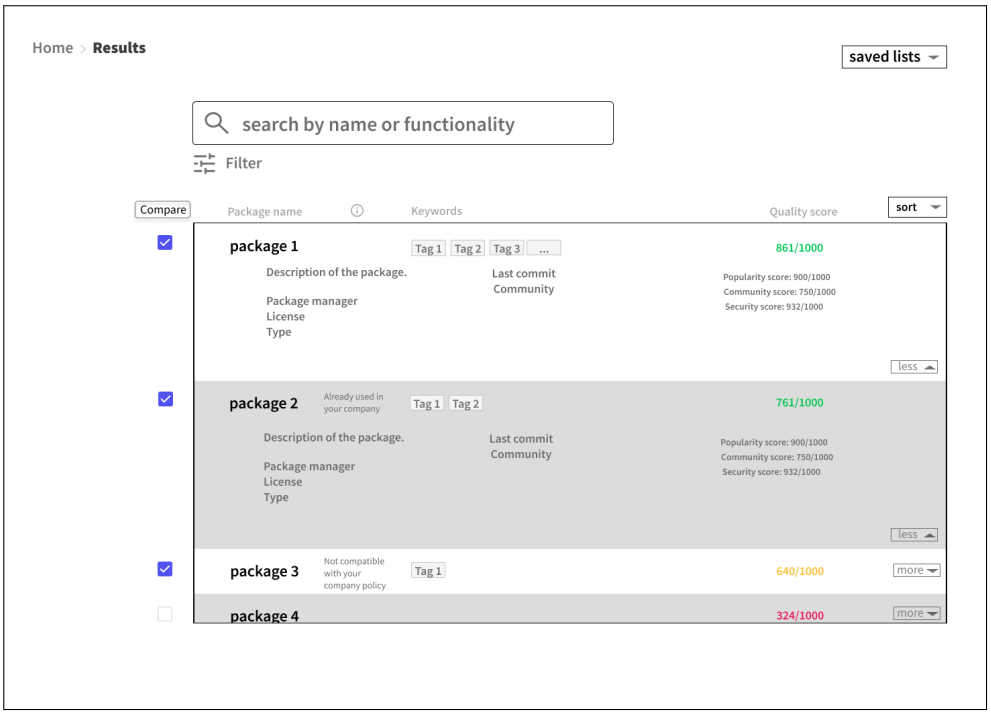


Figure 10.12: The search result in the extended version.

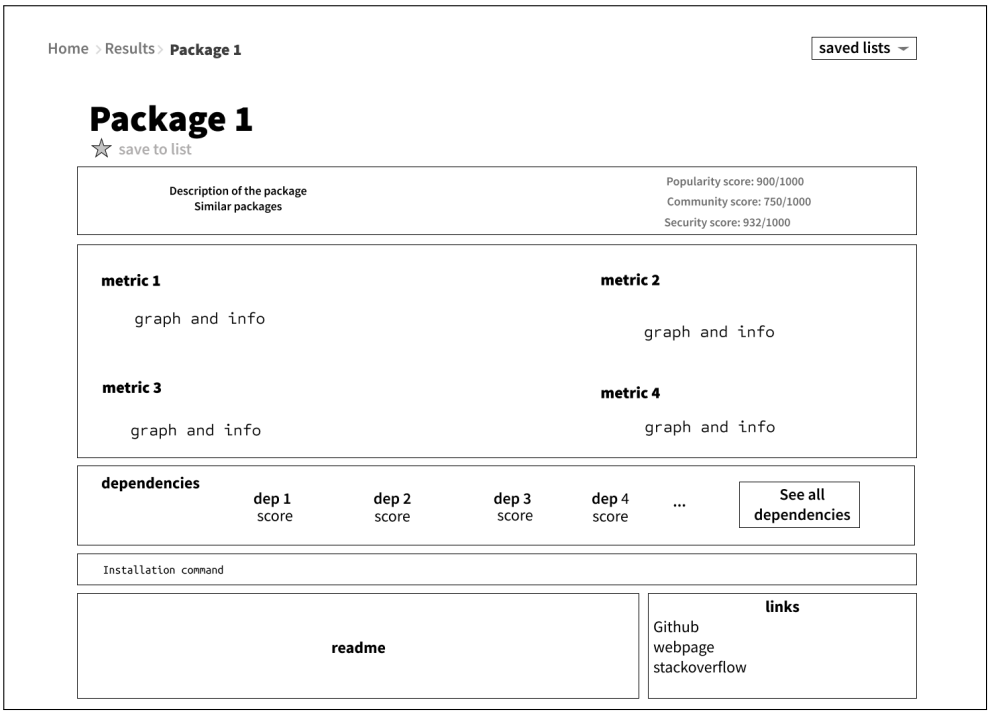


Figure 10.13: Package page for package 1.

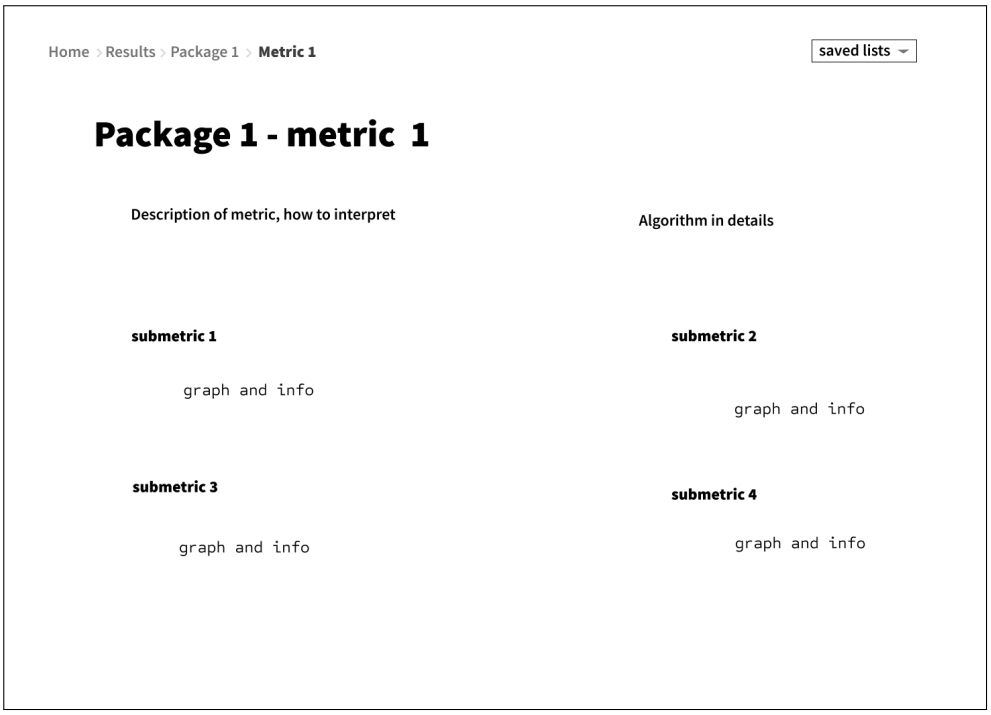


Figure 10.14: Package page showing the deep analysis options.

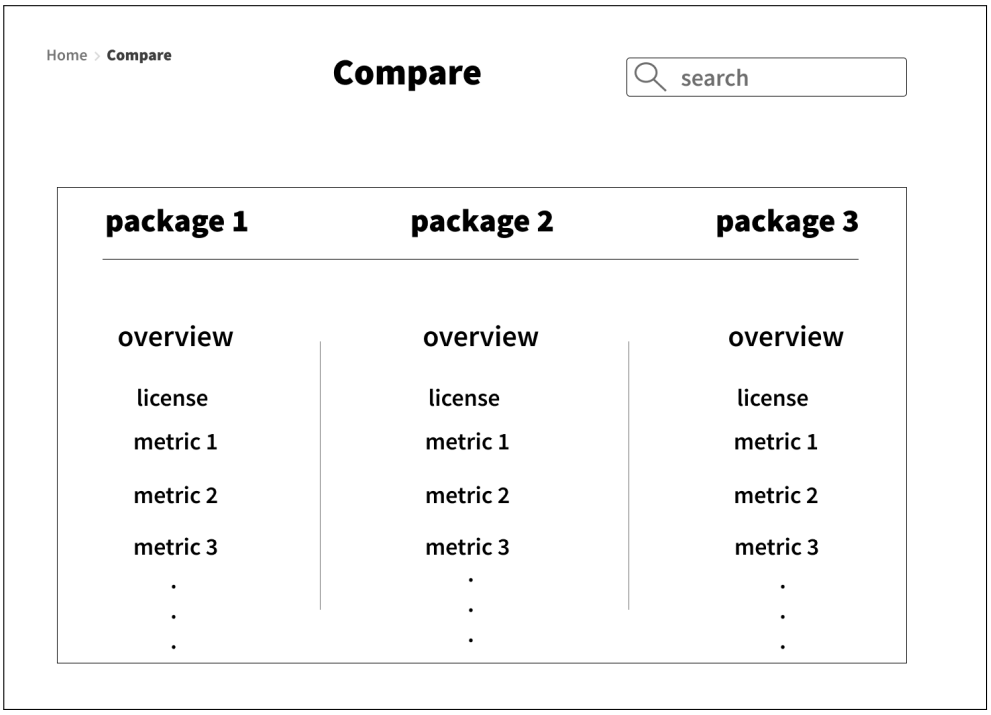


Figure 10.15: The compare functionality with the comparison between package 1 and 2.

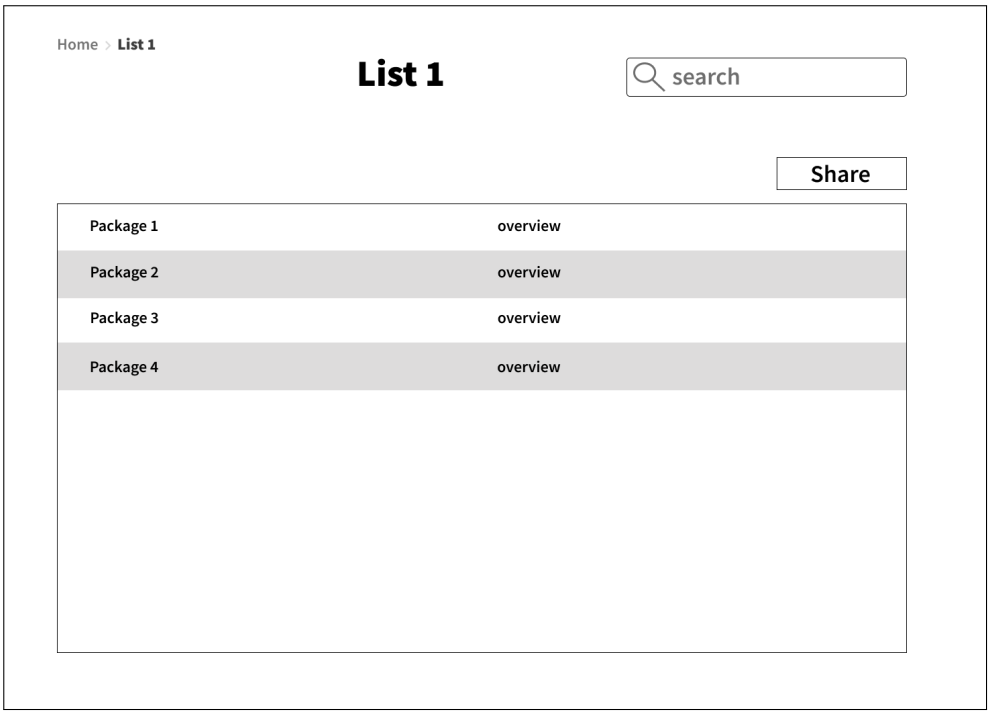


Figure 10.16: List with saved packages.

