FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Patterns for Documenting Open Source Frameworks

João Rafael da Silva Santos

Mestrado em Engenharia de Software

Supervisor: Filipe Figueiredo Correia

July 27, 2021

Patterns for Documenting Open Source Frameworks

João Rafael da Silva Santos

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

Chair: Prof. Nuno Flores
External Examiner: Prof. Eduardo Guerra
Supervisor: Prof. Filipe Figueiredo Correia

July 27, 2021

Abstract

Software documentation is an important part of the captured knowledge of a software project, and documentation patterns have often been used as a systematic way to describe good practices on software documentation. Still, many software teams are challenged by what to document, how to keep the documentation consistent, and how to make their consumers aware of the relevant documents.

The main goals of this work consist in understanding which best practices for documenting software frameworks exist, how their adoption correlates with their framework users and framework contributors, and how they relate to each other.

A literature review was done over 14 publications and identified 16 quality attributes and 114 patterns about software documentation. This knowledge was analyzed, classified, and led to the proposal of new categories and relationships between the existing patterns.

From the literature review, we carried out a pattern mining process to identify documentation patterns in real-world software projects, specifically, open source frameworks. This process resulted in the gathering of four unidentified patterns and relationships between them.

Therefore, a pattern adoption study was conducted, taking into consideration how and when previously identified patterns and newly proposed patterns are applied. This study covered ten framework patterns in five open source web frameworks projects considering metrics as such days since creation, the number of stars, and the numbers of forks.

The results gathered from our work provide a new perspective of documentation patterns for open source frameworks by revealing that adoption over time occurs in two main groups: *early adoption* and *late adoption*. The results also show us which patterns are adopted simultaneously, and which patterns already exist when a certain pattern is adopted. Ultimately, these insights can be used to guide teams in adopting software documentation practices.

Keywords: software documentation, documentation patterns, pattern adoption, documentation issues, quality attributes, documentation engineering, open source frameworks

Resumo

A documentação de *software* é uma área muito importante na captura do conhecimento de um projeto e os padrões de documentação têm sido utilizados como uma forma sistemática de descrever boas práticas na documentação de *software*. Mesmo assim, as equipas de desenvolvimento de *software* são constantemente desafiadas em saber o que documentar, como manter a documentação consistente e como manter os seus utilizadores a par de quais são os documentos mais relevantes.

Os principais objetivos deste trabalho consistem em saber que boas práticas existem para documentar *frameworks* de *software*, como a adoção dessas práticas se correlaciona com o número de utilizadores e com o número de contribuidores da *framework* e por fim, como essas práticas se relacionam entre si.

Uma revisão da literatura foi conduzida sobre 14 publicações, tendo sido identificados 16 atributos de qualidade e 114 padrões sobre documentação de *software*. Este conhecimento foi analisado e gerou uma nova visão sobre as categorias dos padrões e as relações entre os padrões existentes.

A partir desta revisão da literatura, foi conduzido um processo de extração de padrões para identificar padrões de documentação em projetos reais de *software*, especificamente em *frameworks open source*. Este processo resultou na definição de quatro novos padrões e nas relações entre eles.

De seguida foi realizado um estudo sobre a adoção de padrões, considerando como e quando os padrões anteriormente identificados e os padrões propostos são aplicados na prática. Este estudo engloba dez padrões de *frameworks* em cinco projetos distintos, considerando métricas dos projetos como os dias desde a sua criação, número de *stars* e número de *forks*.

Os resultados obtidos através do nosso trabalho fornecem uma nova perspetiva sobre padrões de documentação em *frameworks open source*, mostrando que a adoção destes padrões ao longo do tempo ocorre em dois grupos distintos: adoção prematura e adoção tardia. Estes resultados também demonstram que padrões foram adotados simultaneamente e que padrões já existiam quando um determinado padrão é adotado. Por fim, é esperado que estes resultados possam ser utilizados para conduzir as equipas de desenvolvimento de *software* na adoção destas práticas.

Keywords: documentação de software, padrões de documentação, adoção de padrões, problemas de documentação, atributos de qualidade, engenharia da documentação, frameworks open source

Acknowledgements

I would like to show my appreciation for my supervisor, Filipe Figueiredo Correia, which was an indispensable element of this work, and to whom I'm grateful for the professionalism, enthusiasm, and availability throughout the time of this thesis.

I also want to acknowledge my professors and coursemates for always raising the bar and for keeping all moments enjoyable in this journey we shared.

Thank you as well to Tomás, Pedro, and Hugo for helping me grow professionally and for encouraging me, believing in my capabilities, and trusting me in the process of balancing my work and studies.

To my dear parents, who helped me become the person I am today, and that showed me a great example of work and perseverance, and as well to my brother for all the love, support, and for providing everything I needed to pursue my goals in life.

My final words are addressed to my darling, Maria, to whom I dedicate this work and who is always there for me, celebrating during good times but also encouraging me during tough times.

João Santos

"Our thoughts make us what we are."

Dale Carnegie

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
FAQ	Frequently Asked Questions
GH	GitHub
JSON	JavaScript Object Notation
PR	Pull Request
RQ	Research Question
UI	User Interface
SPAs	Single-Page Applications
URL	Uniform Resource Locator

Chapter 1

Introduction

1.1 Context

Documentation engineering—the set of procedures needed to write and maintain artifacts of documentation—is often neglected in software projects, and it is considered tedious and time-consuming by many [4] [27]. Notwithstanding, software documentation can be an important part of the captured knowledge of a software project because it is a way to record non-structured and informal content, and it's been considered a recommended practice in software development and maintenance since its early days [28].

One goal of documentation is to help developers, testers, managers, and end-users to understand their systems and to collaborate. It facilitates communication and can be expressed in different forms, such as wikis, code annotations, user manuals, and system requirements [5].

Even when teams see it as important, projects often lack consistent, useful or even up-to-date documentation [31]. This is, to a large extent, due to the difficulty of writing and maintaining it [19].

1.2 Motivation

Framework documentation has distinctive characteristics. It's usually harder to document frameworks than other types of software because frameworks are solutions that create software, and frameworks also imply reusability and flexibility. These conditions hinder the process of explaining how to use the frameworks for users.

Considering that, many software teams are challenged by what to document, how to keep the framework documentation consistent, and how to make their consumers aware of the relevant documents. Current literature contains several catalogs on specific kinds and aspects of software documentation in general. However, they are dispersed and organized in different levels of granularity (regarding different types of documentation and audience). Adding to this, there are also few studies around the adoption of documentation patterns, much less specifically for frameworks. It is

also essential that software engineers need the information to be clear, easily identifiable, usable, and relevant [23] so that it is easier to trust and use documentation in the future.

Due to this, there is a need to comprehend which best practices are being adopted, and how the pattern adoption occurs.

This knowledge is a fundamental step to guide development teams to adopt documentation best practices and to minimize the carelessness when writing and maintaining software documentation.

1.3 Contributions

The main contributions of this work are:

- **State-of-the-Art Review** - in which we provide a comprehensive study of documentation in published works, as well as an analysis in empirical studies with patterns;
- **Pattern Mining Process** - our pattern mining process provides a new perspective of unidentified documentation patterns that are adopted in open source frameworks and can be used to guide teams in adopting these software documentation practices;
- **Pattern Adoption Study** - our study cover prospect areas on documentation pattern adoption by analyzing how their adoption evolves towards time, user usage and, user contribution.

1.4 Document Structure

This document is structured by other five chapters.

Chapter 2 details which documentation best practices exist, their relationships, and their main quality attributes. In this chapter, we also tackle empirical studies with patterns.

Chapter 3 presents the work problem and scope, the research questions that guided this work, and finally the validation methodology that we used.

Chapter 4 provides an overview of the gathered unidentified patterns, as well as the methodology we used to gather them.

Chapter 5 focuses on assessing framework patterns adoption in real-world software projects, specifically, open source frameworks.

Lastly, Chapter 6 summarizes the entire work, as well as describing future work around this topic.

Chapter 2

State of the Art

In this chapter, we conduct a literature review and a comprehensive study of documentation in published works, including, specifically, what are the quality attributes of documentation and what documentation patterns exist. Lastly, we analyze empirical studies with patterns focusing on their purpose and research methodology.

2.1 Documentation Quality Attributes

We have first surveyed the literature for documentation issues described in existing publications [12, 5, 13, 35, 19, 25, 27, 16, 38, 8, 21, 3, 31, 37], and cataloged a comprehensive list of attributes that should be taken in consideration when writing or maintaining documentation.

These quality attributes are also used in this work to relate and group existing patterns, which can be used to guide a team seeking to achieve them.

2.1.1 Methodology

We surveyed and analyzed the literature for documentation issues following these steps:

1. Collect publications that describe some kind of issue on software documentation including those that describe documentation patterns. In particular, we searched for literature using Google Scholar with keywords related to software documentation, such as *"patterns software documentation"*, *"software documentation"*, *"software documentation issues"* and *"software documentation practices"*. The most relevant literature was chosen based on title and abstract;

2. Remove duplicates in the dataset, as we have found the same issues covered in different publications.

   We found these duplicates to be expressed in different ways, as some authors describe them in more detail, and others with just a few words. For example, *"documentation of all types is frequently out of date"* [27], *"outdatedness"* [3], *"software documentation is rarely, if*

ever, updated" [16] and *"one of the highest costs of maintaining documentation for a large system is to ensure that it is kept in-sync within itself and among its related artifacts, a practice that may require continuous review"* [12] are all related with the same concern. So for this step, we have considered just one way to express the issue that would make the underlying quality attribute easier to attain;

3. Define quality attributes that should be considered when writing or maintaining documentation from the previously identified issues. For this purpose, we have converted the different expressions of issues to quality attributes that can be very useful in a certain categorization or in a future comparison.

   Some examples of expressions and their associated quality attribute are: *"organizations try to re-document their software systems, but this is a costly operation that would benefit from a clear indication of the software documents to focus on"* [13] (*cost*), *"many teams either develop too much documentation or too little documentation"* [38] (*sufficiency*) and *"lack of documentation"* [21] (*completeness*);

4. Analyze the identified quality attributes and propose relationships between them, based on the literature and on our own experience.

   For example, one work mentioned that *"it is very important that the documentation is inexpensive and easy to evolve; otherwise it won't be updated at all, becoming inconsistent, and thus negating the benefits of producing framework documentation"* [5] (*up-to-dateness* and *consistency*) and other mentioned that *"correct documentation provides accurate information in accordance with facts"* [3] (*correctness* and *accuracy*).

2.1.2 Identified Attributes

From the 14 collected publications, we found a total of 91 issues. Many of these were closely related in content, which resulted in some sort of duplication.

In order to have all the issues in the same level of knowledge, we combined some of them to have less granularity and make them easier to compare and relate to. The analysis of the *issues* led us to identify the 16 *quality attributes* that are briefly described below. A summary of which quality attributes is referenced by each publication is given in Table 2.1.

- **Accessibility.** Easy to reach and handle documentation pieces [13].
- **Accuracy.** Writing information that it's in conformity with the documentation domain [5, 3].
- **Automatability.** Having programmatic processes when writing or maintaining documentation to avoid manual human work [12].
- **Completeness.** Containing complete content on documentation provided all necessary details [5, 13, 35, 19, 38, 8, 21, 3].

- **Consistency.** Maintaining consistent uniform and coherent with the whole document [5, 35].
- **Correctness.** Including precise content into documentation artifacts [35, 27].
- **Cost.** Using an adjusted cost process when writing or maintaining the artifacts [5, 13, 27].
- **Effort.** The cognitive load that's necessary to write or maintain documentation [5].
- **Readability.** The documentation text is clear and legible [12].
- **Relevance.** Content related to the overall purpose of documentation [5, 38].
- **Standardization.** Implementing and defining standards and guidelines for teams to follow [5, 13].
- **Sufficiency.** Providing sufficient documentation to fulfil the goals of the intended reader [13, 19, 27, 31].
- **Understandability.** The documentation content is comprehensible to intended readers [5, 25].
- **Up-to-dateness.** Maintaining artifacts updated [12, 13, 35, 19, 27, 16, 38, 3, 31].
- **Usability.** Satisfaction of use of documentation in general [5, 3].
- **Usefulness.** To have utility and positive impact to the end-users [5, 27, 3].

Table 2.1: Quality attributes mentioned on each publication

Attributes	**[12]**	**[5]**	**[13]**	**[35]**	**[19]**	**[25]**	**[27]**	**[16]**	**[38]**	**[8]**	**[21]**	**[3]**	**[31]**	**[37]**
Accessibility			✓											
Accuracy		✓										✓		
Automatability	✓													
Completeness		✓	✓	✓	✓				✓	✓	✓	✓		
Consistency		✓		✓										✓
Correctness				✓			✓							
Cost		✓	✓				✓							✓
Effort		✓												
Readability							✓					✓	✓	✓
Relevance													✓	✓
Standardization		✓	✓											✓
Sufficiency			✓		✓		✓						✓	✓
Understandability		✓				✓								✓
Up-to-dateness	✓		✓	✓	✓		✓	✓	✓			✓	✓	✓
Usability		✓										✓		✓
Usefulness		✓					✓					✓		✓

2.1.3 Results

We find that some attributes are alluded to more often in the literature than others, which indicates a bigger awareness of researchers towards them. In Table 2.1, the most referenced quality attributes are *up-to-dateness* (10), *completeness* (8) and *sufficiency* (5) while the less referenced are *accessibility* (1), *automatability* (1) and *effort* (1).

We have also analyzed the relationship between the attributes based on the literature and on our own experience. We propose a set of relationships between the quality attributes, which is depicted by Figure 2.1. We believe this can provide a clearer view of what are the most pressing concerns around software documentation.