D.2 Prototype 2

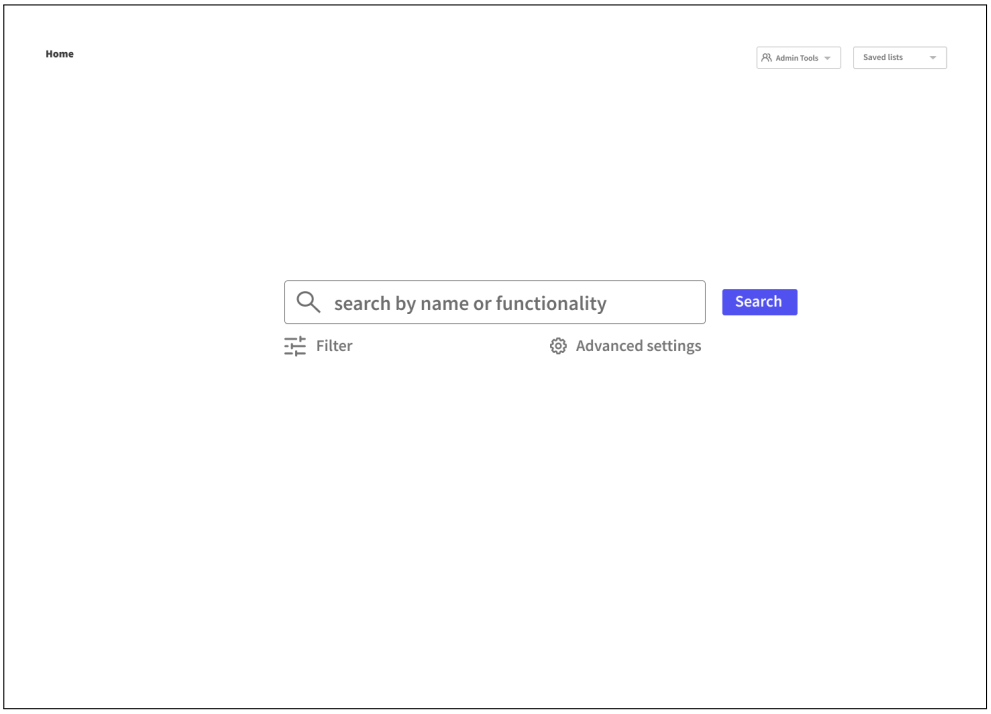
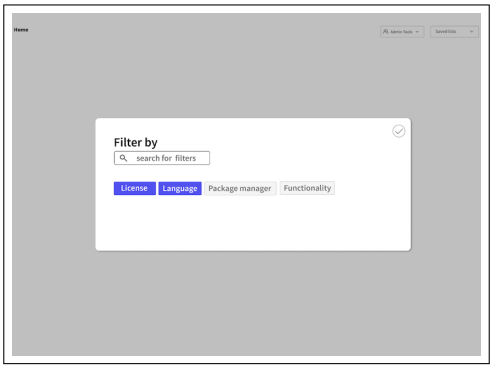
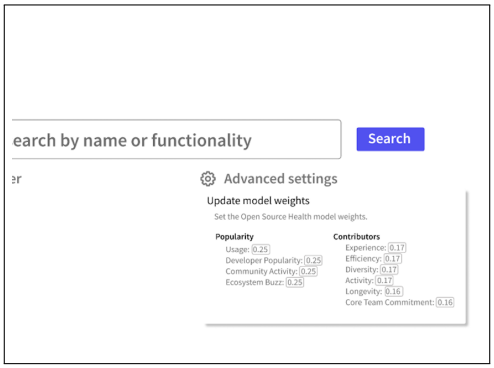


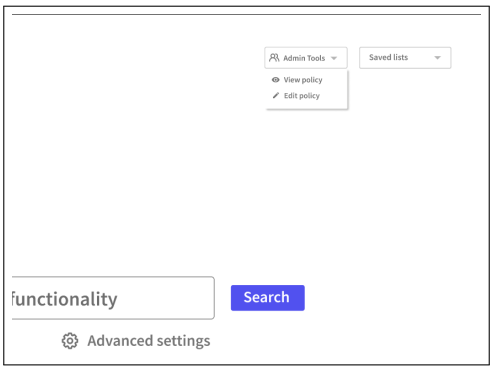
Figure 10.17: Start page iteration 2.



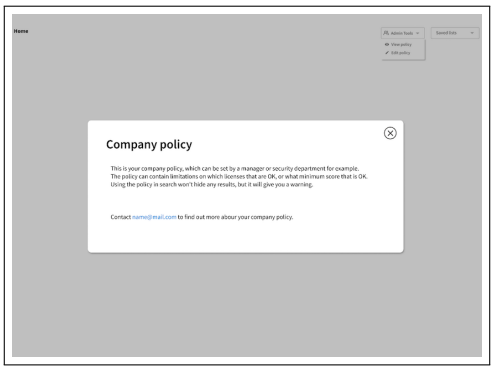
(a) Filter



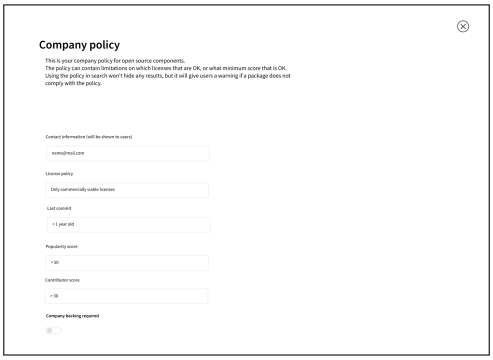
(b) Model weights



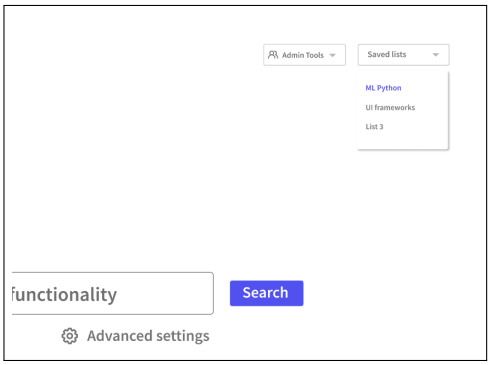
(c) Admin tools



(d) View policy



(e) Edit Policy



(f) List

Figure 10.18: Sub features of the start page.

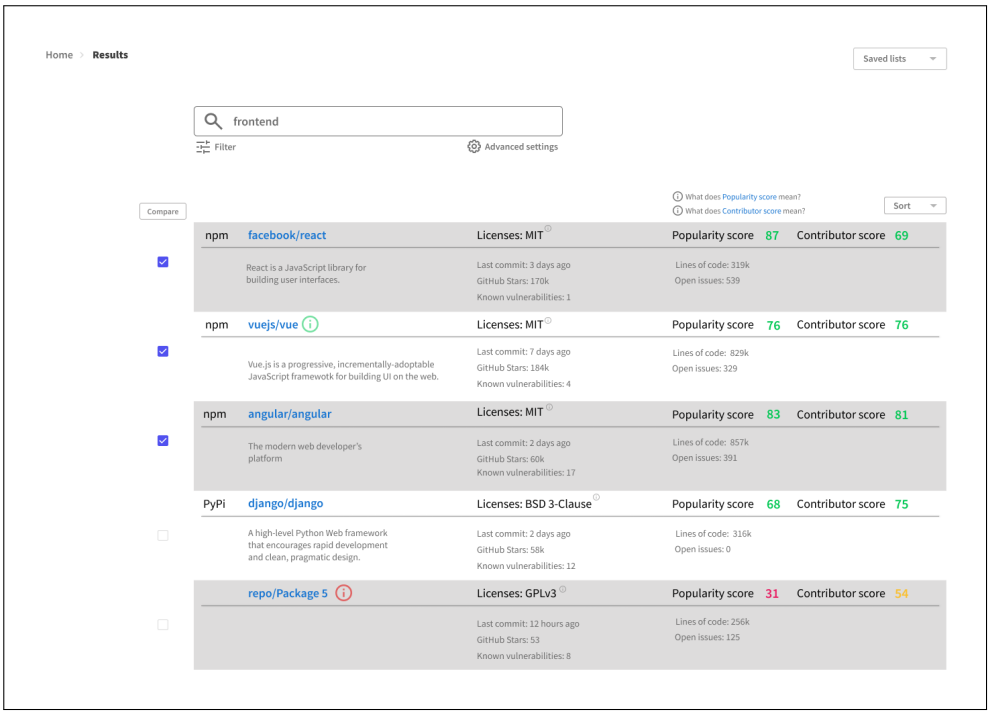
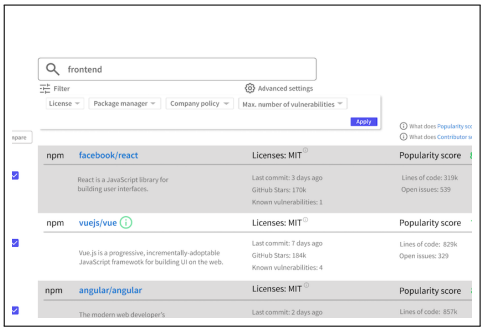
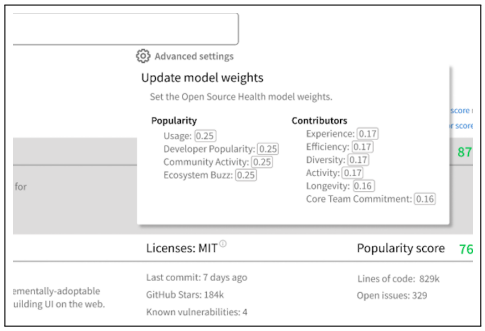


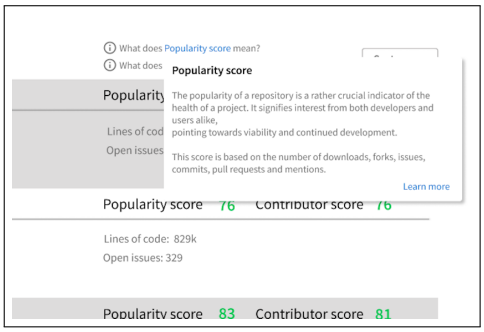
Figure 10.19: Search result with search word ”frontend”.



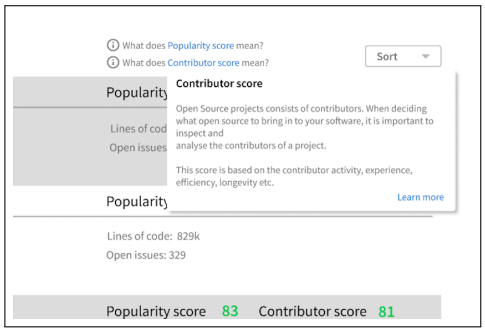
(a) Filter the result



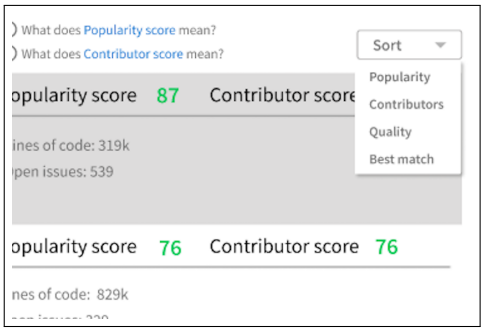
(b) Update model weights



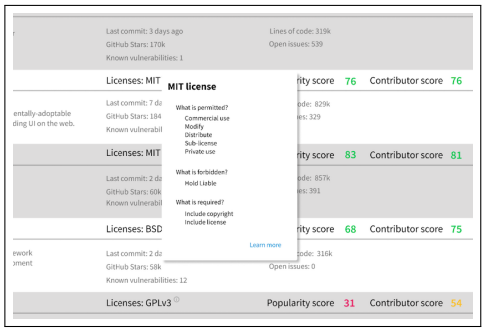
(c) Popularity description



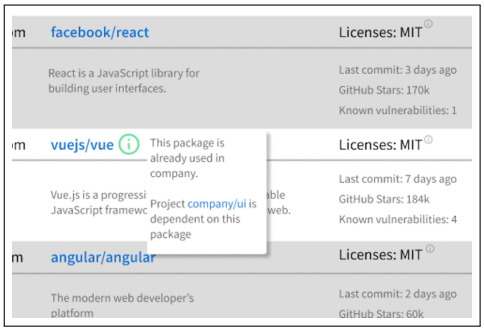
(d) Contributor description



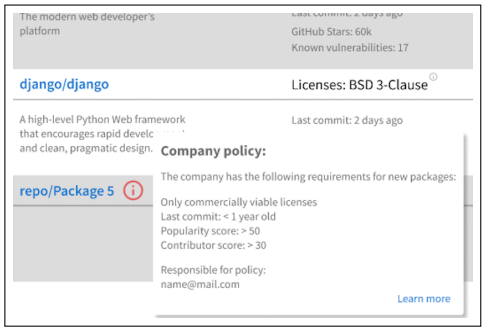
(e) Sort results



(f) License description



(g) Contextualization



(h) Policy trigger

Figure 10.20: Sub features of the search result page.

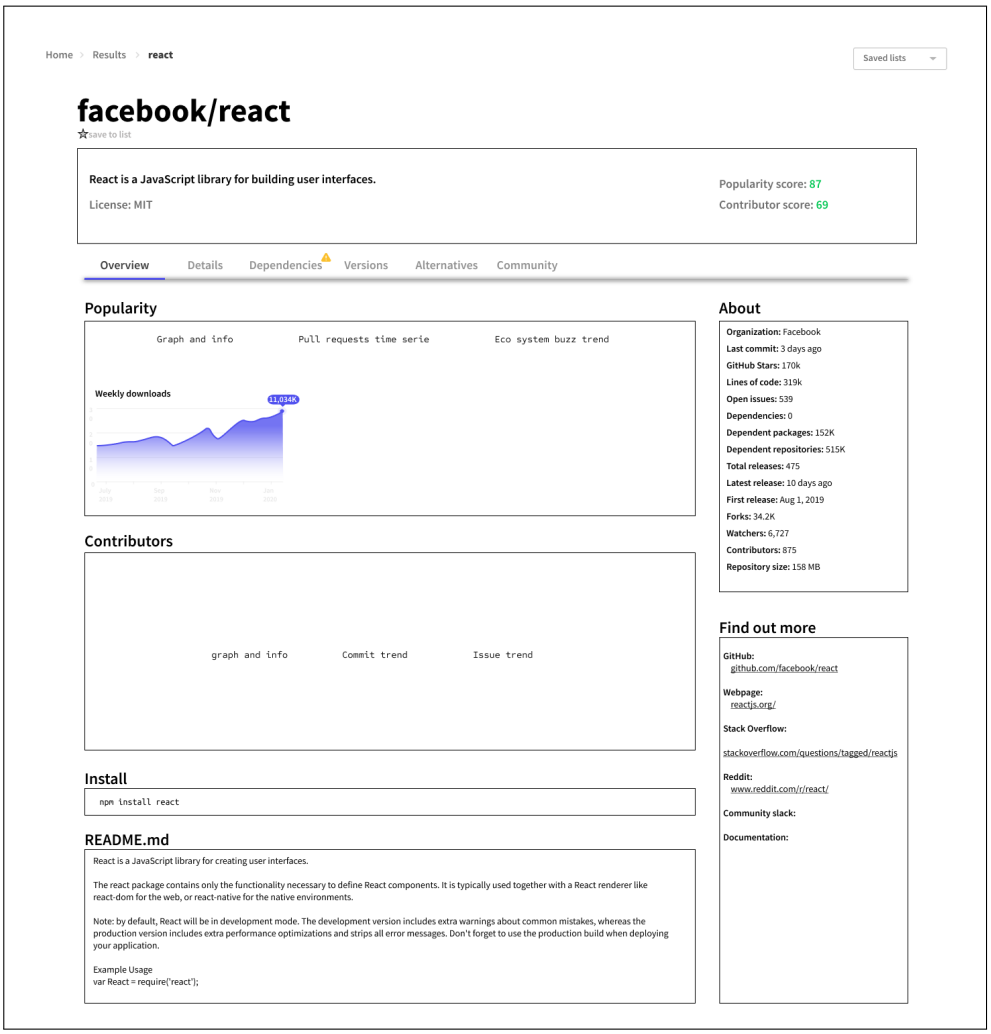


Figure 10.21: Package page for React package.

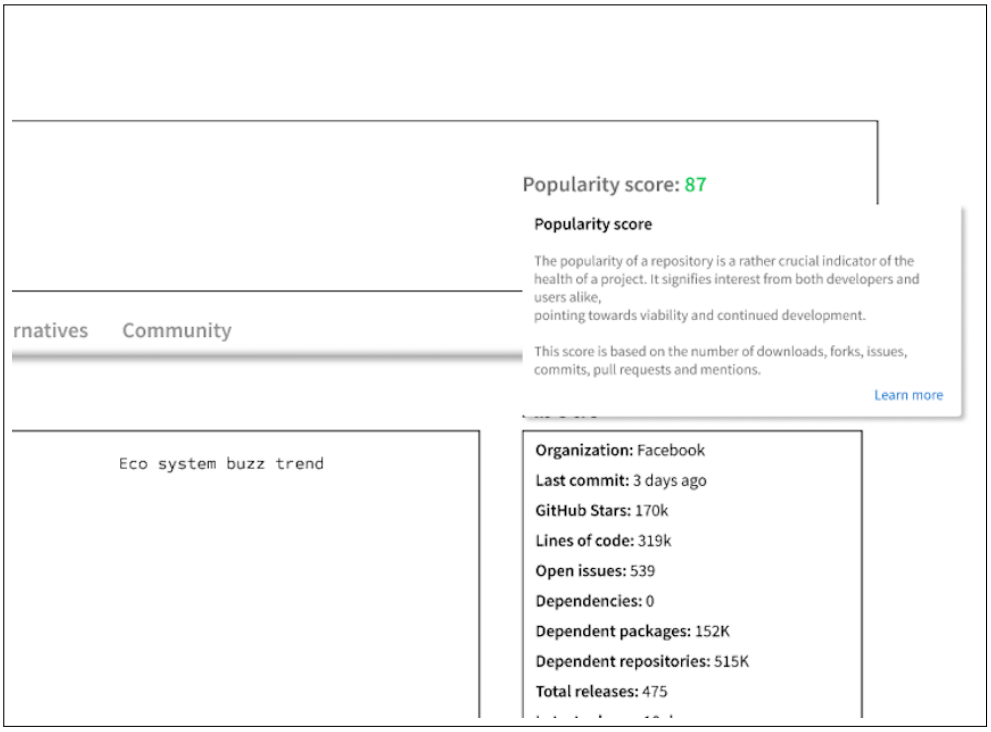


Figure 10.22: Popularity score description

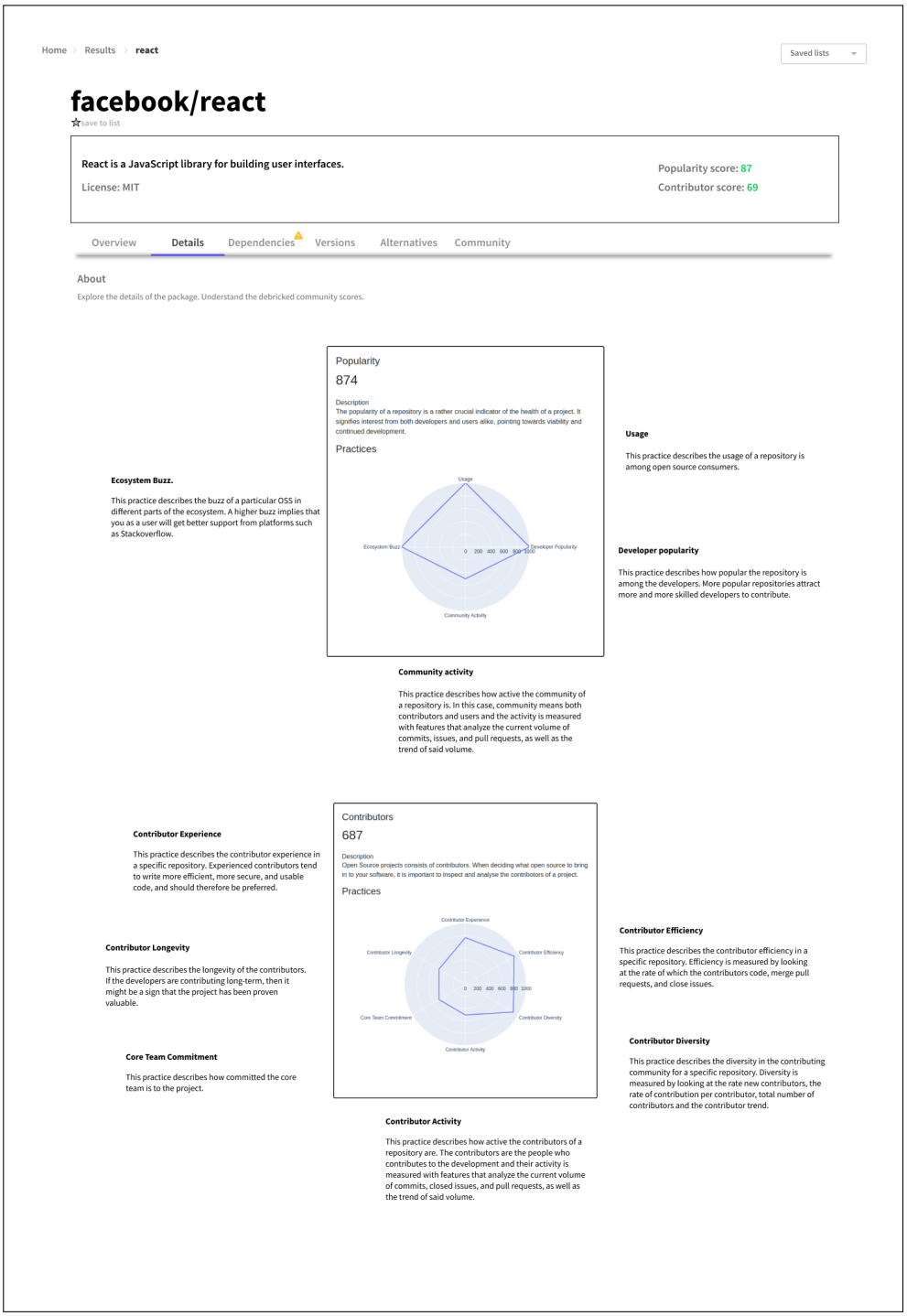


Figure 10.23: Detail view for React package.