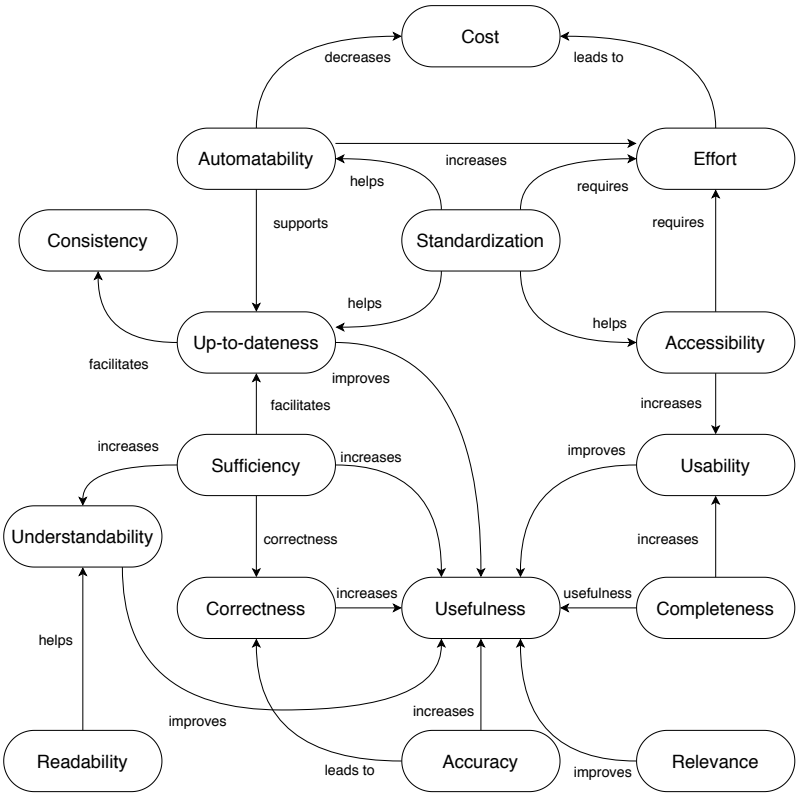

Figure 2.1: Relationships between quality attributes.

2.2 Documentation Patterns

This section has the goal of identifying which documentation patterns have been written and what are their constituent parts (*e.g.*, context, problem, solution, forces, and main quality attributes that they try to address) to build a catalog of documentation patterns, but also to help to identify differences and commonalities between them, and to capture these relationships.

2.2.1 Methodology

We surveyed the literature for software documentation patterns, and structured our analysis according to the following phases:

1. Collect works that identify any type of documentation pattern. We searched for literature using Google Scholar with terms related to software documentation such as *"patterns software documentation"*, *"software documentation"*, *"software documentation issues"* and *"software documentation practice"*. The relevant literature was chosen based on title and abstract;

2. Describe as patlets all the patterns that have been found, with a focus on the problems and solutions;

3. Identify also each pattern group and the relationships between patterns that are reported by existing literature;

4. Combine the gathered patterns into a single pattern map showing the different pattern languages and pattern categories;

5. Rearrange the patterns to form a new pattern map;

   (a) Regroup patterns by purpose;

   (b) Analyze the new pattern groups and establish relationships between the patterns within each group that had not yet been identified in the literature;

   (c) Merge similar/duplicated patterns that can be merged into one more general pattern that consists of several others;

   (d) Identify patterns that fall outside the scope of our analysis—in particular, patterns that are not specific to *software* documentation, or patterns for which we don't have enough information to conclude that they are related to the *software* documentation.

2.2.2 Identified Patterns

A total of 114 documentation patterns were identified over the 14 publications, six of which described patterns [12, 5, 25, 21, 31, 37]. We extracted from this literature several pattern characteristics, such as their problem, solution, pattern category, forces, and quality attributes. Some patterns don't share the same format, and for some, it was not possible to clearly find the context, the problem, and the pattern category. We have compiled the available information as a set of patlets, which are available in Appendix A.

The different pattern languages and categories that we have identified are shown in Figure 2.2, which depicts 14 different pattern categories and 90 relationships.

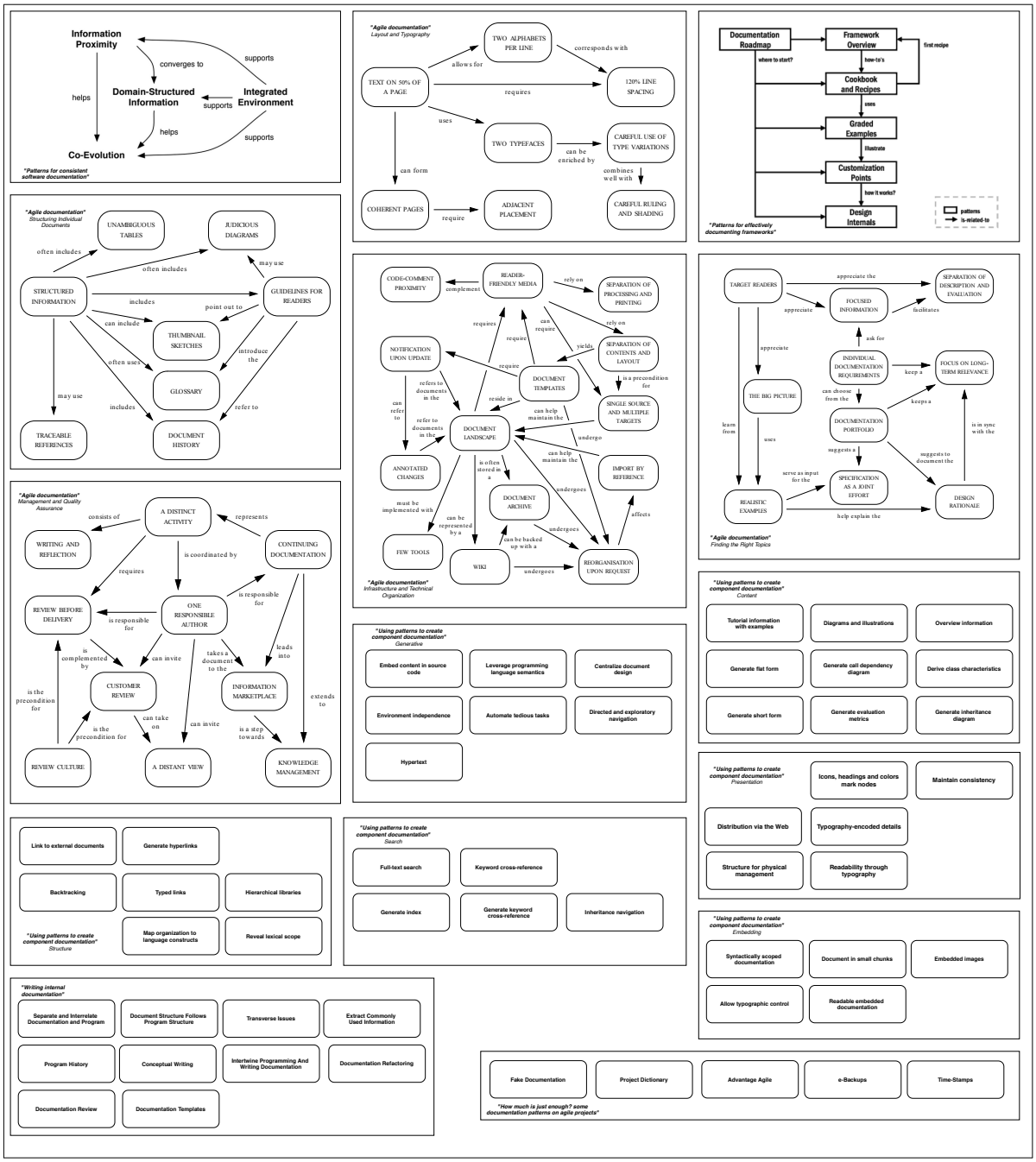

Figure 2.2: Pattern map that contains information from the extracted literature which describes 90 relationships between patterns among 14 different pattern groups.

2.2.3 Results

The process of regrouping patterns by purpose resulted in the pattern map shown in Figure 2.3. It helped us to identify 14 different groups, combining patterns from different sources, and within them, 76 relationships between the patterns.

Within these new groups we identified 16 similar patterns that we chose to merge. These are CODE-COMMENT PROXIMITY and EMBED CONTENT IN SOURCE CODE; DOCUMENTATION PORTFOLIO and DOCUMENTATION TEMPLATES; DOCUMENT TEMPLATES and SEPARATION OF CONTENTS AND LAYOUT; CONCEPTUAL WRITING, GLOSSARY and PROJECT DICTIONARY; GRADED EXAMPLES and TUTORIALS INFORMATION WITH EXAMPLES; and finally FRAMEWORK OVERVIEW, OVERVIEW INFORMATION and THE BIG PICTURE.

This new pattern map also shows the 19 patterns that were excluded from our analysis, as we considered them not to be specific enough to the domain of *software* documentation and could easily be applied to other fields.

An analysis of the number of patterns where each quality attribute is a concern rendered the result shown in Table 2.2. We can conclude that the attributes most addressed are *usefulness* (38), *accessibility* (29) and *usability* (25) and that the least are *correctness* (2) and *accuracy* (1).

Table 2.2: Relationship between quality attributes and number of times they are related with the extracted patterns

Quality Attributes	**Number of Related Patterns**
Usefulness	38
Accessibility	29
Usability	25
Consistency	23
Understandability	18
Readability	17
Relevance	15
Standardization	11
Automatability	10
Up-to-dateness	10
Effort	6
Cost	4
Completeness	4
Sufficiency	3
Correctness	2
Accuracy	1

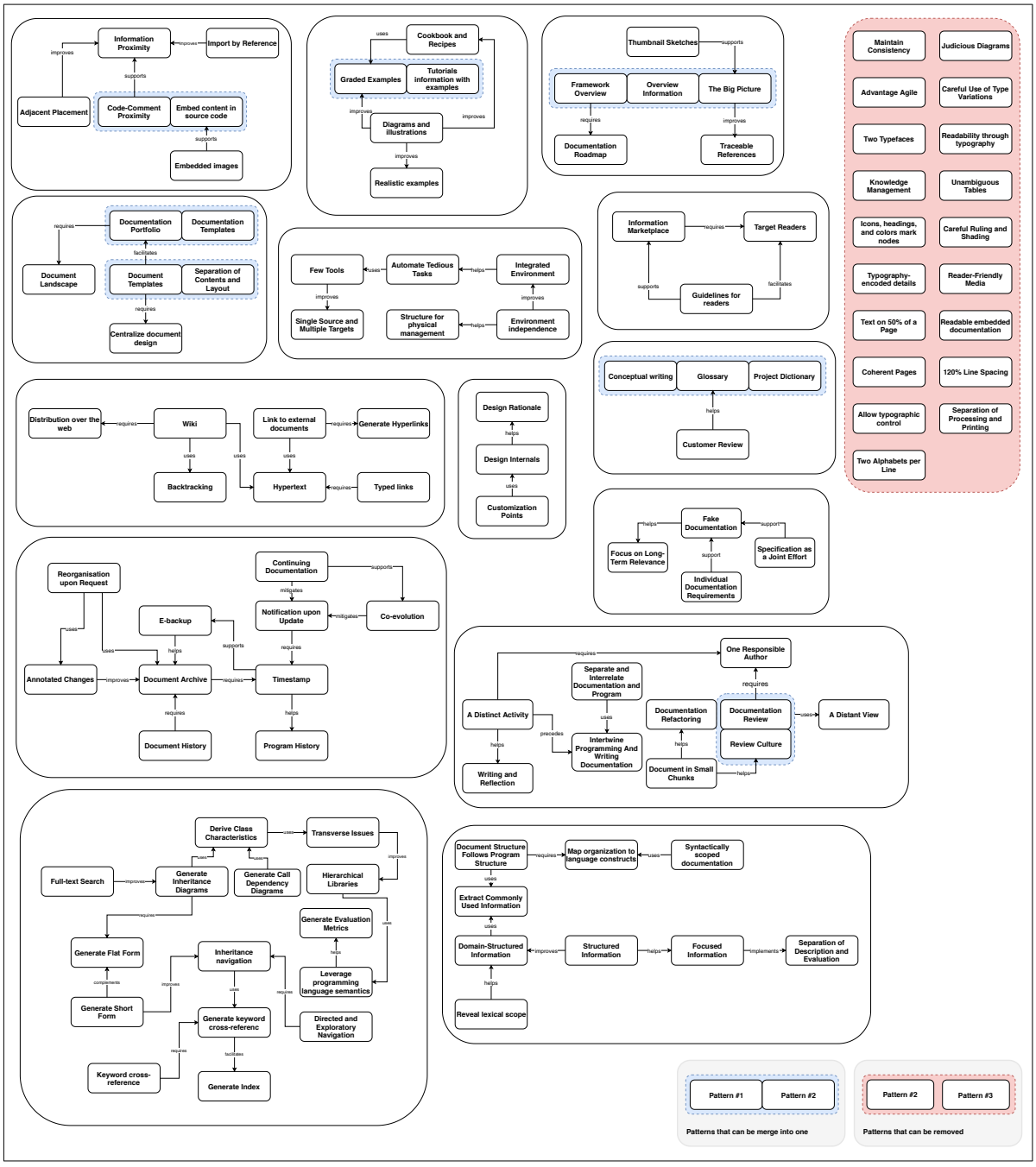

Figure 2.3: A second pattern map, providing another view over the patterns, and showing 14 rearranged groups of already existing patterns based on each pattern similarity and 76 new identified relationships between patterns. Similar patterns that can be merged into one and potential patterns to remove based on the previously defined criteria were considered as well.

2.3 Empirical Studies With Patterns

This section aims to analyze empirical studies with patterns to understand current literature on these topics, specifically how pattern adoption is studied and the existent pattern mining techniques.