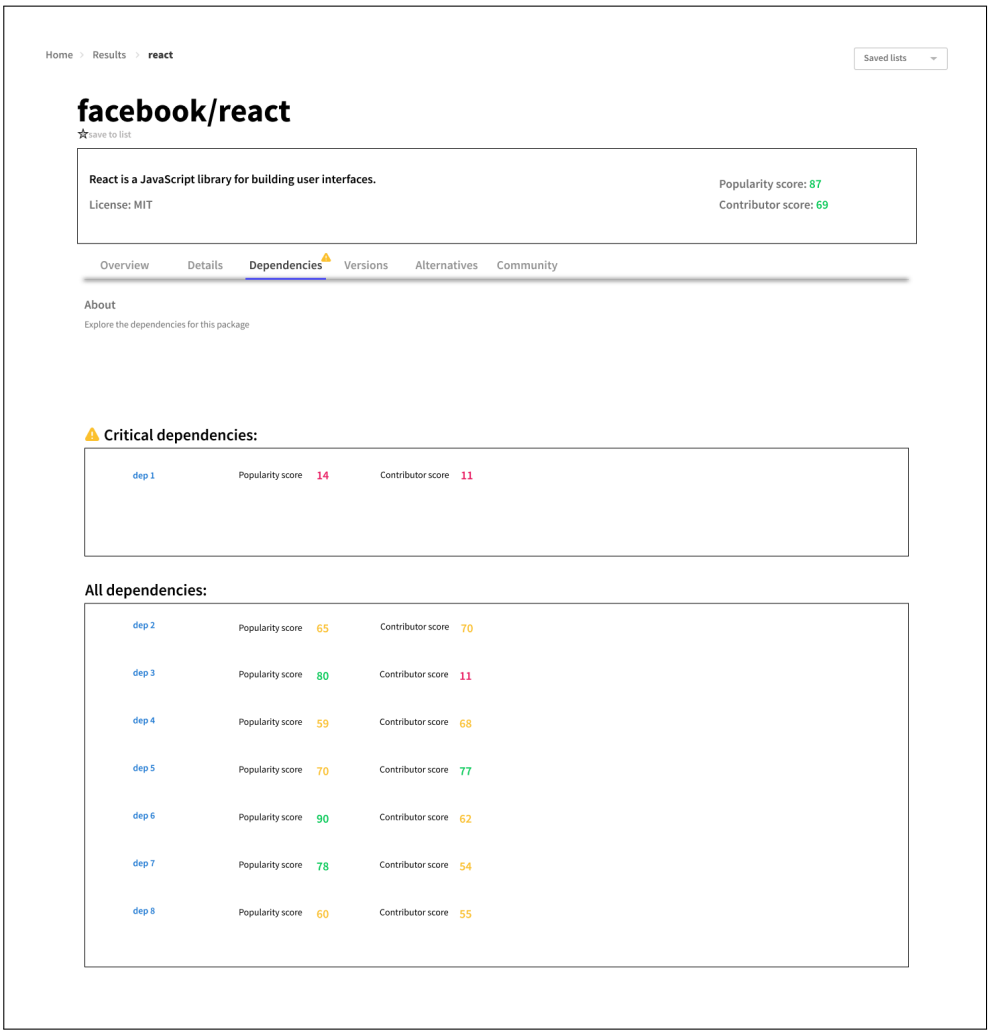


Figure 10.24: Dependency view, dependencies of React.

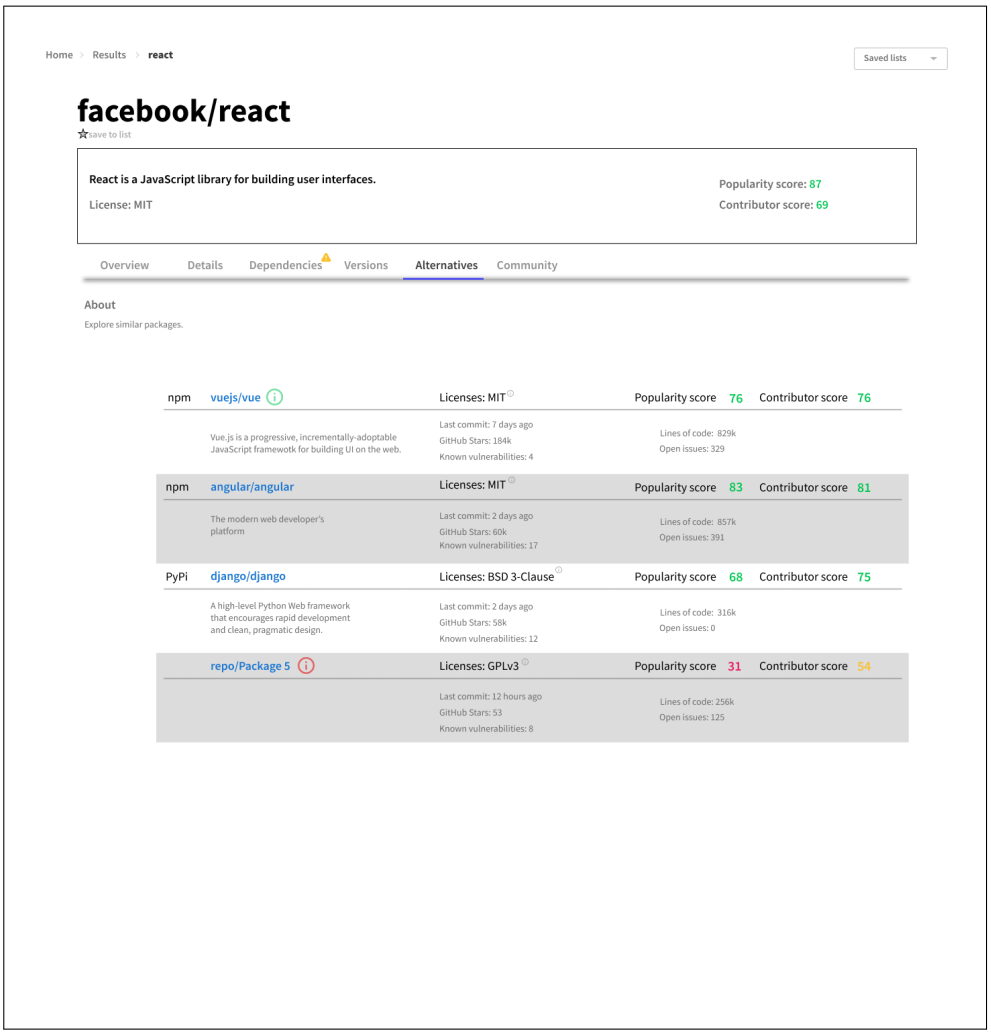


Figure 10.25: Similar packages to React.

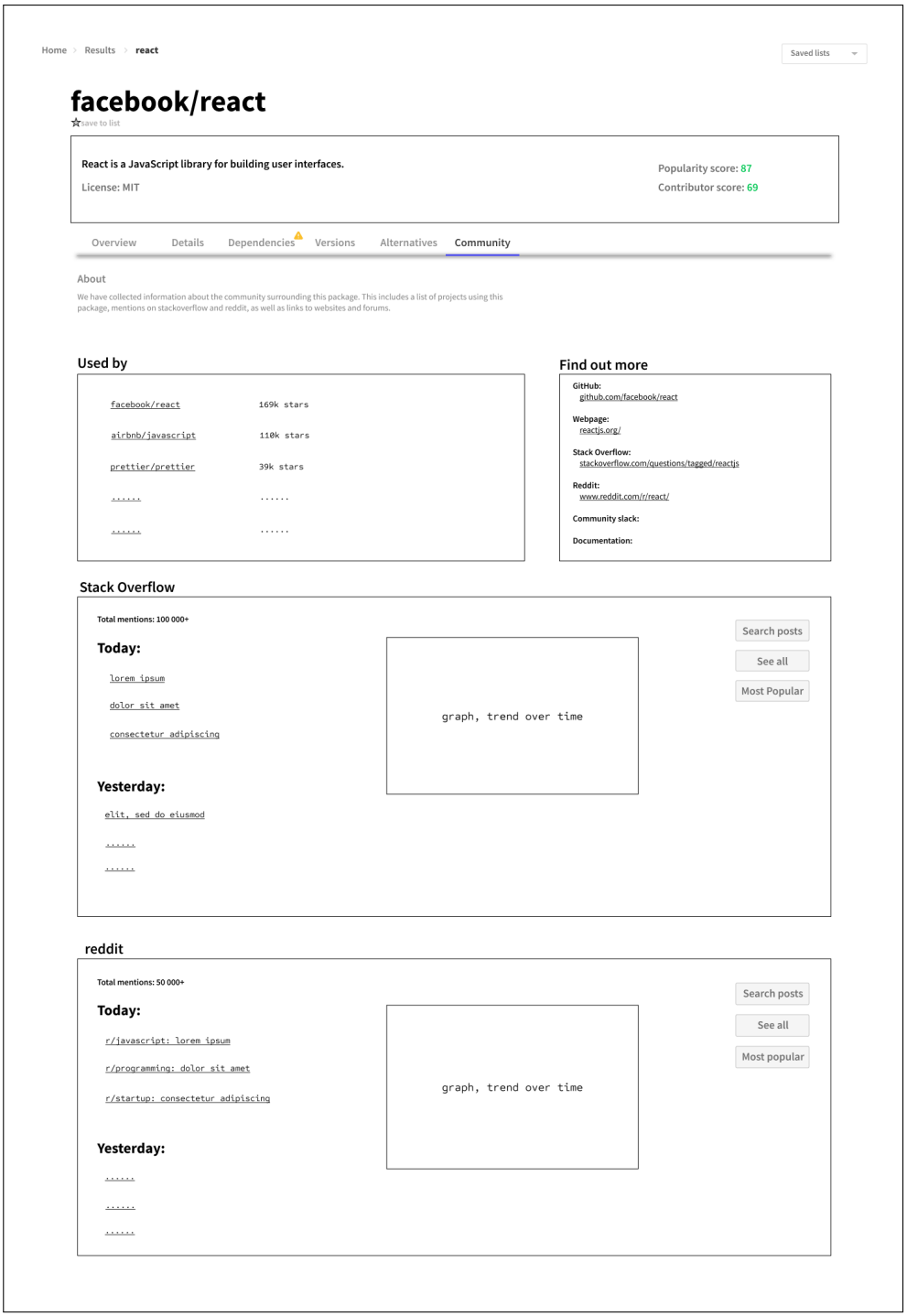


Figure 10.26: Exploring the React community.

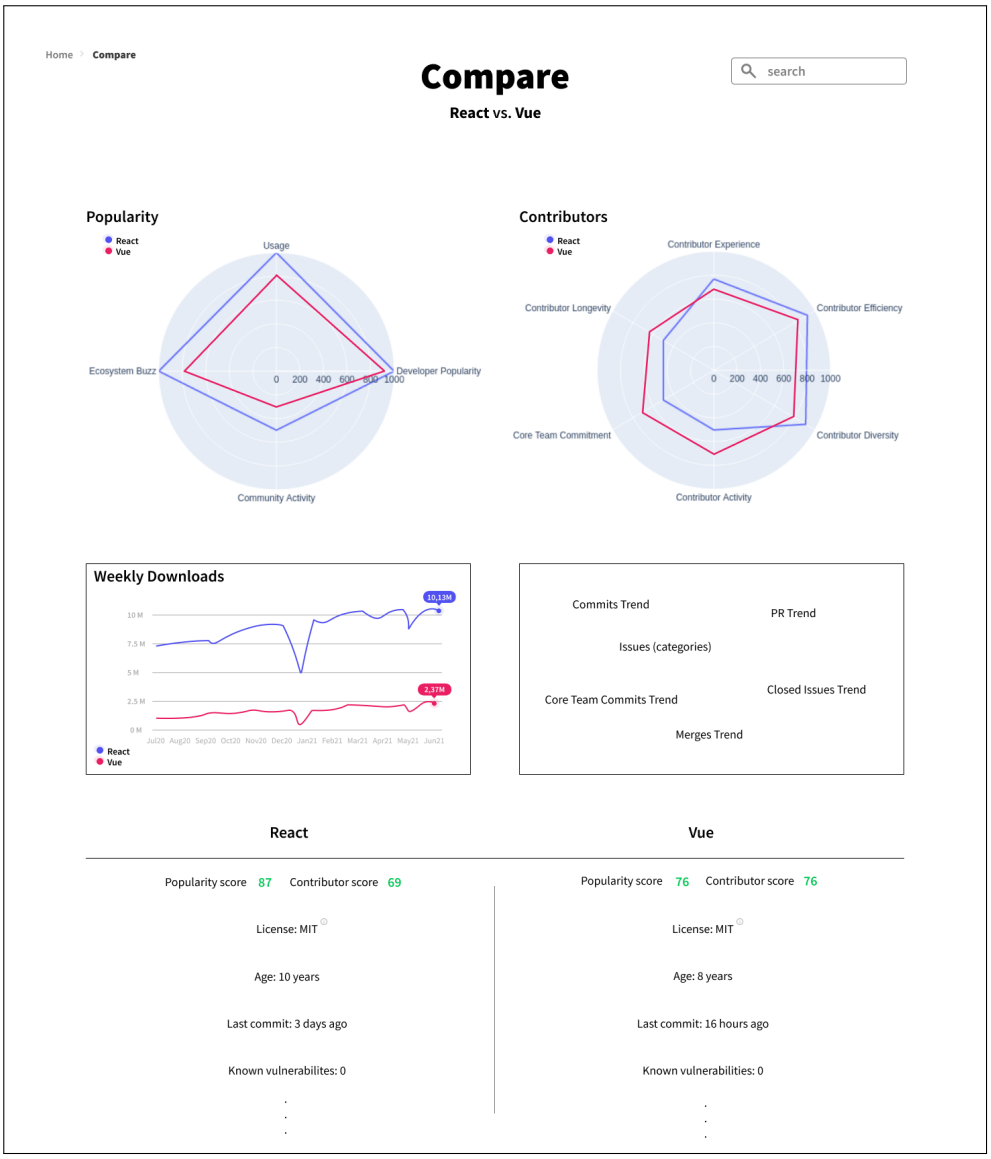


Figure 10.27: Compare functionality, comparison between React and Vue.

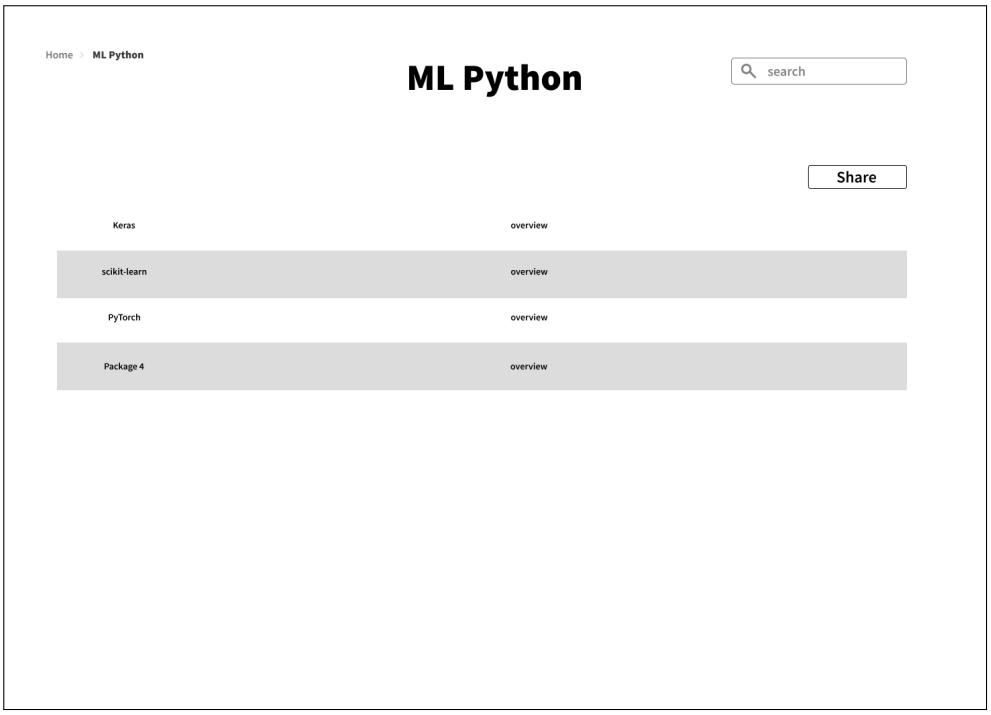
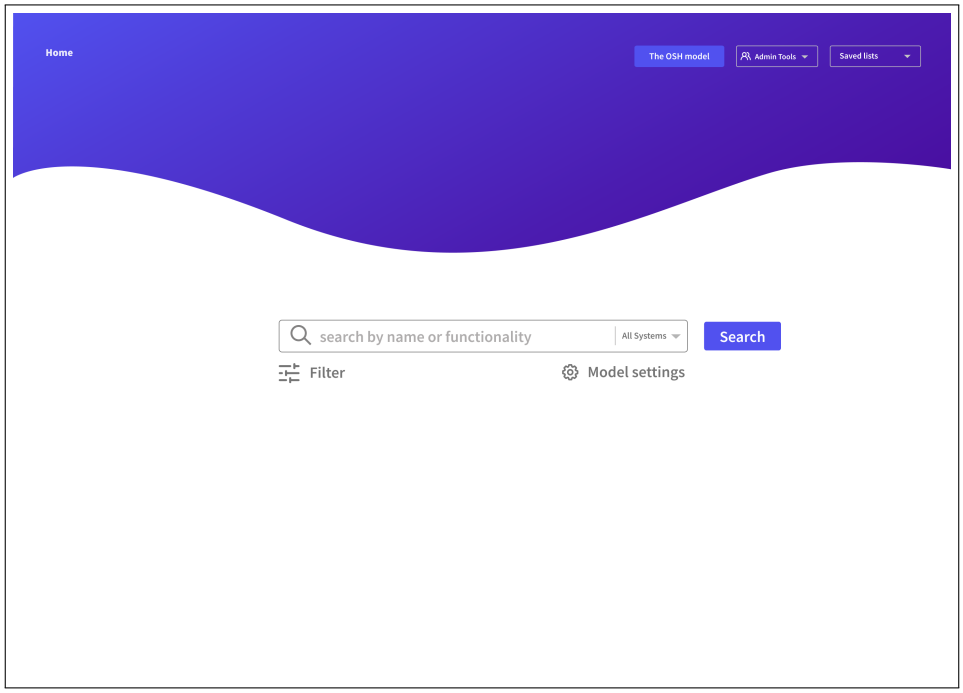
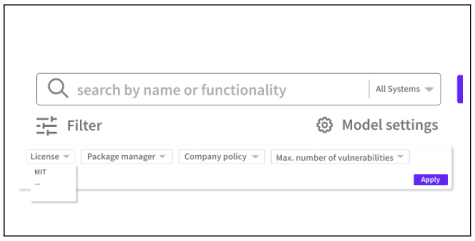


Figure 10.28: List with saved machine learning packages in Python.

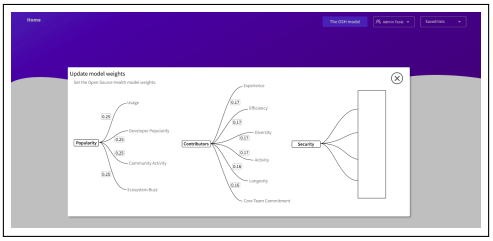
D.3 Prototype 3



(a) Start page iteration 3 - full view.