2.3.1 Methodology

We surveyed and analyzed the literature for empirical studies with patterns following these steps:

1. Collect publications that describe empirical studies on software. In particular, we searched for literature using Google Scholar with keywords related to this topic, such as *"software pattern adoption"*, and *"software pattern mining"*. The relevant literature was chosen based on title and abstract;

2. Collect publications from references on previously collected publications. Also, the relevant literature was chosen based on title and abstract;

3. Analyze each study and categorize them in several attributes: *research methodology*, *study area*, and *study type*.

   Research methodology typifies the methodology used in the study (e.g. if it was conducted a survey or if it was a case study).

   Study area indicates the specific study domain like open source frameworks, machine learning, or other relevant domains.

   Lastly, *study type* described if the study was a study around pattern adoption or if it was a study to identify patterns (pattern mining).

2.3.2 Results

We analyzed seven publications that contained empirical studies with patterns, as seen in Table 2.3. From these, two were about pattern adoption, while the remaining were about pattern mining.

Table 2.3: Comparison between studies with patterns

Authors	Research Methodology	Study Area	Study Type
Iba et al. [22]	Theory Building	Patterns & Research Methods	Pattern Mining
Sousa et al. [34]	Survey	Cloud Computing	Pattern Adoption
Riehle et al. [29]	Case Study	Patterns & Research Methods	Pattern Mining
Hahsler [20]	Case Study	Object-Oriented Patterns	Pattern Adoption
Sasabe et al. [32]	Theory Building	Patterns & Research Methods	Pattern Mining
Rising [30]	Theory Building	Patterns Mining Techniques	Pattern Mining
Akado et al. [6]	Theory Building	Patterns & Research Methods	Pattern Mining

We consider studies about pattern mining very helpful to prepare us for the thinking of extracting patterns, specifically which techniques exist to extract that knowledge. However, even

considering their utility, we found these techniques very broad regarding their domain. From the five studies we analyzed around pattern mining, none of them focus on documentation patterns.

On the other hand, and regarding pattern adoption, we noticed a lack of studies approaching how and when pattern adoption occurs. Also, from this lower number of studies around pattern adoption, all of them were more specific to other contexts than software documentation as cloud computing patterns, and object-oriented patterns.

Considering this, we conclude there is a lack of studies approaching how and when pattern adoption occurs for documentation patterns, particularly in open source frameworks, which implies a need for research on these topics.

Chapter 3

Research Problem

This chapter defines the problem and scope of this work, the research questions of our study, and lastly, it addresses the validation of our methodology.

3.1 Problem and Scope

Documentation is often disregarded while developing software, however, it is undeniable that it's an important part of software development and maintenance, by recording non-structured and informal content.

Also, developers, testers, managers, and end-users benefit from documentation by understanding better their systems and how to collaborate within them.

Even considering the benefits, many projects don't have consistent, useful, or even up-to-date documentation. One reason that supports this fact, is because it subsists a difficulty when writing and maintaining documentation.

From the observation in the previous chapter, we noticed there is a significant amount of information regarding software documentation practices, but that it is dispersed and, as such, doesn't efficiently guide developers through the process of adopting them. This can be the reason for a lack of awareness of the benefits that may be gained from using these practices, and of how to use them given the specific context of each team and project.

At the same time, we acknowledge that there is limited knowledge on documentation pattern adoption, specifically how their adoption correlates to framework usage and contribution over time. To add to these challenges, we also need to tackle the possibility of existing other patterns on open source frameworks that are not defined.

3.2 Research Questions

Based on the problem we stated, we aim to address the following research questions:

- RQ1. *"What are the best practices for documenting software frameworks?"*

We want to identify good and recurring solutions to known problems that are used to write software documentation. These solutions are typically defined as patterns [7].

A portion of this work was done in the state-of-the-art review, however, we will try to find unidentified patterns that cover new types of documents being used in documentation [2] and take into consideration some recent issues found in the documentation process [3].

- RQ2. *"Does a correlation exist between the pattern adoption and the framework users and framework contributors?"*

  We intend to uncover the relationship between framework users, framework contributors, and pattern adoption over time. This relationship strives to guide software development teams adopting these practices in their projects by knowing what to document, how to keep the documentation consistent, and how to make their consumers (end-users) aware of the relevant pieces of the documentation.

- RQ3. *"Is the adoption of specific patterns related to the adoption of others?"*

  We strive to identify relationships between different patterns, finding how implementing a specific pattern affects other patterns' implementation. This relationship can potentially conduct to help documentation writers knowing when is the best occasion to adopt a certain pattern, and how the adoption of a specific pattern influences other patterns.

3.3 Methodology

Our main goal with this work is to make a comprehensive study of documentation in real-world software development focused on open source framework projects. We start with the **state-of-the-art review** to known which best practices exist and how they are adopted. From this review, we went to the **pattern mining** process to identify patterns that are present in the documentation but not yet identified. From the **pattern mining**, we carry out a validation on the **pattern adoption** to understand how framework documentation patterns (the ones gathered in the **state-of-the-art review** and in the **pattern mining** process) evolve towards time, user usage and, user contribution.

Chapter 4

Pattern Mining

In this chapter, we present the followed methodology to collect unidentified patterns, the patterns overview, and the gathered patterns.

4.1 Methodology

We conducted a pattern mining process on real software projects to uncover patterns that are present in the documentation but are not yet described as patterns.

We started by selecting the most popular open source framework projects following well-defined criteria. Then we defined a set of procedures to analyze documentation uniformly. Lastly, we defined a structure for each gathered pattern.

4.1.1 Sampling

The selection of the projects followed the criterion below:

- Selection of the most popular web frameworks from the Stack Overflow Developer Survey of 2020[1].

  We targeted projects which had high documentation usage from their users. For that, we selected the most popular projects that are projects utilized by more users [9] [14]. Their popularity also reflects the existence of more documentation and higher evolution of documentation [40] [1]. Finally, the main purpose of documentation for frameworks is implementation, and the usage of documentation for that purpose is higher than other purposes like maintenance [18].

  We elected StackOverflow as the survey source because it's a reference platform in the software engineering industry [39] [36]. This survey contains almost 65,000 answers which represents a diversity of developers worldwide and their technology preferences;

[1] Full survey available at https://insights.stackoverflow.com/survey/2020

- Repositories where we could gather complete metrics over the entire project life-cycle. This condition occurs because our data source doesn't provide enough metrics for older projects.

Table 4.1 represents the most popular web frameworks, as well as the eligible projects, which are highlighted: *Angular*, *ASP.NET Core*, *Gatsby*, *React.js*, and *Vue.js*.

Table 4.1: Most popular web frameworks (StackOverflow survey 2020)

Project	**Created at**
jQuery	2009-04-03
React.js	**2013-05-24**
Angular	**2014-09-18**
ASP.NET	2002-01-05
Express	2009-06-26
ASP.NET Core	**2014-03-11**
Vue.js	**2013-07-29**
Spring	2010-12-08
Angular.js	2010-01-06
Django	2012-04-28
Flask	2010-04-06
Laravel	2011-06-08
Ruby on Rails	2008-04-11
Symfony	2010-01-04
Gatsby	**2015-05-21**
Drupal	2009-01-04

4.1.2 Procedures

To start the pattern mining process, we needed a strategy to tackle unidentified patterns.

We know that defining a well-defined systematic process on uncertain topics can be a hard task. Based on that, we essentially applied our **personal experience** in using and maintaining open source frameworks to find recurring solutions for recurring problems. However, not to start without any idea where to start looking into, we researched literature to consider **new types of documents** being used in documentation [2] and also some **recent issues** found in the documentation process [3]. This was the starting point for the pattern mining process.

4.1.3 Pattern Form

To achieve uniformity when presenting all patterns, we needed to define a general structure for each pattern to follow.

Several authors define their patterns considering different formats. We combined all that knowledge [7, 17, 10, 33, 11, 15] in order to have an adequate format to present the identified patterns.

The identified patterns contain the following sections:

- **Name.** This section defines the pattern name, which can indicate their overall purpose.
- **Context.** This section describes the environment where the pattern is applied and where the problem occurs.
- **Problem.** This section begins by defining an overview of the problem, and it's followed by a more detailed description.
- **Forces.** This section presents the overall pattern forces that address pattern problems and complement the pattern solution.
- **Solution.** This section starts by presenting an initial statement of the pattern solution. Then it goes through the solution with more details.
- **Related Patterns.** This section explains which patterns relate with the one being described and how that relationship proceeds.
- **Known Uses.** This section details real-world examples of where the pattern is adopted.

4.2 Patterns Overview

We identified four different patterns: CONTRIBUTION GUIDELINES, DOCUMENTATION VERSIONING, MIGRATION HANDBOOK, and MULTI-LANGUAGE SUPPORT.

These patterns are related to each other in several ways, and their relationships are depicted in Figure 4.1.

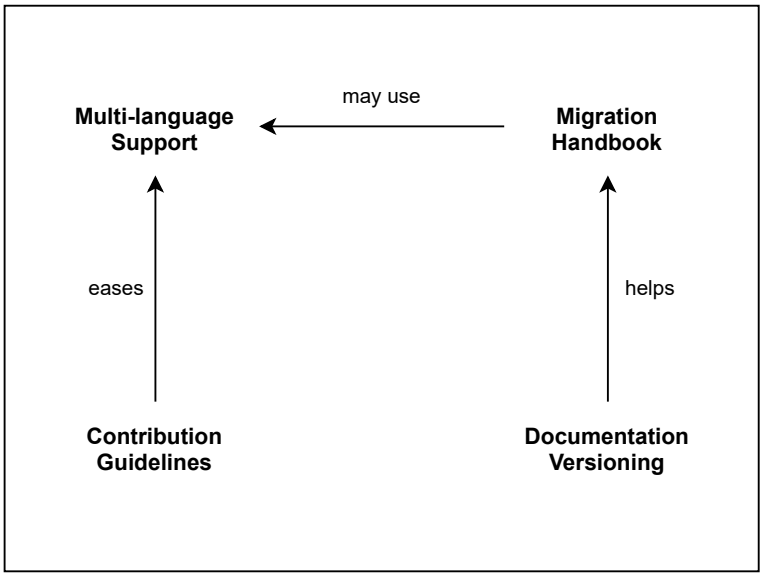

Figure 4.1: Identified patterns and their relationships

Contribution is an essential part of the open source frameworks, and project maintainers should encourage people to contribute to their projects. CONTRIBUTION GUIDELINES facilitates new contributions by giving relevant information on how contribution runs for that particular project.

With software evolution, old previous documentation versions should become accessible for framework users who still use those versions. DOCUMENTATION VERSIONING allows that by providing an organization over different documentation versions.

New software versions can introduce changes on how to implement framework features, creating difficulties for those who want to upgrade their old versions. MIGRATION HANDBOOK handles those challenges by informing framework users on how to operate these changes.

Language barrier should not demoralize frameworks users that don't speak the language in which documentation is written. MULTI-LANGUAGE SUPPORT eases that barrier by offering documentation content in different languages.

4.3 CONTRIBUTION GUIDELINES

As a project evolves in size and complexity, more users are needed to contribute to that open-source project. Guidelines should be present to help users in this process.

4.3.1 Problem

How can users start contributing to a project and know which practices are being used by the current team?

When reading the documentation for the first time, users that want to contribute want to know how they can do it for that project. Specifically, they might know the system internals, how to report bugs, and which style guide the team implements.

4.3.2 Forces

- **Standardization.** Often contributors can't find guidance on how they should contribute to projects. Considering this, a single source of truth should be available.
- **Sufficiency.** Instructions should be enough to solve contributors' goals otherwise they will not contribute to the project.

4.3.3 Solution

Provide guidelines on how to contribute to the project and which best practices the team follows.

Start by providing information on how a user can contribute to the project, specifying conventions the team is following, how to report bugs, and how to submit changes.

It should point to an overview of the system to let the user know, at an early level, the main building blocks of the system.

4.3.4 Related Patterns

- Guidelines should describe the design principles of the system. Using THE BIG PICTURE [31] allows readers to understand the overall system architecture.
- Authors want to help contributors on how to understand better their systems. Considering that, DESIGN INTERNALS [5] is used to describe the fundamental layers behind the framework, and potentially the areas where customization is possible.
- Often, contributors don't know how they should read the guidelines, considering that we may use the GUIDELINES FOR READERS [31] to introduce an initial overview towards documents that help contributors facing these documents.
- Know how to reach this section is also important for users. DOCUMENTATION ROADMAP [5] helps by organizing all documentation sections, and in particular including this contributing section in their organization.

4.3.5 Known Uses

- **React.js** implements CONTRIBUTION GUIDELINES by containing one main page that includes several explanations on how to contribute to the code and documentation, their design principles, code of conduct, versioning policy, and style guide.
- **Gatsby** uses this pattern by providing one main page with six different sections that explain their contribution process to code, blog, and documentation, followed by their brand guidelines, and it includes a description of the community behind the project.
- **Angular** adopts CONTRIBUTION GUIDELINES by providing a page with all Angular projects. The main platform link redirects to a markdown file in GitHub. The file contains information about the code of conduct, how to report bugs, their submission guidelines, the review process, and the code style.

4.4 DOCUMENTATION VERSIONING

With the launching of new versions of the software, documentation usually is updated. However, not all frameworks users use the last framework version, some use previous versions.

4.4.1 Problem

How can users access previous versions of the documentation?

While new versions of your framework are launched, many users still use old versions. Those users should not face only newer documentation versions. They should be able to search through previous documentation that reflects previous software versions of the framework that they are still using.

4.4.2 Forces

- **Accessibility.** Previous versions should be easily searchable, so users can navigate more smoothly between different versions.
- **Up-to-dateness.** Artifacts of previous versions most of the time are not updated while new fixes arrive.

4.4.3 Solution

Provide an organization over different documentation versions, that allow the users to easily navigate between those versions.

Begin by implementing a section where users can find previous versions of your framework, as well as links that redirect to those versions. Those versions can be separated into a whole new page or they can be listed in a dropdown placed near the artifact that displays the document's organization.