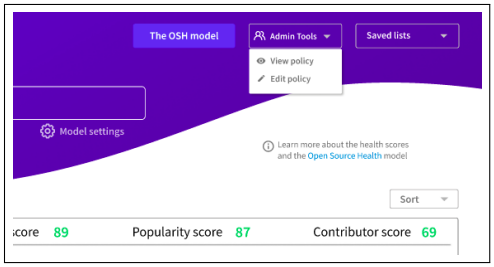


(b) Filter

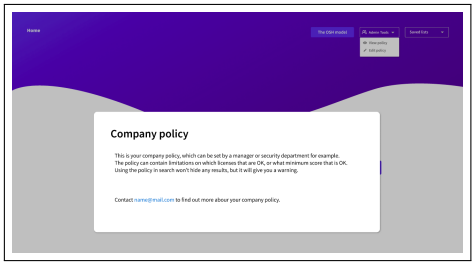


(c) Model settings

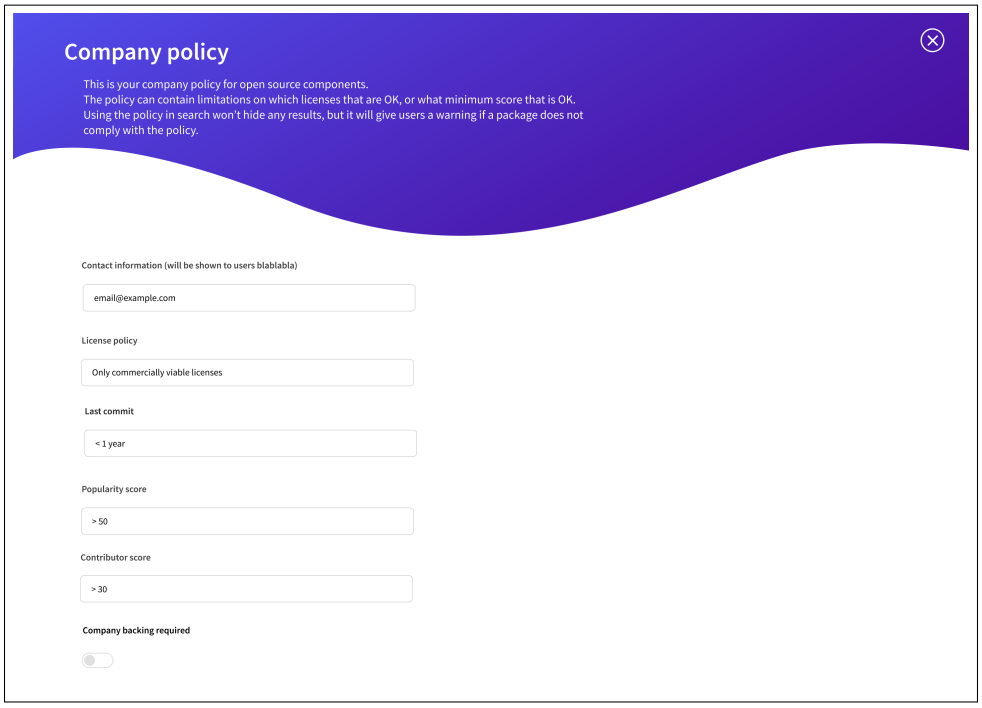
Figure 10.29: Start page iteration 3.



(a) Admin tools

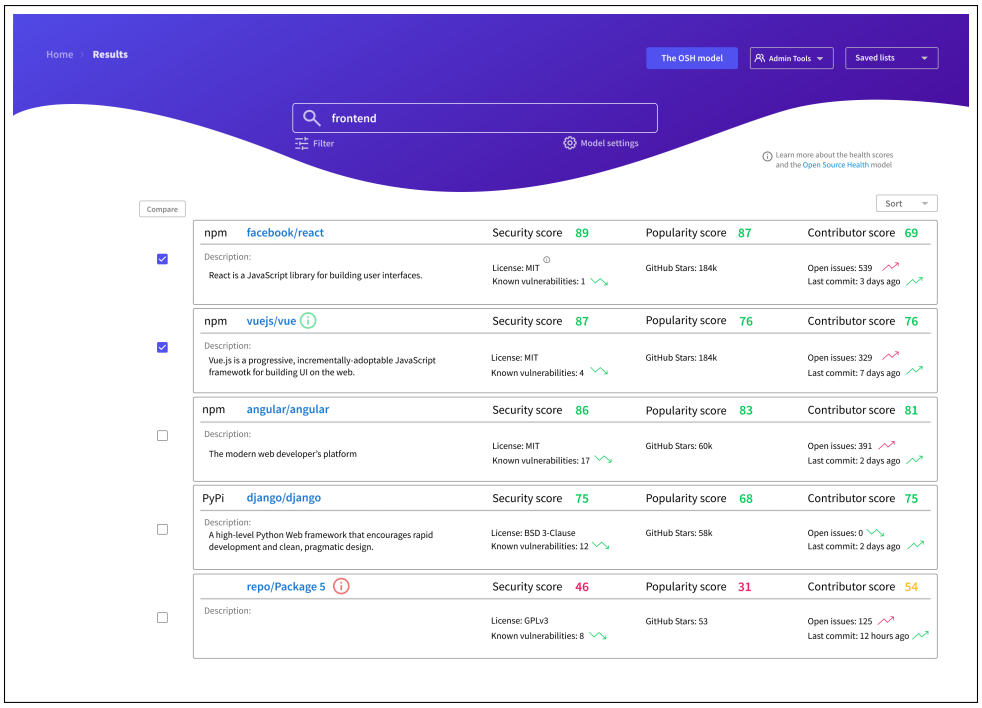


(b) View policy

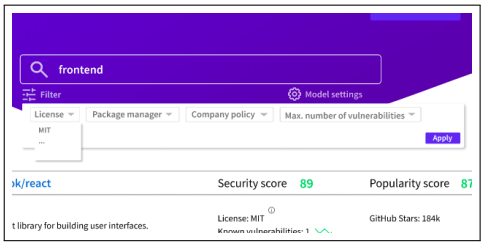


(c) Edit Policy

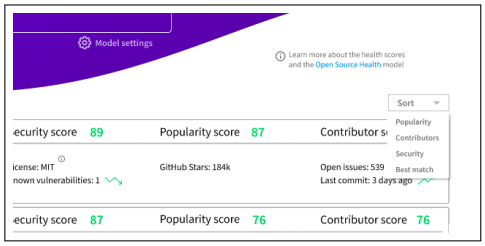
Figure 10.30: Policy



(a) Search result with search word ”frontend” - full view.



(b) Filter the result



(c) Sort results

Figure 10.31: Search result with search word ”frontend”.

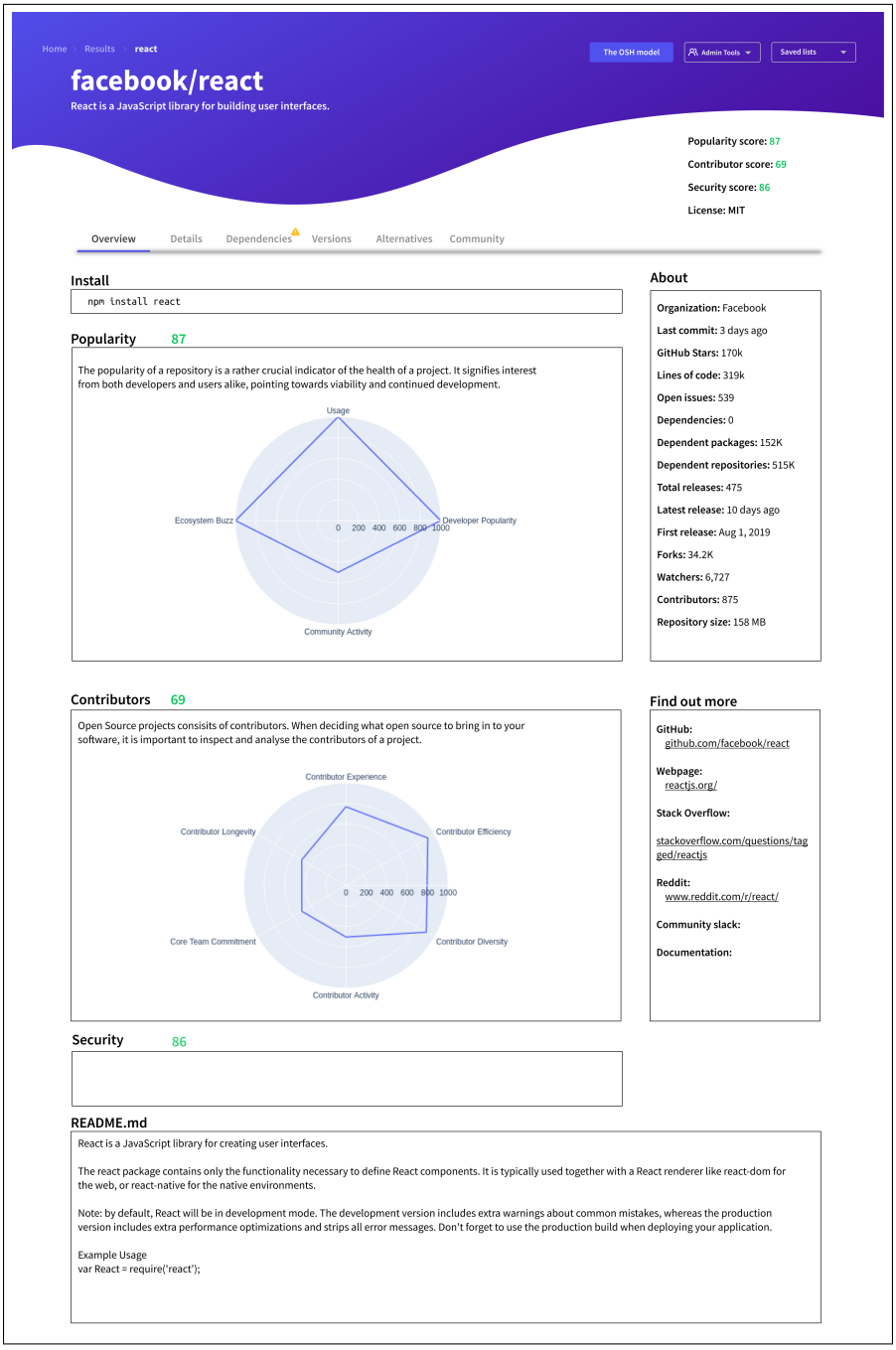
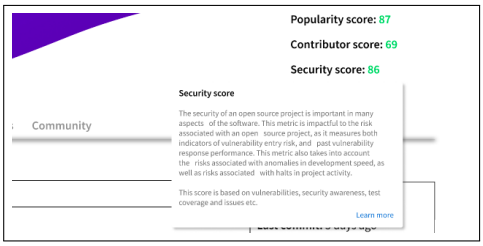
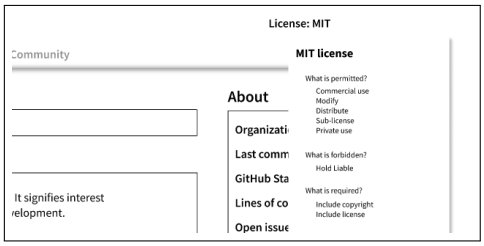


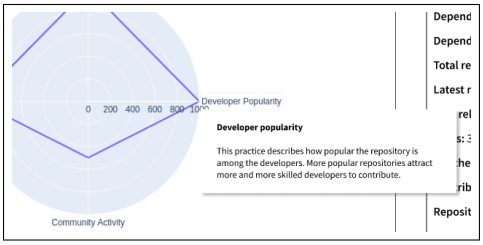
Figure 10.32: Package page for React package.



(a) Security score description



(b) License description



(c) Security score description

Figure 10.33: Subfeatures at the package page.

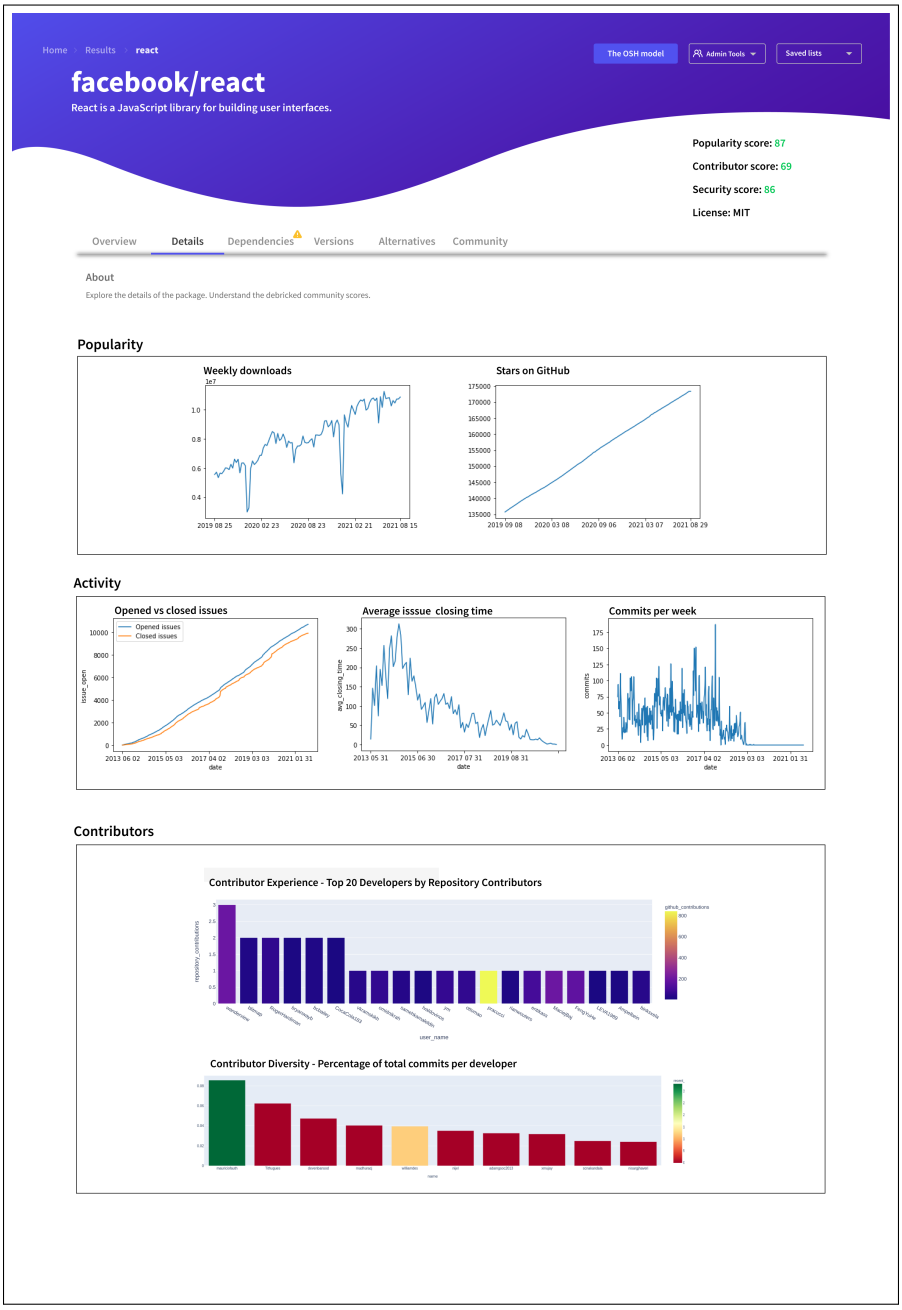


Figure 10.34: Detail view for React package.

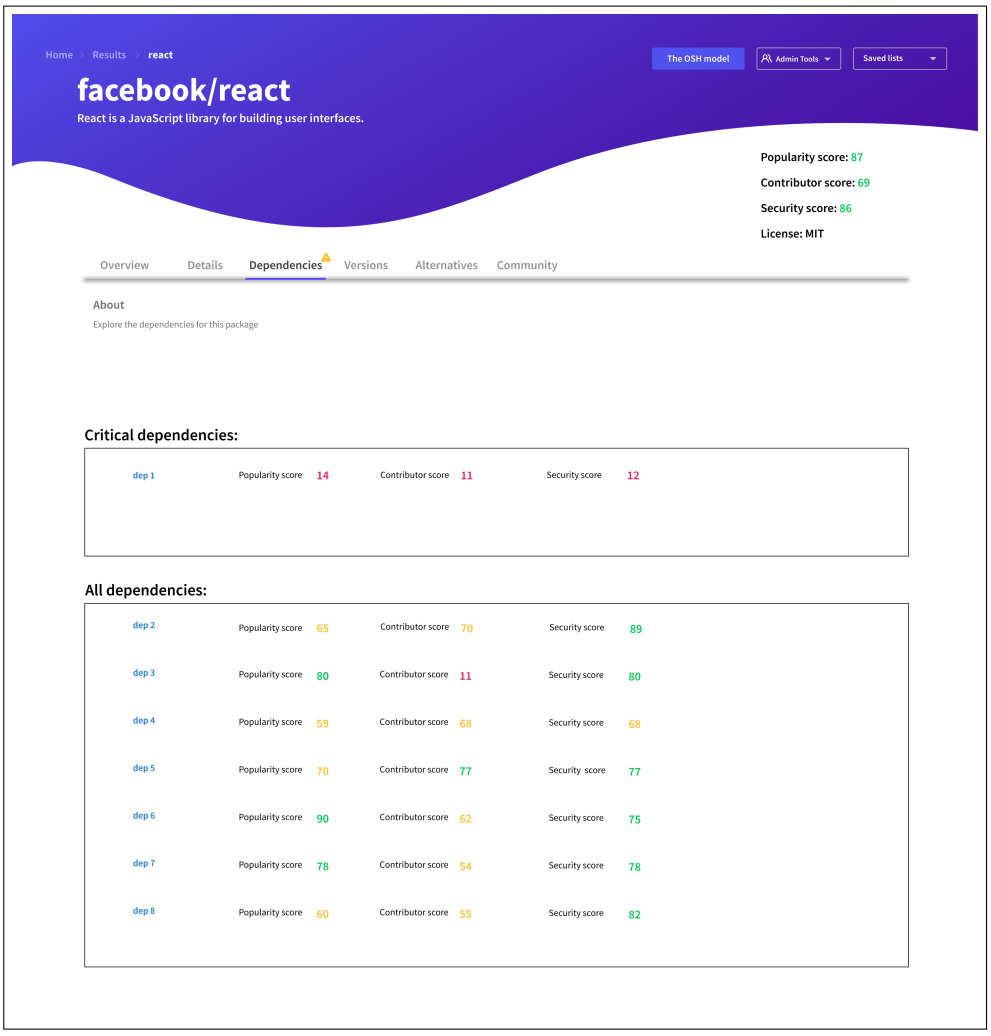


Figure 10.35: Dependency view, dependencies of React.

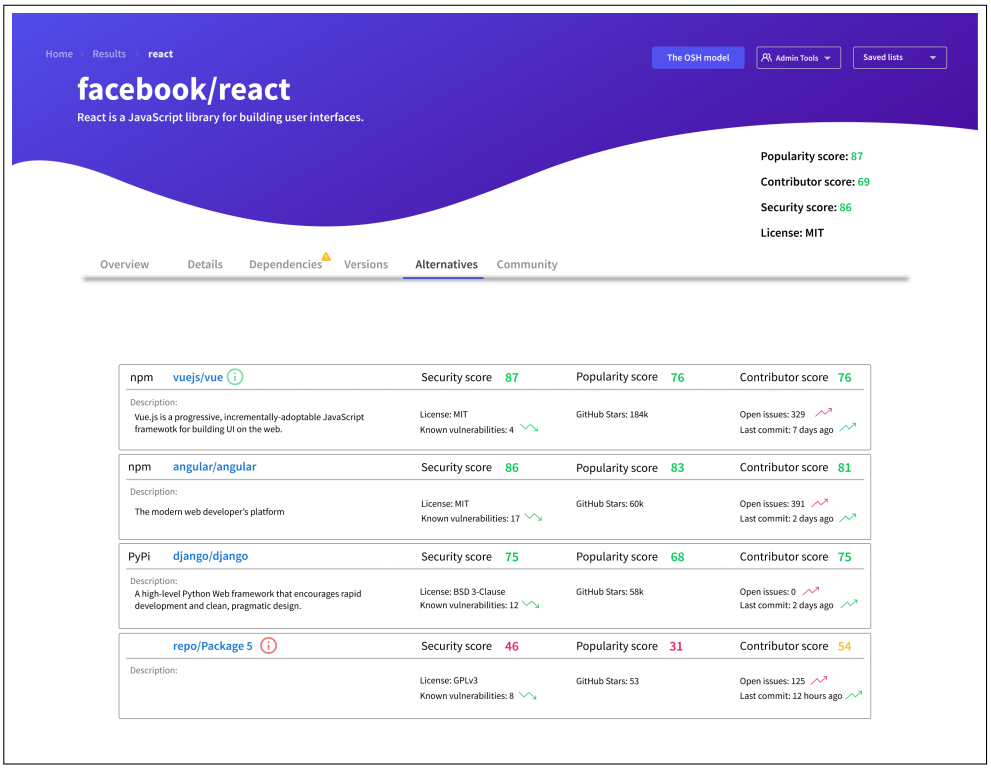


Figure 10.36: Similar packages to React.