4.4.4 Related Patterns

- The organization of different versions requires a history of these previous versions, for that this pattern may be applied after DOCUMENT HISTORY [31] is applied.
- DOCUMENTATION ROADMAP [5] helps DOCUMENTATION VERSIONING by including an option to change between documentation versions, or to indicate the documentation version which the user is reading.
- This pattern also supports DOCUMENT ARCHIVE [31] by providing an organization that allows navigation through previous versions.

4.4.5 Known Uses

- **React.js** adopts DOCUMENTATION VERSIONING by providing a list of previous versions as well as their changelogs and snapshots.
- **Vue.js** implements this pattern by presenting a dropdown menu with previous versions as values. Each change of the dropdown redirects to the documentation version page.
- **Angular** also adopts this pattern by showing a dropdown menu with their previous versions. A redirect occurs after changing the dropdown value.

4.5 MIGRATION HANDBOOK

Frameworks evolve naturally over time. With the increase of their use, more features likely become available with new versions. At the same time, it's possible that breaking changes happen

because early versions of open source projects are not mature enough and don't have a clear and well-defined architecture.

4.5.1 Problem

How can users easily migrate their systems that use an outdated version of the framework to use the latest version of the same framework?

Users who use older versions of the framework may want to update due to security reasons or because the new version includes more features. However, migrating from one older version to a more recent one can be a hard task to operate without guidance.

4.5.2 Forces

- **Accessibility.** Migration documents sometimes are not easy to find and access for their users.
- **Effort.** Migration should not be a hard to task to perform, otherwise users will not update their systems.
- **Relevance.** Contents should be as much as necessary, so users can migrate their frameworks versions.

4.5.3 Solution

Provide a guide containing the new features and breaking changes the new version delivers.

Start by presenting a list of changes and deprecations. These changes should include code snippets of how it was done in the previous version and how it's done now in the most recent version, so users can understand better those changes with examples.

You can complement each change or deprecation by including a motivation behind that decision and possibly a link to the original PR that affected that feature.

4.5.4 Related Patterns

- Users who use outdated versions can be notified of a significant change on the document. Considering that, NOTIFICATION UPON UPDATE [31] can help this pattern by providing navigation to the migration documents that allows a version upgrade.
- DOCUMENTATION VERSIONING helps MIGRATION HANDBOOK by referring changes and deprecations to that specific previous version. For instance, while a user finds a particular change in a recent version it may be helpful to navigate to that version to see more API details.
- This pattern may use MULTI-LANGUAGE SUPPORT to facilitate migration for users who don't speak the original language in which the documentation was written.

4.5.5 Known Uses

- **Vue.js** applies this pattern containing two main parts: the first is a FAQ section approaching relevant questions to their users; the second describes API changes with code snippets to help users to fulfill their goals.

- **Angular** adopts MIGRATION HANDBOOK describing an initial overview regarding the migration, followed by the changes and deprecations of the newer version. Each change is detailed, providing a link to the PR where that change occurred.

- **ASP.NET Core** implements this pattern by containing different versions that can be migrated, followed by the migration prerequisites. Additionally, they provide a detailed explanation for each feature, containing the old behavior, the new behavior, and in some cases, the motivation behind that decision.

4.6 MULTI-LANGUAGE SUPPORT

Frameworks that grow in usage tend to have a large user base that doesn't speak the language in which the documentation is written.

4.6.1 Problem

How to remove language barriers from users who don't speak the language in which the documentation is written?

Software engineering is not an easy domain to master, so why should we increase this difficulty by having language barriers? The frameworks users should be comfortable reading artifacts, however, not everyone speaks the language in which documentation is written.

4.6.2 Forces

- **Readability.** Translations must be clear and easy to read.

- **Understandability.** Documentation should be understood for all audiences, despite their native language.

4.6.3 Solution

Produce and distribute documentation artifacts in more than one language. Preferentially in languages that are spoken by your users.

Start by analyzing your audience and the languages they speak. Translate artifacts into those languages. After, provide a list with all languages your documentation is translated and inform your users of these new additions.

4.6.4 Related Patterns

- Translated documents can be hard to found by users. For that, this pattern can be supported by INFORMATION MARKETPLACE [31] letting the users know that exists a translation in their native language for that document.
- Writing the same document with different languages can be a complex task. To ease that, DOCUMENTATION TEMPLATES [37] can help this pattern by providing templates which only vary in language.
- CONTRIBUTION GUIDELINES eases MULTI-LANGUAGE SUPPORT by emphasizing best practices for contributors to follow when writing documentation, particularly on how they should translate documents.

4.6.5 Known Uses

- **ASP.NET Core** uses this pattern, offering a possibility in the footer to change the language in which the documentation is translated. This language change is done on an external page that redirects to the translated documentation page.
- **Angular** implements MULTI-LANGUAGE SUPPORT by offering all languages available in the footer. The selection of the new language will redirect the user to the translated page.
- **React.js** adopts this pattern by providing a link in their menu to the translations page. That page contains all available translations that are located individually on a new page.

Chapter 5

Pattern Adoption

This chapter defines the study methodology, following by how documentation is handled for real-world software projects, taking into consideration how and when previously identified patterns and newly proposed patterns are applied. Finally, we describe the threats to validity of this study.

5.1 Methodology

We conducted an empirical study on real software projects to understand how pattern adoption correlates to framework usage and contribution over time, and how implementing a specific pattern correlates with the implementation of others.

We started by selecting a few open source frameworks among the most popular, following well-defined criteria. Then we defined a set of procedures that allows to compare documentation uniformly. This consistency is needed so the results between different projects and between different data points of the same project can lead to clarify and answer the previously defined research questions.

These procedures explain the process behind the metrics extraction, the data points selection for each project, and the criteria for each pattern when analyzing the documentation.

5.1.1 Variable Identification

Our evaluation is focused to assess pattern adoption by open source frameworks. Therefore, the dependent variable of our study is inevitably the **pattern adoption**.

To help define relationships, we decided to extract useful metrics from the GitHub project repository. From these extracted metrics we considered three independent variables: **days since creation**, **number of stars**, and **number of forks**.

The **days since creation**, as its name suggests, is a metric that indicates the number of days since the creation of a project. This evidence can help us understand if a pattern or a group of patterns appears more early or later in the project life cycle. It can be helpful, as well, to potentially categorize patterns.

The projects we will analyze are all available in GitHub and this platform has two distinct concepts: the **number of stars** and the **number of forks**.

The **number of stars** is a measure of awareness regarding a repository. That means a project with a higher number of stars is more prone to be known to users than a project with a lower number of stars. Stars can also be seen as a metric of appreciation towards the work. Considering this, we can define stars as a metric of people that use the framework (*users*) or people that potentially will use the framework (*potential users*).

The **number of forks** indicates the number of copies that a repository has. This copy lets people make changes without affecting the original repository, and it's advised when a contributor wants to start a contribution to a project. Recognizing this, we can define forks as a metric of people that contribute to the framework (*contributors*) or people that potentially will contribute to the framework (*potential contributors*).

5.1.2 Sampling

The selection of the projects followed the same criterion we used for the pattern mining process as described in Section 4.1.1.

5.1.3 Data Collection and Analysis

To extract the metrics over the selected repositories, we used an open source tool called GH Archive[1]. The data is stored in JSON format and contains all the events[2] for a certain hour and day (given in the URL).

As we wanted all the events from the beginning of the repository until the end of the project analysis (30th of April 2021), we created a helper tool[3] to support the data mining, data cleaning, and data visualization processes.

These processes consisted of different steps:

1. Retrieves all events during the entire repository life-cycle with a step of three months;
2. Standardizes the data in a normalized format to be later analyzed by a data visualization tool;
3. Calculates the interpolation for the days that don't contain events. If a specific day doesn't contain events we download the day immediately before and the day immediately after;
4. Joins the data in a unique file per repository with all the days (step of three months). Each day structure can be seen in Figure 5.1;

[1] GH Archive is a tool that records the public GitHub API over the years and provides the data in JSON format to facilitate analysis of all the gathered data. This tool is available at https://www.gharchive.org/

[2] GitHub Events are specific actions performed in a repository. All event types can be seen at https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types

[3] Repository available at https://github.com/joaorafaelsantos/msc-thesis

```
{
  "size": 154388,
  "stargazers": 149194,
  "forks": 28984,
  "open_issues": 600,
  "date": "2020-05-24"
}
```

Figure 5.1: JSON data structure for React.js on the 24th of May 2020

5. Reads all this data and outputs a visualization of these metrics. This visualization can be per project or through the combination of all five projects in one plot. The stars and forks evolution for all projects can be seen in Figures 5.2 and 5.3.

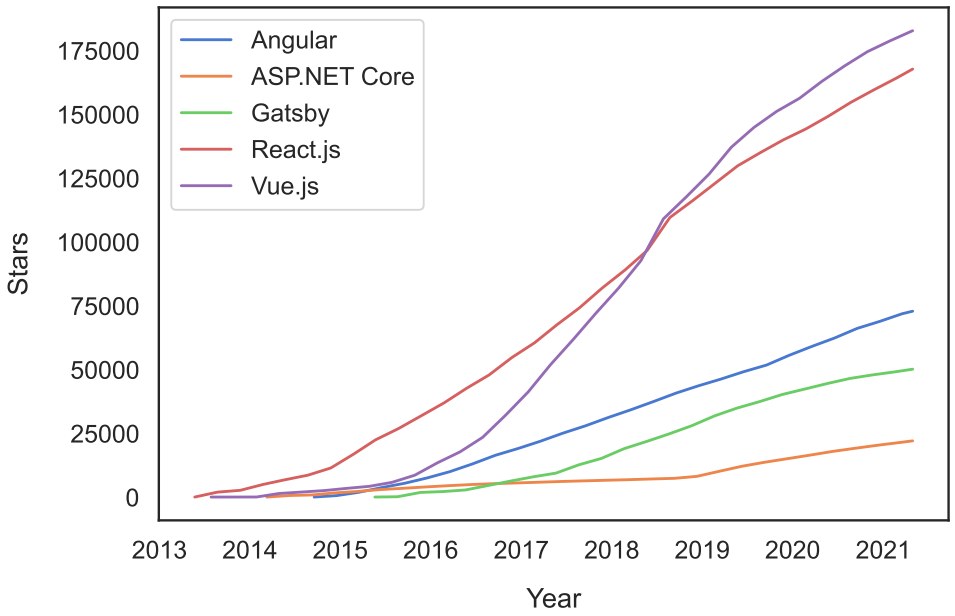

Figure 5.2: Stars evolution for all five projects

We believe this data could help further investigations over this topic, and to help researchers follow all these steps, we made this tool available on the GitHub platform as a public repository.

After selecting the projects and knowing that these projects have a vast number of documentation versions since their creation we needed to define which documentation versions will be analyzed. From now on, we will refer to documentation versions as data points.

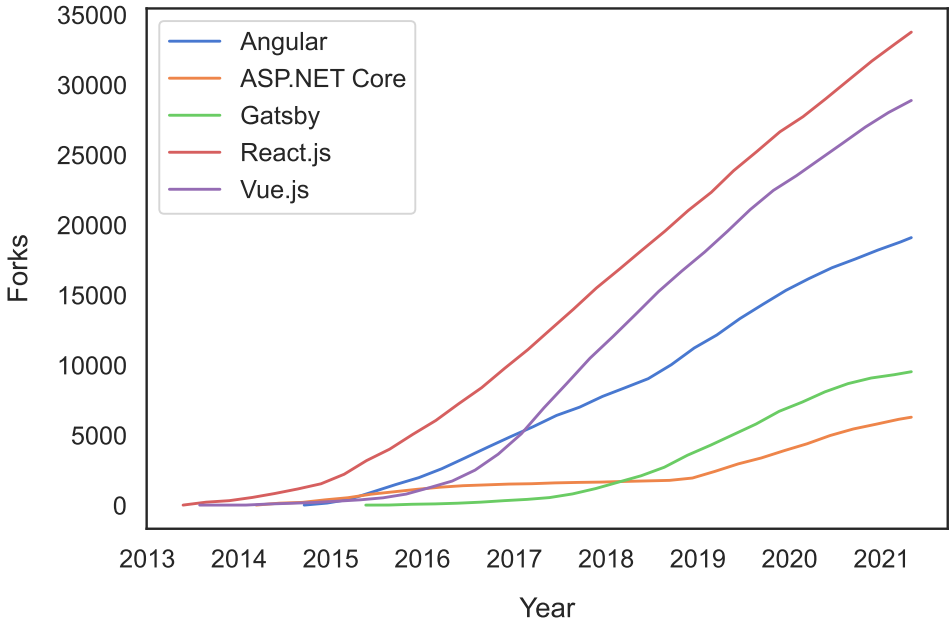

Figure 5.3: Forks evolution for all five projects

The data points for a certain project can be gathered from different sources. We considered three different sources: Wayback Machine [4], the project website, and the project repository.

Wayback Machine is our preferred format to download documentation snapshots for certain data because it provides a large available database. The second is the snapshots provided on the project website and, the last option is to gather documentation from the official GitHub repository. To clarify, these last two are different types of documentation.

Instead of selecting specific dates on the entire project life cycle, we opted to choose, as data points to analyze, the ones where documentation patterns are adopted for the first time. This adoption is unknown, so we applied a binary search algorithm to help understanding when the pattern adoption occurs.

It's important to mention that this analysis requires a considerable amount of time. It's a manual process where we need to validate each data point of every project to find the pattern characteristics. Considering that, the amount of time increases by adding more projects which contain several data points.

The binary search algorithm is an efficient search algorithm to find an element within a sorted list. With binary search, we start by comparing the middle element of that list with the target value. For that, we consider two initial boundaries: lower bound and upper bound that in the beginning equals to the list first and last values. If the middle element doesn't match the target value, we exclude the half where the value doesn't fit, and we continue the search on the other half until we find the target value.