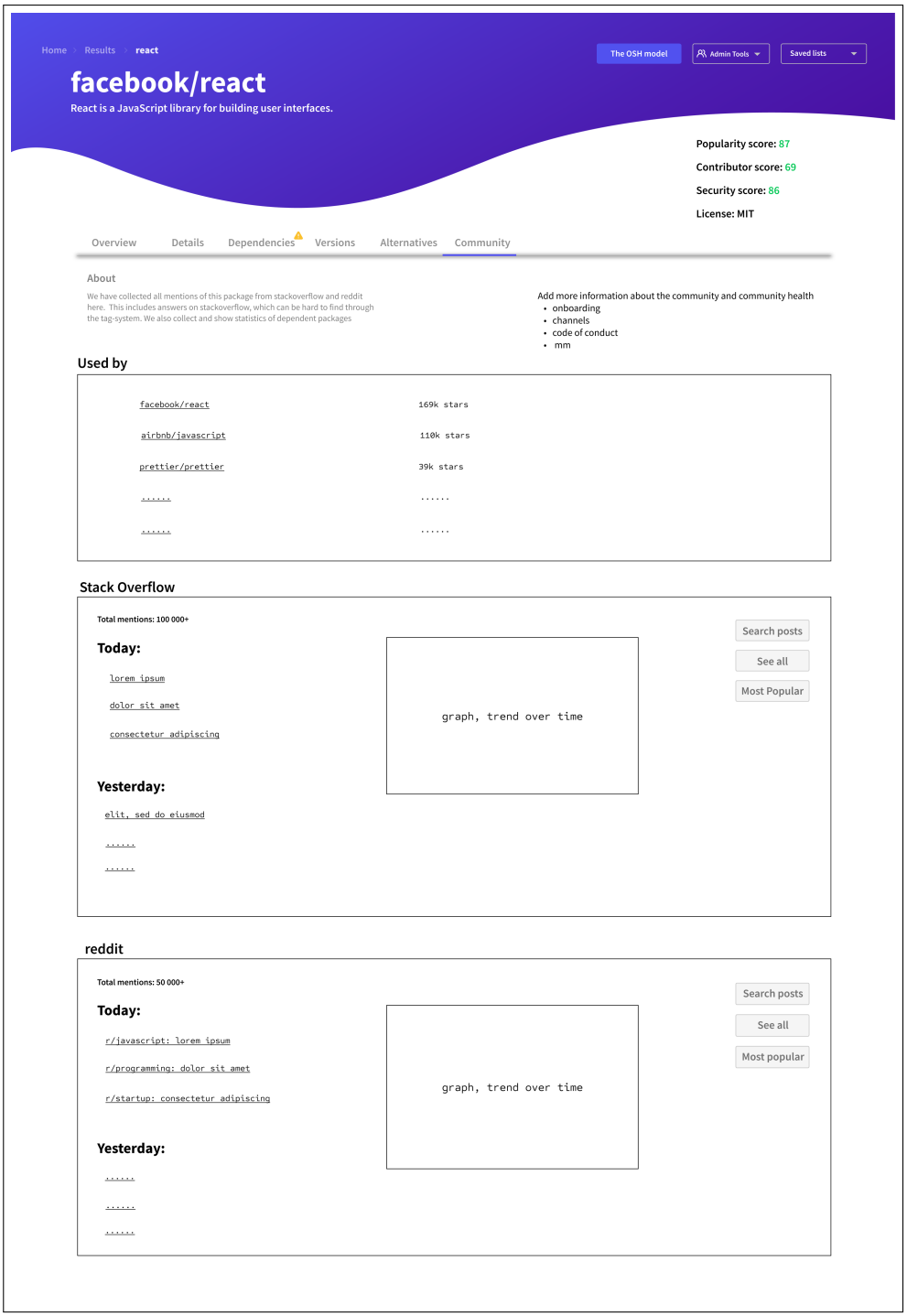


Figure 10.37: Exploring the React community.

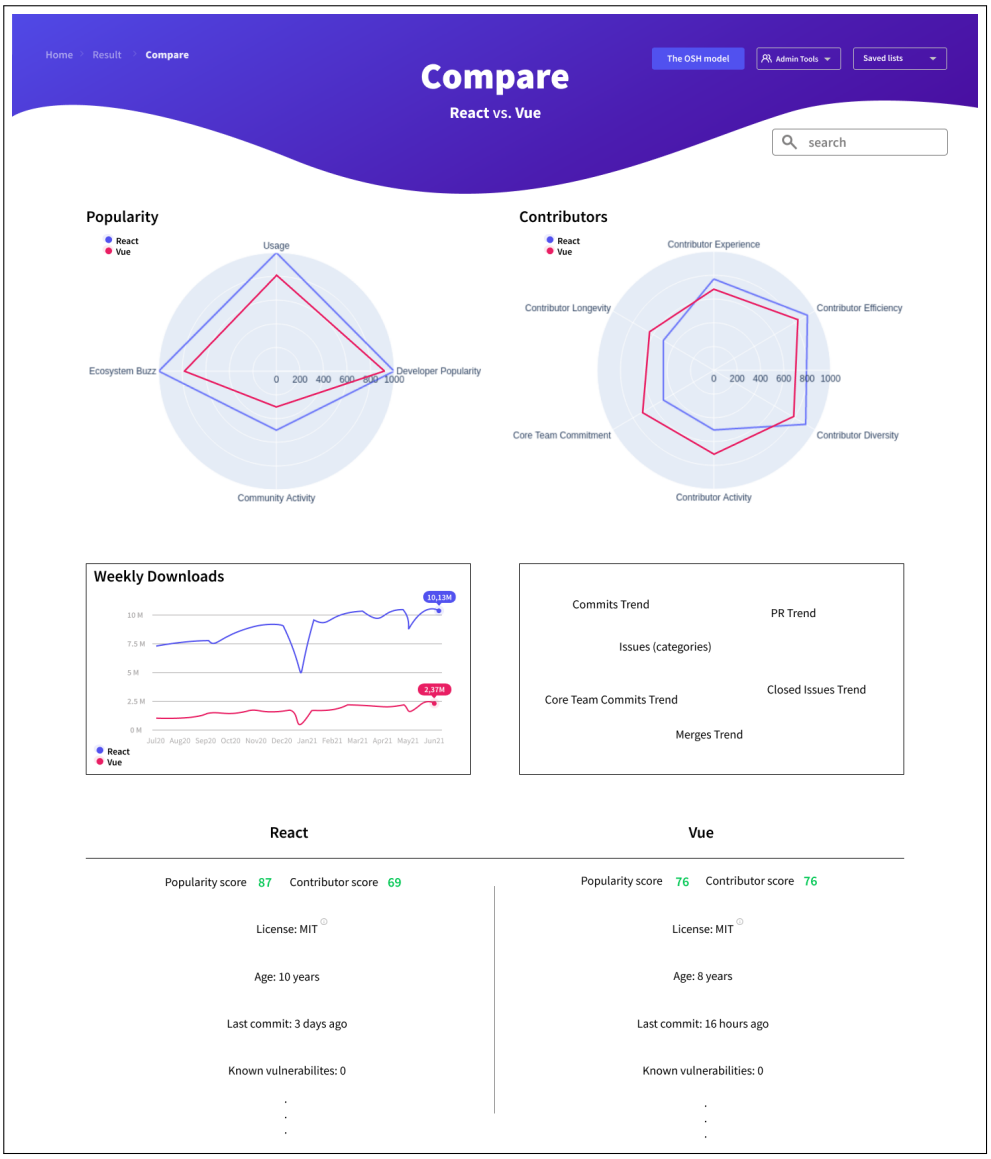


Figure 10.38: Compare functionality, comparison between React and Vue.

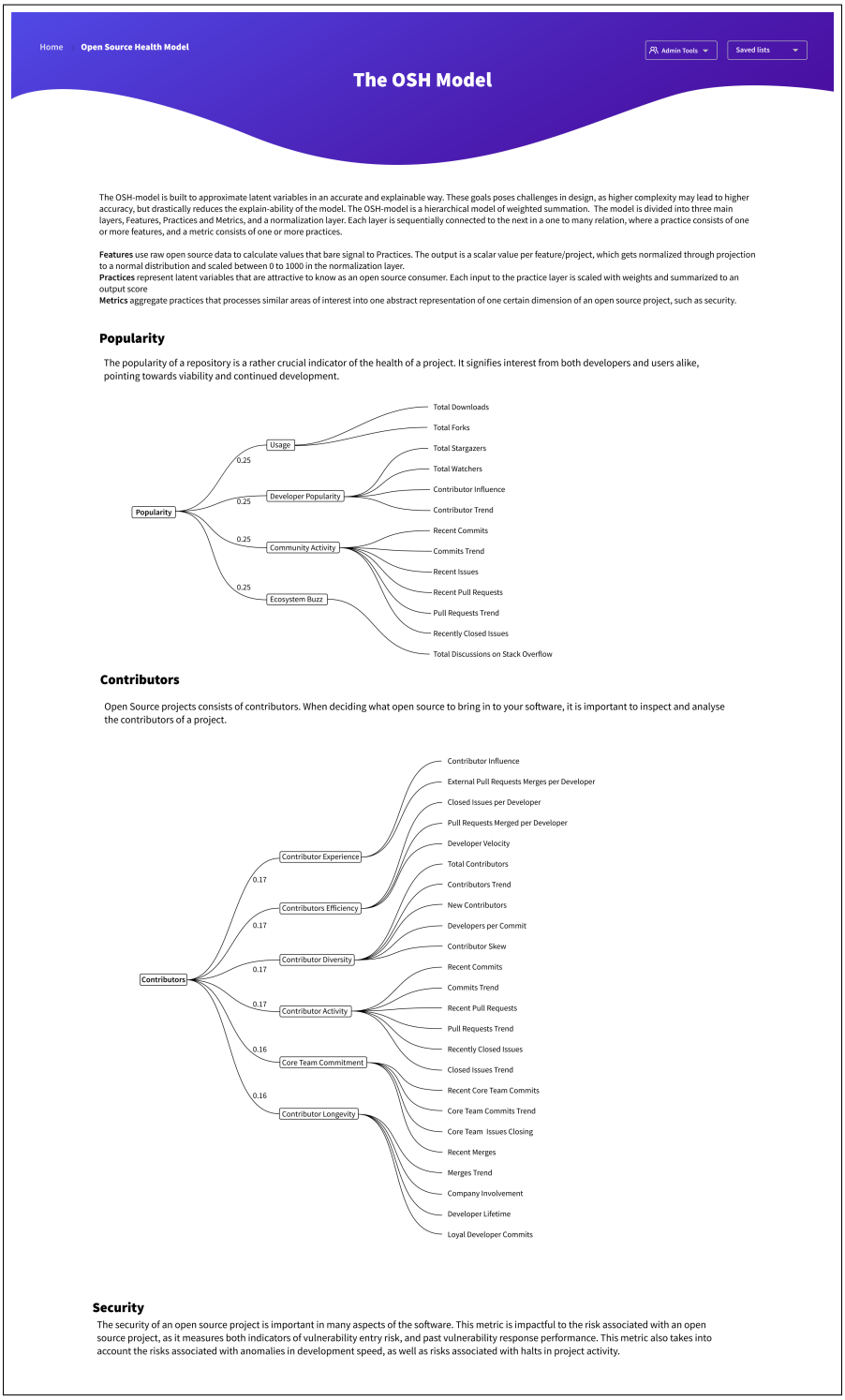


Figure 10.39: Description of the OSH model.The security model is cut off due to space on the page.

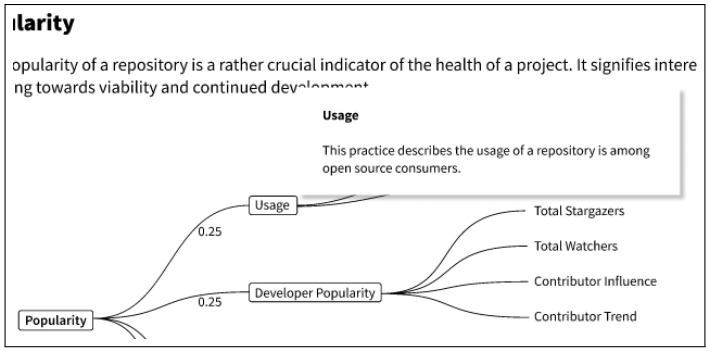


Figure 10.40: Descriptions of the practices in the OSH model.