[4] Wayback Machine is a website maintained by the Internet Archive that stores web pages over time and is available at https://web.archive.org/

Our main goal with this algorithm is to know if a pattern adoption occurs at a specific time. To achieve that goal, we performed steps:

- In the lower bound we considered the first snapshot available (which varies from project to project) and in the upper bound we took the snapshot available on the 30th of April 2021. The documentation for these limits is the first to be analyzed;
- Then like the original implementation, we found the middle element (in days);
- If the adoption occurs and the distance between each boundary and the middle element is less than three months, we recognized the pattern adoption to happen at that data point;
- If the adoption doesn't occur or the distance is bigger than three months, we continue the search process;
- The search can happen in both directions (left and right) because the only elimination factor is the distance being lower than three months. This factor also means the dates we provide for the pattern adoption have an error margin of three months. We also considered this timeframe to increase the possibility to have significant differences between documentation. For instance, versions that are separated by two weeks, typically will not contain many differences between each other;
- It is also important to mention that a middle element is adjusted if a snapshot for a data point doesn't exist. In this case, we search for the nearest data point, preferentially the next day. If an adjustment for the data point occurs and if needed, we adjust the limits (lower and upper bound) as well.

We can recognize our approach as a variant of the binary search algorithm, where the main advantages of applying this technique are: knowing when a specific pattern is adopted with a potentially lower error margin and reducing the number of data points that are required to analyze for each project.

Considered all five projects combined, we gathered 52 data points to be analyzed in Chapter 5 as detailed in Figure 5.4.

After defining which data points we will analyze, we needed to clarify how we are analyzing these data points, particularly which patterns and which characteristics we will verify when analyzing the documentation.

From the previously identified patterns in the literature, we selected six framework patterns [5] of the 114 total identified patterns. We decided to choose these patterns because they are related to frameworks, which is the domain of the documentation we want to analyze.

For each data point, we analyzed the available documentation to find patterns based on specific criteria that vary for each pattern.

We defined these criteria from an analysis of the context, problem, and solution proposed by the pattern author.

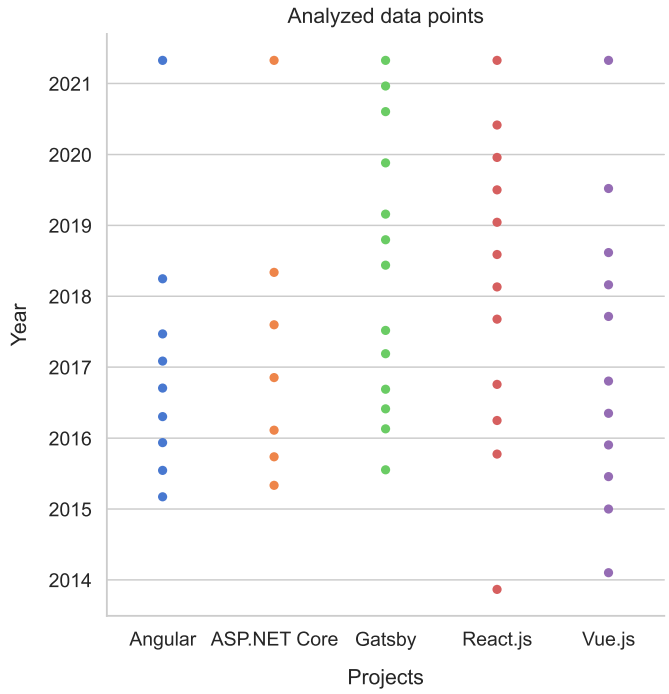

Figure 5.4: Data points to be analyzed for all projects

"**Cookbook & Recipes** explains how to provide readers with information that explain how-to-use the framework to solve specific problems of application development, and how to combine this prescriptive information with small amounts of descriptive information to help users on understanding what they are doing." [5]

- Has a collection of recipes.
- Each recipe should describe one framework customization.
- The cookbook performs as a guide, organizing all the recipes.

"**Customization Points** describes how to provide readers with task-oriented information with more precision and more design detail than cookbooks and recipes, so that readers can quickly identify the customizable points of the framework (hot-spots) and to understand how they are supported (hooks)." [5]

- Provides a list of customization points (hot-spots).
- Each hot-spot should describe the hooks it provides and the hot-spot subsystem that implements.
- Organized by framework functionality or by frameworks parts and modules.

"**Design Internals** explains how to provide detailed information about what can be adapted and how the adaptation is supported, by referring the patterns that are used in its implementation and where they are instantiated." [5]

- Has details about the design internals.
- Includes explanation on how hot-spots handles configuration.

"**DOCUMENTATION ROADMAP** helps on deciding what to include in the documentation overview to provide readers of different audiences with useful and effective hints on what to read to acquire the knowledge they are looking for." [5]

- There is a main entry point of the documentation.
- Provides an organization of the documentation.
- Facilitates navigation to topics (as links).
- It's organized by audience and/or kind of uses (beginners or even more advanced users).

"**FRAMEWORK OVERVIEW** suggests providing introductory information, in the form of a framework overview, briefly describing the domain, the scope of the framework, and the flexibility offered, because contextual information about the framework is the first kind of information that a framework user needs." [5]

- Explains the covered domain, the purpose of the framework, and their main documents.
- Has references to other artifacts with more detailed information.

"**GRADED EXAMPLES** describes how to provide and organize example applications constructed with the framework and how to cross-reference them with the other kinds of artefacts (cookbooks, patterns, and source code)." [5]

- Provides examples from simple to more complex ones.
- Each one shows a different way of customization from the previous one.
- The examples should grow in complexity.

5.2 Results Analysis

We start by analyzing pattern adoption considering days since the project was created, followed by the number of stars and forks. Lastly, we relate all analyzed patterns with each other to find uncovered relationships.

5.2.1 Days Since Creation

Figures 5.5 and 5.6 depicts the number of projects who adopt a specific pattern and the number of patterns adopted in an early phase of the project (first data point available for each project) and in an advanced phase of the project (data point analyzed on 30th of April 2021).

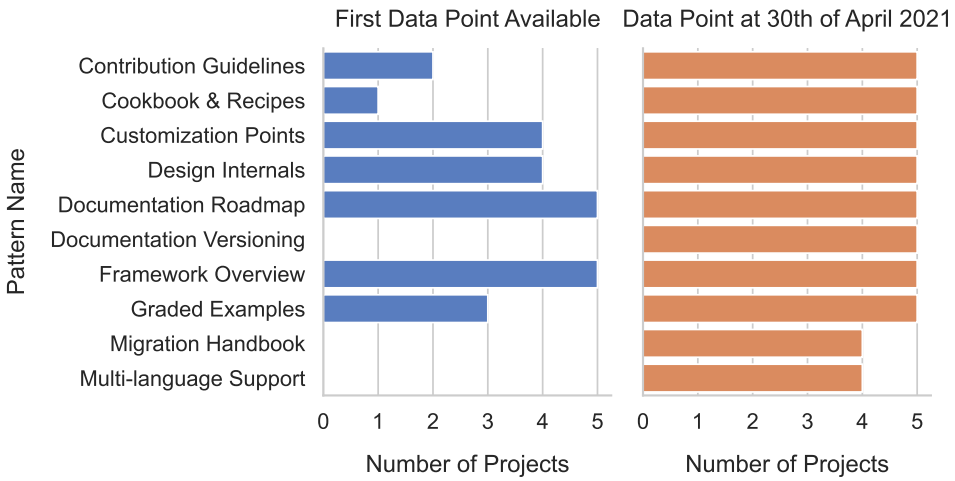

Figure 5.5: Number of projects that adopt a specific pattern at the first data point available and at 30th of April 2021

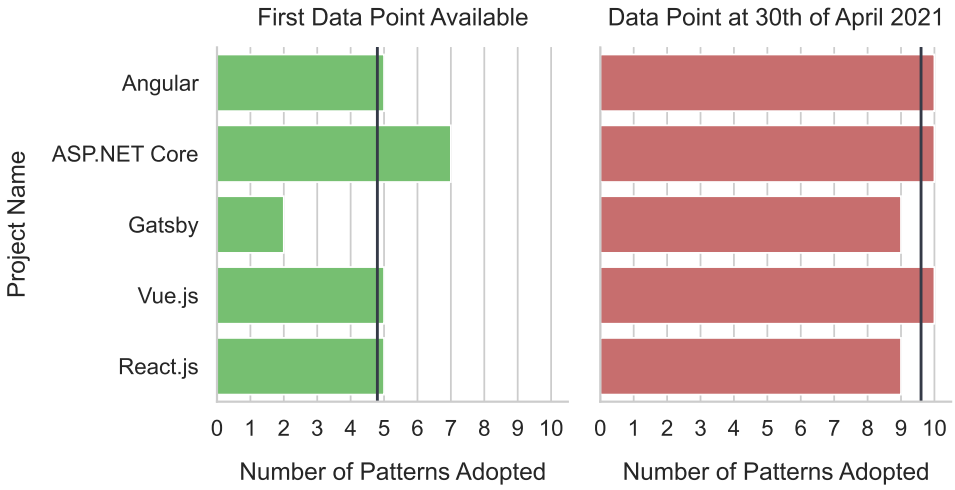

Figure 5.6: Number of patterns adopted at the first data point available and at 30th of April 2021

From this data we observe that:

- At the beginning of a project, the **mean pattern adoption per project** is 4.8 out of 10 patterns, only DOCUMENTATION ROADMAP and FRAMEWORK OVERVIEW are adopted for all projects;
- At an advanced phase of the project, the **mean pattern adoption per project** is 9.6 out of 10 patterns, and with exception to MIGRATION HANDBOOK and MULTI-LANGUAGE SUPPORT, the others patterns are adopted for all projects;
- As expected, we can see the increasing adoption of new patterns over time. This increase is justified because software usually evolves [26] and documentation must reflect these

changes [24]. Lastly, patterns are adopted to solve recurring problems that can appear from these changes.

The adoption in an early phase of the project (first data point available) and in an advanced phase of the project (data point at 30th of April 2021) doesn't reflect how the adoption evolves, so Figure 5.7 details the evolution of the adoption for each pattern over time and in Figure 5.8 we can see the pattern adoption per project and the average day since project creation when each pattern is adopted.

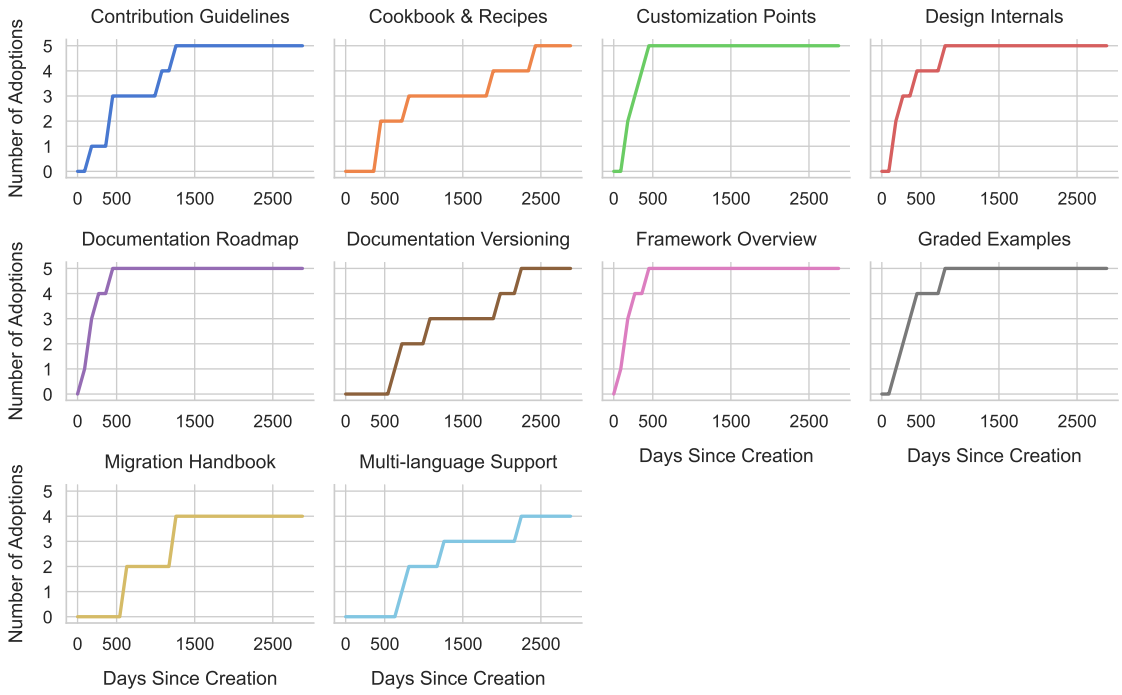

Figure 5.7: Overall pattern adoption over time

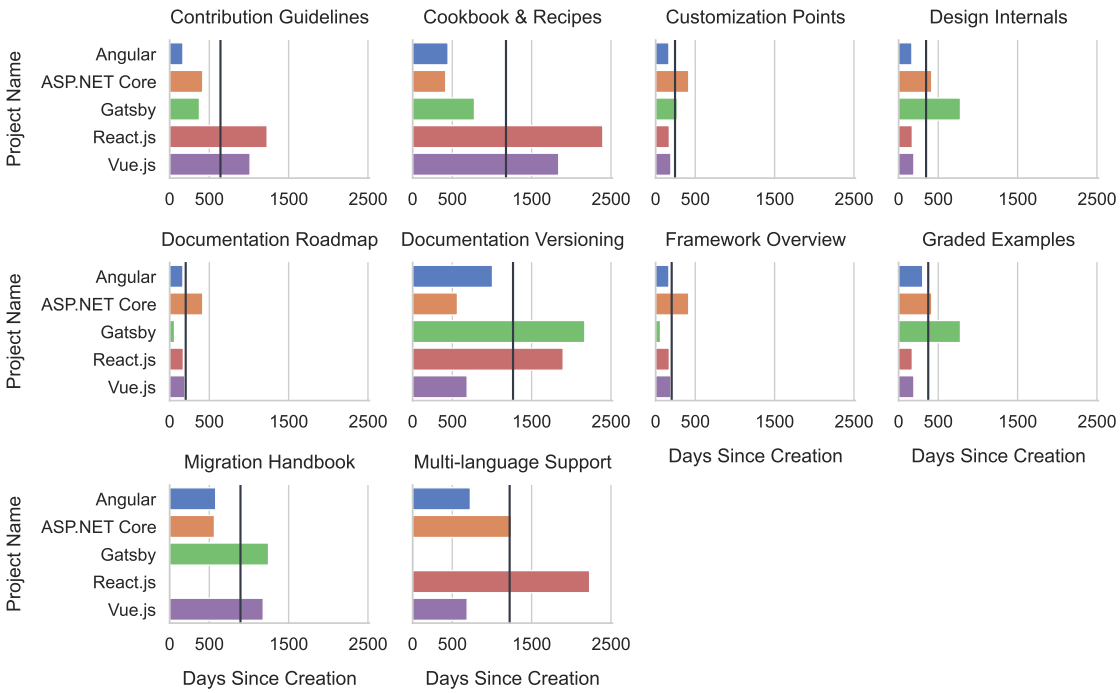

Figure 5.8: Pattern adoption per project considering days since creation. Vertical lines represent the average day when each pattern is adopted.

On this data we visualize that:

- Unsurprisingly the patterns to have an **early adoption** (normally adopted within the first two years) are: CUSTOMIZATION POINTS, DESIGN INTERNALS, DOCUMENTATION ROADMAP, FRAMEWORK OVERVIEW, and GRADED EXAMPLES.

  DOCUMENTATION ROADMAP and FRAMEWORK OVERVIEW are related to the organization and searching of the documentation. CUSTOMIZATION POINTS, DESIGN INTERNALS and GRADED EXAMPLES describe the usage and customization of the framework that helps users to use and customize the framework.

- The patterns DOCUMENTATION ROADMAP and FRAMEWORK OVERVIEW usually are the firsts to be used in the documentation, and they are the only two patterns to be present at the beginning of the documentation for all the projects we analyzed;

- On the other hand, the patterns that have a **late adoption** (usually after the first year and a half) are: DOCUMENTATION VERSIONING, MIGRATION HANDBOOK and MULTI-LANGUAGE SUPPORT.

  DOCUMENTATION VERSIONING and MIGRATION HANDBOOK presume there is more than one version of the framework and that framework versions increase over time, so there is no need to adopt these patterns earlier.

Lastly, MULTI-LANGUAGE SUPPORT requires a large user base that justifies the effort in the translation of documentation. We also acknowledge that, at the beginning of the project, the project resources are not abundant to implement translations for the entire documentation;

- The patterns CONTRIBUTION GUIDELINES and COOKBOOK & RECIPES are the only patterns that don't fit in any group.

  React.js and *Vue.js* adopt CONTRIBUTION GUIDELINES in an advanced phase, while *Angular*, *ASP.NET Core*, and *Gatsby* adopt it in an earlier phase.

  To understand this phenomenon, we analyzed *React.js* and *Vue.js* GitHub repositories instead of their websites, and we found pieces of evidence that show the existence of this pattern at the beginning of the project. In their repositories, we could see one markdown file with guidance towards project contribution.

  The decision of including these guidelines on the website seems to be a team decision rather than a necessity of the project. For this reason, we could potentially fit this pattern into the group of those patterns that are adopted early.

  Regarding COOKBOOK & RECIPES there are two projects that adopt it in an earlier phase: *Angular* and *ASP.NET Core*.

  We believe they are adopted early in these projects because even though we consider they started at the beginning of their repositories' versioning history, they have an older background behind them. *Angular* is a successor to *AngularJS*, which was created in 2010 and *ASP.NET Core* is a successor to *ASP.NET*, which was created in 2002.

  So it's not unreasonable to say that these two projects might be considered just yet another version of their predecessors. Considering that the other three projects adopt this pattern after the first year and a half of their creation, this pattern could potentially fit into the group of the ones that are adopted later.

5.2.2 Number of Stars and Number of Forks

In Figures 5.9 and 5.10, we can see five patterns that are adopted with a low number of users (stars) and a low number of contributors (forks). These five patterns are: CUSTOMIZATION POINTS, DESIGN INTERNALS, DOCUMENTATION ROADMAP, FRAMEWORK OVERVIEW, and GRADED EXAMPLES.

These are the same patterns mentioned above when we analyzed the association between pattern adoption and days since creation. This situation probably occurs because when the project is in its beginning, users and contributors are in a lower number than when a project is in an advanced phase.

For the remaining patterns, there is no clear conclusion from this data.

We don't dismiss the probability of existing other metrics to study when analyzing documentation that we didn't look at. Another possibility is that the number of stars and number of forks, are not good metrics for the number of users and the number of contributors, respectively.

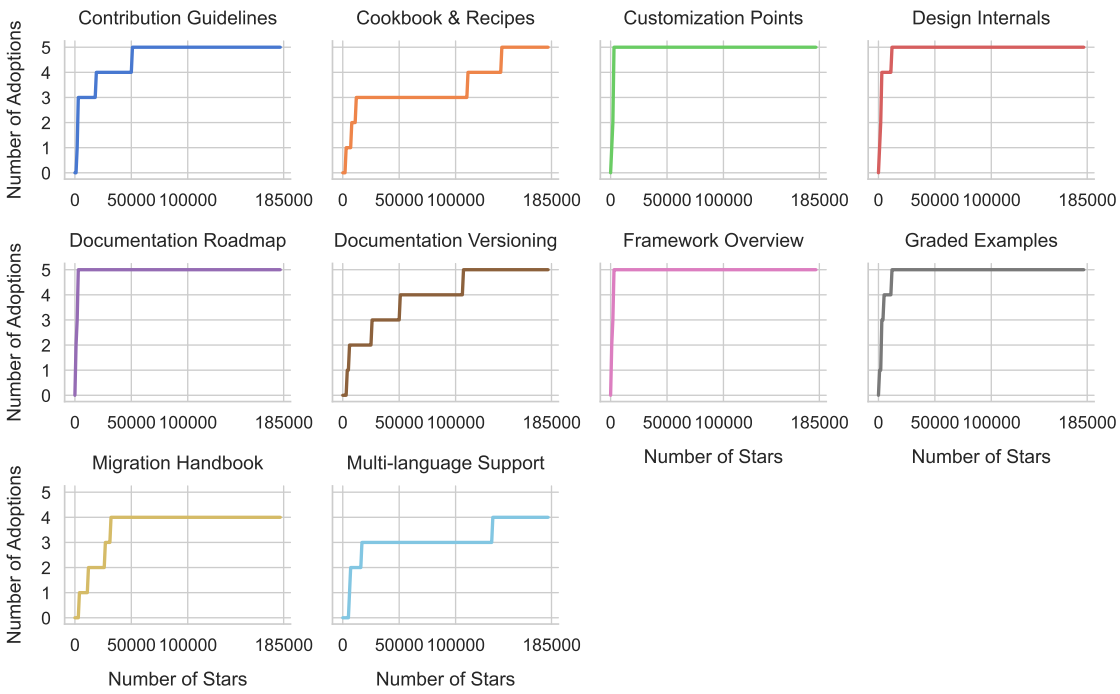

Figure 5.9: Overall pattern adoption over stars

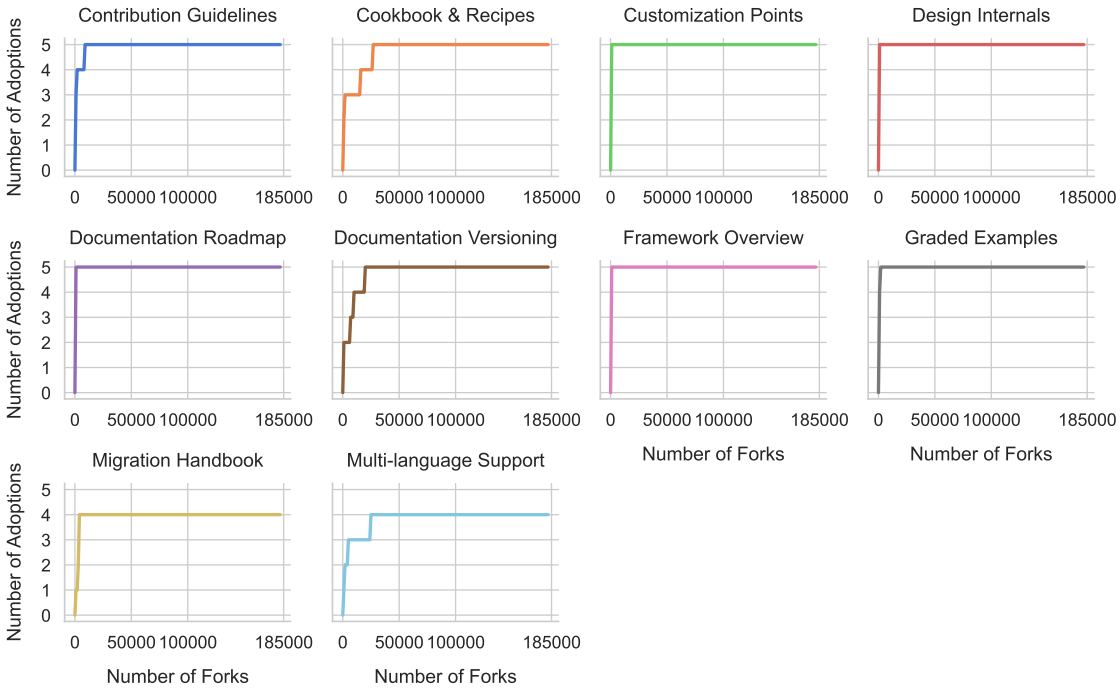

Figure 5.10: Overall pattern adoption over forks

5.2.3 Pattern Relationships

To identify potential relationships between these ten patterns besides the ones that are already identified in the pattern descriptions, and to understand if current relationships actually occur, we targeted two different questions:

1. *"Which patterns are adopted simultaneously?"*

   To answer this, we calculated the Pearson correlation to assess how linearly correlated one pattern adoption is to another pattern adoption. Figure 5.11 depicts that correlation.

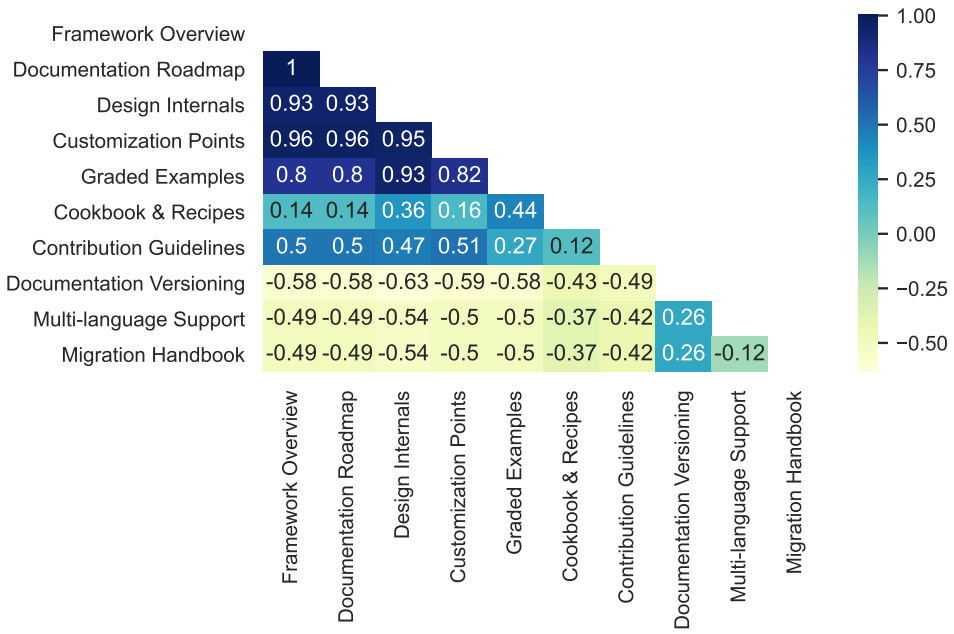

Figure 5.11: Pearson correlation coefficient between patterns simultaneous adoption.

From this data, we consider two types of correlations: *very likely* (when values are equal or higher than 0.8) and *always occurs* (when values equal to 1).

- FRAMEWORK OVERVIEW and DOCUMENTATION ROADMAP adoption *always occurs* when the other is adopted.

  In each project that we analyzed, FRAMEWORK OVERVIEW is the first element of the DOCUMENTATION ROADMAP. So, when a user navigates to the roadmap section he will immediately observe an introduction about the framework. As their implementations are tightly coupled, their adoption occurs naturally at the same time;

- GRADED EXAMPLES is *very likely* to be adopted when DESIGN INTERNALS adoption occurs.

  We think this happens because the set of examples given by GRADED EXAMPLES typically include grasps of the system internals, which requires the adoption of DESIGN INTERNALS;

- DESIGN INTERNALS is *very likely* to be adopted when CUSTOMIZATION POINTS is also adopted.

  This situation happens because CUSTOMIZATION POINTS describes how and which parts of a framework can be customized. To address each customization, it's needed to understand the internal details of the system which are detailed by DESIGN INTERNALS. Therefore, adopting CUSTOMIZATION POINTS requires adopting DESIGN INTERNALS.

2. *"Which patterns already exist when a certain pattern is adopted?"*

   For this question, we calculated the conditional probability of one pattern already existing when another is adopted. Unlike the previous approach, we have asymmetries in pattern adoption. So, the existence of pattern A when pattern B is adopted doesn't necessarily assert that pattern B exists if pattern A is adopted. Figure 5.12 shows us the conditional probability, where we can see that:

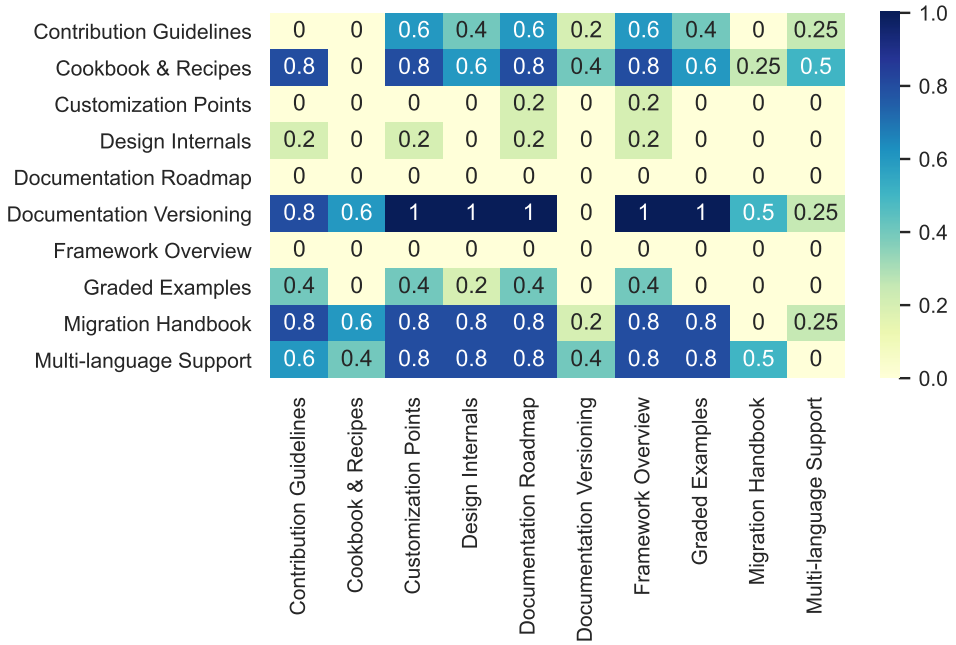

Figure 5.12: Conditional probability of adopting a pattern in the y-axis (e.g., Contribution Guidelines), considering the pattern in the x-axis existed (e.g., $P(\text{Contribution Guidelines} \mid \text{Cookbook \& Recipes})$).

- DOCUMENTATION VERSIONING is the one who has more patterns already adopted (six patterns) when itself is adopted.

  This indicates that this pattern occurs in a more advanced phase of the project when other patterns are already adopted. This is not surprising because DOCUMENTATION

VERSIONING provides an organization of documentation previous versions, and at the beginning of a software project, there is only one version.

The need to implement this pattern is only perceived when software evolves, with an increase of their software versions;

- FRAMEWORK OVERVIEW and DOCUMENTATION ROADMAP don't have any patterns being adopted before themselves.

  This reinforces what we saw in Section 5.2.1, that these patterns are usually the firsts to be used in the documentation. We think this happens because it's essential for users that reach documentation for the first time that they understand the framework purpose, and they know how to navigate into documentation sections.

Considering a *highly positive correlation* (higher than 0.8) and a *higher conditional probability* (higher than 0.8), we constructed a pattern map as seen in Figure 5.13.

In this pattern map we have ten patterns from two different groups. Each group is separated by their author. The first one consists of the following patterns: CONTRIBUTION GUIDELINES, DOCUMENTATION VERSIONING, MIGRATION HANDBOOK, and MULTI-LANGUAGE SUPPORT. The second group includes the following patterns: COOKBOOK & RECIPES, CUSTOMIZATION POINTS, DESIGN INTERNALS, DOCUMENTATION ROADMAP, FRAMEWORK OVERVIEW, and GRADED EXAMPLES.

This pattern map describes the already defined relationships between patterns (orange dashed arrow), potential new relationships within the pattern group inferred from the conditional probabilities (green solid arrow), potential new relationships outside the pattern group inferred from the conditional probabilities (purple solid arrow), and potential new relationships inferred from the correlations (blue solid line).

We can see 13 relationships that are already defined. However, we notice an appearance of **19 potential relationships** between patterns. The majority of these (16) occur between different groups. This phenomenon is comprehensible because it never existed an analysis of how these two groups relate with each other.

On the other hand, the three potential relationships found within the same groups can be justified because typically in the pattern mining process, the authors define each pattern section based on their experience and without a well-defined systematic process. In this case, the possible relationships appear from a systematic process that we implemented in previous sections, that happens in the open source frameworks domain.

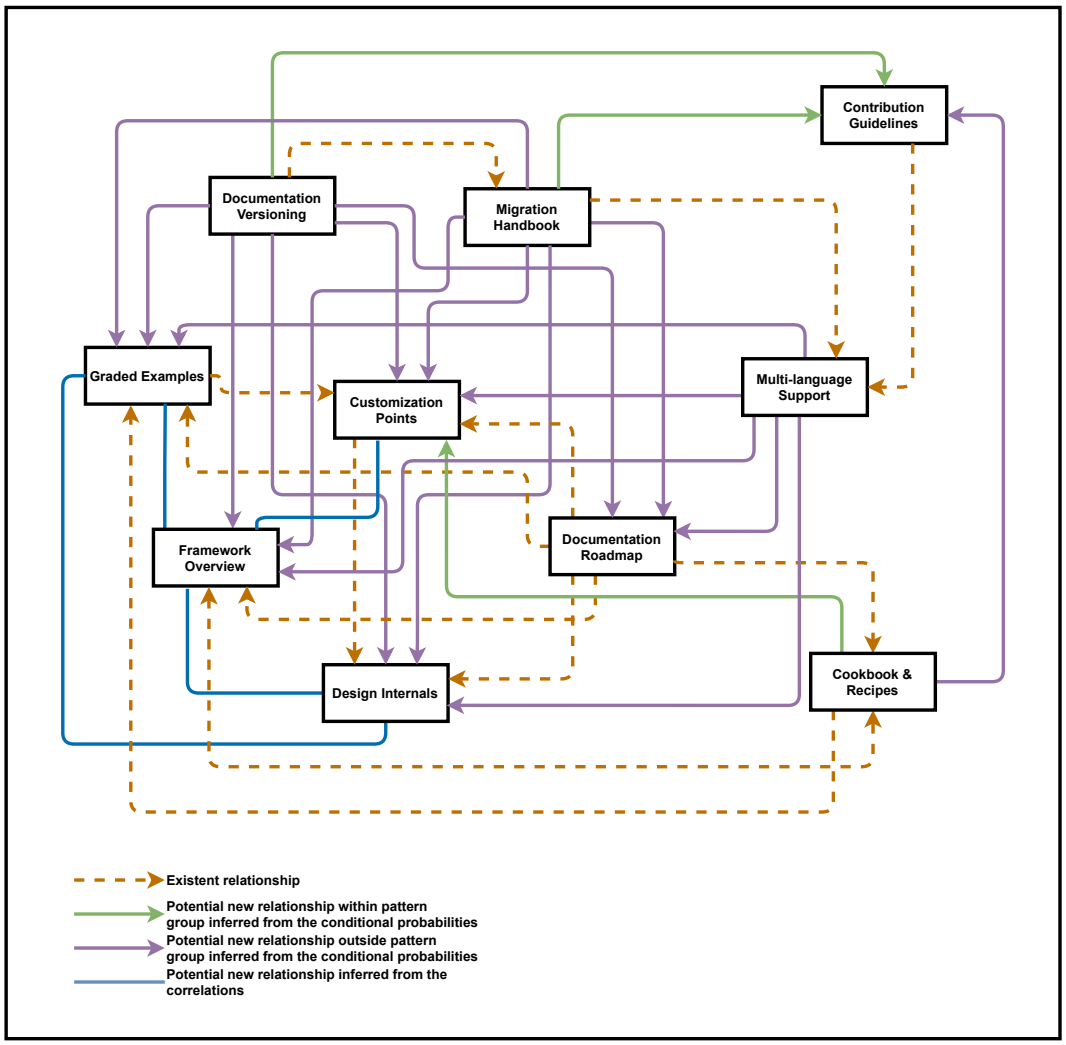

Figure 5.13: Pattern map gathered from Pearson correlation and conditional probability calculations

5.3 Threats to Validity

Naturally, this work contains several threats that were considered. For each one, it was considered possible procedures to mitigate their overall impact on our work. This impact if not taken into consideration could potentially lead to biased results.

- The validation over documentation on **open source projects** instead of documentation used in industry projects could potentially lead to the wrong interpretation of the data. To mitigate this situation, we selected some open source projects sponsored by companies or with at least some company members working on them. This decision tries to relate the extracted open source documentation with documentation written or maintained in corporate

environments.

- The **limited sample size** was highly conditioned by the possibility of extracting metrics for the entire project life-cycle. We consider this threat was not completely mitigated, however, we consider that increasing the number of analyzed projects could help strengthen this study's results and diminish this threat.

- Another threat is that this study **may not be possible to generalize** to other framework types, considering that all the selected projects are web frameworks. We are aware that further analysis of other domains can lead to more interpretations. However, the fact that all documentation patterns selected to perform this analysis were related to web frameworks helped us to perform a more accurate analysis of this specific topic.

  We believe that these patterns can be applied to frameworks in other domains like machine learning or mobile frameworks, but further empirical evidence is needed to draw such a conclusion. This is seen positively because it means there is more research needed in this area of software engineering that can benefit from our work.

Chapter 6

Conclusions

In this chapter, we present an overview of our work, followed by our main contributions. Lastly, we provide future steps that could follow this work.

6.1 Research Answers

Software documentation is, without a doubt, an important part of the captured knowledge of a software project. In the last years, the scientific community has raised issues around writing and maintaining documentation and provided best practices that solve recurring problems. Despite existing several studies around this topic, it still lacked a comprehensive study of software documentation in real-world software projects, specifically, in open source frameworks to guide teams in adopting software documentation practices.

Considering this, in our entire work we sought to answer the following research questions:

- RQ1. *"What are the best practices for documenting software frameworks?"*

  During the state-of-the-art review (*cf.* Chapter 2) we analyzed software documentation in published works, resulting in the extraction of 114 documentation patterns adopted in software. Complementing that, we identified four patterns that are adopted in open source frameworks (*cf.* Chapter 4). This resulted in a collection of 118 documentation best practices.

- RQ2. *"Does a correlation exist between the pattern adoption and the framework users and framework contributors?"*

  This question is approached in Sections 5.2.1 and 5.2.2. Regarding their adoption, over time we found two different groups for the majority of the patterns: *early adoption* and *late adoption*. These two groups indicate if a pattern is likely to be adopted in a more initial phase of the project or in a more advanced phase of the project. However, regarding the framework users and framework contributors, we didn't find any useful correlation. As mentioned before, this probably occurred because the number of stars and the number of forks are not good metrics for the number of users and the number of contributors, respectively.

- RQ3. *"Is the adoption of specific patterns related to the adoption of others?"*

  In Section 5.2.3 we tackle this question by studying pattern adoption considering two different approaches. The first approach shows us which patterns are adopted simultaneously, while the second one shows us which patterns already exist when a certain pattern is adopted. Combining these two different results, we found the adoption of specific patterns to be related to the adoption of others.

6.2 Main Contributions

To summarize, our main contributions are the following:

- **State-of-the-Art Review** - We provide analysis in published works of documentation, as well as an analysis in empirical studies with patterns, resulting in the identification of 16 quality attributes and 114 patterns about software documentation. This led to the proposal of new categories and relationships between the existing patterns;
- **Pattern Mining Process** - We propose four unidentified documentation patterns: CONTRIBUTION GUIDELINES, DOCUMENTATION VERSIONING, MIGRATION HANDBOOK, and MULTI-LANGUAGE SUPPORT that are adopted in real-world software projects, specifically, open source frameworks;
- **Pattern Adoption Study** - We study the adoption of ten framework patterns in five different open source web frameworks.

6.3 Future Work

Regarding future work, we considered several steps that should be approached within the pattern mining process, and the study of pattern adoption:

- **Extend to Other Domains —** The gathered patterns during the pattern mining process could be helpful in different domains other than open source frameworks. For that, there is a need to evaluate the adoption of these patterns in other contexts;
- **Increase Sampling —** Increase of the study sampling with more open source frameworks that could help reinforce our results around pattern adoption, and even possibly help to identify more recurring solutions as patterns;
- **Gathered Patterns vs Existing Literature Patterns —** A comprehensive study on how gathered patterns and existing literature patterns around frameworks relate could be performed to provide a complete guide for framework maintainers to follow when writing and maintaining documentation. This study may start from the potential pattern relationships gathered from the correlations and conditional probabilities;

- **Contemplate Other Metrics —** Other metrics other than stars and forks could provide more insights regarding the correlation between patterns adoption and framework users and contributors (*e.g.* number of downloads could be considered for framework users);
- **Survey with Users and Contributors —** A survey evolving framework users and framework contributors could provide insights regarding the relevance of the documentation patterns for users, and motivations behind pattern adoption for contributors.

References

[1] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. Co-evolution of project documentation and popularity within GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 360–363, 2014.

[2] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. Software documentation: the practitioners' perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 590–601. IEEE, 2020.

[3] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1199–1210, Piscataway, NJ, USA, 2019. IEEE Press.

[4] Ademar Aguiar. *Framework documentation: A minimalist approach*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

[5] Ademar Aguiar and Gabriel David. Patterns for effectively documenting frameworks. In *Transactions on pattern languages of programming II*, pages 79–124. Springer, 2011.

[6] Yuma Akado, Sakurako Kogure, Alice Sasabe, Jei-Hee Hong, Keishi Saruwatari, and Takashi Iba. Five patterns for designing pattern mining workshops. In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, pages 1–13, 2015.

[7] Christopher Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977.

[8] Felix Bachmann, Len Bass, Jeromy Carriere, Paul Clements, David Garlan, James Ivers, Robert Nord, and Reed Little. Software architecture documentation in practice: Documenting architectural layers. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2000.

[9] Hudson Borges, Marco Tulio Valente, André C. Hora, and Jailton Coelho. On the popularity of GitHub applications: A preliminary note. *CoRR*, abs/1507.00604, 2015.

[10] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns*, volume 1. John Wiley & Sons, 2008.

[11] Filipe Figueiredo Correia. *Documenting Software With Adaptive Software Artifacts*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2015.

[12] Filipe Figueiredo Correia, Ademar Aguiar, Hugo Sereno Ferreira, and Nuno Flores. Patterns for consistent software documentation. In *Proceedings of the 16th Conference on Pattern Languages of Programs*, page 12. ACM, 2009.

[13] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*, SIGDOC '05, pages 68–75, New York, NY, USA, 2005. ACM.

[14] Tapajit Dey and Audris Mockus. Are software dependency supply chain metrics useful in predicting change of popularity of npm packages? In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 66–69, 2018.

[15] Nuno Flores. *Patterns and Tools for Improving Framework Understanding: a Collaborative Approach*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2012.

[16] Andrew Forward and Timothy C Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33. ACM, 2002.

[17] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. Elements of reusable object-oriented software. *Design Patterns. massachusetts: Addison-Wesley Publishing Company*, 1995.

[18] Golara Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. Evaluating usage and quality of technical software documentation: an empirical study. In *Proceedings of the 17th international conference on evaluation and assessment in software engineering*, pages 24–35, 2013.

[19] Golara Garousi, Vahid Garousi-Yusifoğlu, Guenther Ruhe, Junji Zhi, Mahmoud Moussavi, and Brian Smith. Usage and usefulness of technical software documentation: An industrial case study. *Information and Software Technology*, 57:664 – 682, 2015.

[20] Michael Hahsler. A quantitative study of the adoption of design patterns by open source software developers. In *Global Information Technologies: Concepts, Methodologies, Tools, and Applications*, pages 560–577. Igi Global, 2008.

[21] Rashina Hoda, James Noble, and Stuart Marshall. How much is just enough?: some documentation patterns on agile projects. In *Proceedings of the 15th European Conference on Pattern Languages of Programs*, page 13. ACM, 2010.

[22] Takashi Iba and Taichi Isaku. Holistic pattern-mining patterns. In *19th Pattern Languages of Programs conference*. Citeseer, 2012.

[23] IEEE/IEC international standard for preparation of information for use (instructions for use) of products - part 1: Principles and general requirements. Standard, IEC/IEEE, 2019.

[24] Mira Kajko-Mattsson. A survey of documentation practice within corrective maintenance. *Empirical Software Engineering*, 10(1):31–55, 2005.

[25] J. Kotula. Using patterns to create component documentation. *IEEE Software*, 15(2):84–92, March 1998.

[26] Manny M Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124. Springer, 1996.

[27] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. *IEEE software*, 20(6):35–39, 2003.

[28] Peter Naur and Brian Randell. Software engineering: Report of a conference sponsored by the NATO science committee, garmisch, germany, 7th-11th october 1968. *NATO*, 1969.

[29] Dirk Riehle, Nikolay Harutyunyan, and Ann Barcomb. Pattern discovery and validation using scientific research methods. Technical Report CS-2020-01, Technische Fakultät, 2020.

[30] Linda Rising. Patterns mining. *Handbook of object technology*, pages 38–39, 1999.

[31] Andreas Rüping. *Agile documentation: a pattern guide to producing lightweight documents for software projects*. John Wiley & Sons, 2005.

[32] Alice Sasabe, Tomoki Kaneko, Kaho Takahashi, and Takashi Iba. Pattern mining patterns: a search for the seeds of patterns. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*, pages 1–16, 2016.

[33] Douglas C Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, 2013.

[34] Tiago Sousa, Hugo Sereno Ferreira, and Filipe Figueiredo Correia. A survey on the adoption of patterns for engineering software for the cloud. *IEEE Transactions on Software Engineering*, 2021.

[35] G. Uddin and M. P. Robillard. How API documentation fails. *IEEE Software*, 32(4):68–75, July 2015.

[36] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. StackOverflow and GitHub: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[37] Thomas Vestdam. Writing internal documentation. In *EuroPLoP*, pages 511–534, 2001.

[38] Marcello Visconti and Curtis R Cook. An overview of industrial software documentation practice. In *12th International Conference of the Chilean Computer Science Society, 2002. Proceedings.*, pages 179–186. IEEE, 2002.

[39] Shaowei Wang, David Lo, and Lingxiao Jiang. An empirical study on developer interactions in StackOverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1019–1024, 2013.

[40] Simon Weber and Jiebo Luo. What makes an open source code popular on GitHub? In *2014 IEEE International Conference on Data Mining Workshop*, pages 851–855. IEEE, 2014.

Appendix A

Identified Patterns From Literature

The patterns identified in the literature are specified below as patlets [10]. This format contains the pattern name, the problem, the solution, the quality attributes related to that pattern and the corresponding pattern category (if mentioned in the publication by the authors).

Patterns for Consistent Software Documentation

INFORMATION PROXIMITY [12]

Problem: "How to preserve documentation consistency when fragments of related content are scattered across documents?"

Solution: "Keep related information fragments easily accessible from each other, using a single source, links, views, or transclusion, for example, so that it may be easier to assess if they are in-sync."

Quality attributes: Consistency; Up-to-dateness; Readability;

CO-EVOLUTION [12]

Problem: "When to update a related piece of information in documentation?"

Solution: "When a change is introduced, update the related information parts."

Quality attributes: Up-to-dateness;

DOMAIN-STRUCTURED INFORMATION [12]

Problem: "How to structure the information in documentation?"

Solution: "Organize contents according to their domain, so that the information form directly relates to domain concepts."

Quality attributes: Relevance; Usefulness;

INTEGRATED ENVIRONMENT [12]

Problem: "How to support the maintenance of consistency between independent artifacts with related content?"

Solution: "Use an integrated environment, where several types of artifact and their relations may be maintained uniformly."

Quality attributes: Consistency;

Patterns for Effectively Documenting Frameworks

DOCUMENTATION ROADMAP [5]

Problem: "How to help readers on quickly finding in the overall documentation their way to the information they need?"

Solution: "Start by providing a roadmap for the overall documentation, one that reveals its organization, how the pieces of information fit together, and that elucidates readers of different audiences about the main entry points and the paths in the documentation that may drive them quickly to the information they are looking for, especially at their first contact."

Quality attributes: Accessibility; Usefulness;

FRAMEWORK OVERVIEW [5]

Problem: "How to help readers on getting a quick, but precise, first impression of a new framework?"

Solution: "Provide an introductory document, in the form of a framework overview, that describes the domain covered by the framework in a clear way, i.e. the application domain and the range of solutions for which the framework was designed and is applicable."

Quality attributes: Accessibility; Relevance; Understandability; Usefulness;

COOKBOOK AND RECIPES [5]

Problem: "How to quickly provide users with information that helps them learning how to use the framework?"

Solution: "Provide a collection of recipes, one for each framework customization, organized in a cookbook, which acts as a guide to the contents of all its recipes."

Quality attributes: Usability; Usefulness;

GRADED EXAMPLES [5]

Problem: "How to help readers on getting started fast using a framework both for simple and complex kinds of usage?"

Solution: "Provide a small but representative graded set of training examples to illustrate the framework's applicability and features, each one illustrating a single new way of customization, smoothly growing in complexity, and eventually altogether providing complete coverage."

Quality attributes: Relevance; Understandability; Usefulness;

CUSTOMIZATION POINTS [5]

Problem: "How to help readers learn in detail how to customize a specific part of a framework?"

Solution: "Provide a list of the framework's customization points, also known as hot-spots, i.e., the points of predefined refinement where framework customization is supported, and, for each one, describe in detail the hooks it provides and the hot-spot subsystem that implements its flexibility."

Quality attributes: Relevance; Understandability; Usefulness;

DESIGN INTERNALS [5]

Problem: "How to help framework users on quickly grasping the design and implementation of a framework to support them on achieving advanced customizations, not typical, or not specifically documented?"

Solution: "Provide concise detailed information about the design internals of the framework, especially the areas designed to support configuration, known as hot-spots. You should start by describing how the framework hot-spots support configuration."

Quality attributes: Understandability; Usability; Usefulness;

Using Patterns To Create Component Documentation

ENVIRONMENT INDEPENDENCE [25]

Solution: "The documentation system must be feasible and effective in any operating environment."

Pattern category: "Generative Patterns"

Quality attributes: Accuracy;

HYPERTEXT [25]

Solution: "Present the documentation as hypertext to enable effective navigation of the information."

Pattern category: "Generative Patterns"

Quality attributes: Accessibility; Usability;

EMBED CONTENT IN SOURCE CODE [25]

Solution: "Ensure that the component documentation is synchronized with the component source code."

Pattern category: "Generative Patterns"

Quality attributes: Up-to-dateness;

AUTOMATE TEDIOUS TASKS [25]

Solution: "Automate as many documentation production and maintenance tasks as possible to relieve the workload of the authors."

Pattern category: "Generative Patterns"

Quality attributes: Automatability;

LEVERAGE PROGRAMMING LANGUAGE SEMANTICS [25]

Solution: "Take advantage of all the information about the component that is inherent in the source code."

Pattern category: "Generative Patterns"

Quality attributes: Usefulness;

CENTRALIZE DOCUMENT DESIGN [25]

Solution: "Centralize the layout and design of the documentation to ensure its consistency."

Pattern category: "Generative Patterns"

Quality attributes: Consistency; Usability;

DIRECTED AND EXPLORATORY NAVIGATION [25]

Solution: "Provide a means for both directed and exploratory navigation to support reader education and research."

Pattern category: "Generative Patterns"

Quality attributes: Accessibility; Usability;

TUTORIAL INFORMATION WITH EXAMPLES [25]

Solution: "Provide the client with information on how to use the component in specific, common use cases."

Pattern category: "Content Patterns"

Quality attributes: Usability; Usefulness;

DIAGRAMS AND ILLUSTRATIONS [25]

Solution: "Include diagrams and illustrations in documentation for clarification purposes"

Pattern category: "Content Patterns"

Quality attributes: Readability; Usability; Usefulness;

OVERVIEW INFORMATION [25]

Solution: "Provide high-level overview descriptions and summaries in the documentation for each component, library, and class."

Pattern category: "Content Patterns"

Quality attributes: Relevance; Understandability; Usefulness;

GENERATE SHORT FORM [25]

Solution: "Automatically create a brief summary of the extensions that a derived class makes to its base class."

Pattern category: "Content Patterns"

Quality attributes: Understandability; Usefulness;

GENERATE FLAT FORM [25]

Solution: "Automatically create a complete summary of all class members from all parent classes."

Pattern category: "Content Patterns"

Quality attributes: Understandability; Usefulness;

GENERATE INHERITANCE DIAGRAMS [25]

Solution: "Automatically generate diagrams depicting the inheritance structure of the classes."

Pattern category: "Content Patterns"

Quality attributes: Automatability;

GENERATE CALL DEPENDENCY DIAGRAMS [25]

Solution: "Automatically generate diagrams depicting the interclass dependencies in each library or component."

Pattern category: "Content Patterns"

Quality attributes: Readability; Usability; Usefulness;

GENERATE EVALUATION METRICS [25]

Solution: "Automatically generate object-oriented code metrics to help clients evaluate the components."

Pattern category: "Content Patterns"

Quality attributes: Automatability; Usefulness;

DERIVE CLASS CHARACTERISTICS [25]

Solution: "Automatically list the descriptive attributes that characterize each class."

Pattern category: "Content Patterns"

Quality attributes: Automatability; Usefulness;

MAP ORGANIZATION TO LANGUAGE CONSTRUCTS [25]

Solution: "Structure the hypertext nodes and the links between them to parallel the programming language constructs used to implement the component."

Pattern category: "Structure Patterns"

Quality attributes: Correctness; Relevance;

HIERARCHICAL LIBRARIES [25]

Solution: "In the documentation, provide a mechanism for grouping related classes together to form subsystem libraries."

Pattern category: "Structure Patterns"

Quality attributes: Consistency; Readability;

LINK TO EXTERNAL DOCUMENTS [25]

Solution: "Join supplementary documents to the component documentation through hypertext links"

Pattern category: "Structure Patterns"

Quality attributes: Accessibility;

BACKTRACKING [25]

Solution: "Provide the user with links that backtrack to related, higher levels of the documentation"

Pattern category: "Structure Patterns"

Quality attributes: Accessibility; Usability;

GENERATE HYPERLINKS [25]

Solution: "Automatically hyperlink each occurrence of a software artifact's name to its full documentation."

Pattern category: "Structure Patterns"

Quality attributes: Accessibility; Automatability;

REVEAL LEXICAL SCOPE [25]

Solution: "Structure the documentation to reflect the lexical scope of the software artifacts within a component."

Pattern category: "Structure Patterns"

Quality attributes: Consistency; Usability;

TYPED LINKS [25]

Solution: "Classify hyperlinks by type to enable more directed, selective navigation through the component documentation."

Pattern category: "Structure Patterns"

Quality attributes: Accessibility; Usefulness;

KEYWORD CROSS-REFERENCE [25]

Solution: "Provide a mechanism to help clients find classes in a component within categories of interest."

Pattern category: "Search Patterns"

Quality attributes: Accessibility; Usability;

INHERITANCE NAVIGATION [25]

Solution: "Provide an efficient, convenient means to quickly navigate through the inheritance structure for a given class."

Pattern category: "Search Patterns"

Quality attributes: Accessibility; Usability;

FULL-TEXT SEARCH [25]

Solution: "Provide utilities to search through the full text of the documentation."

Pattern category: "Search Patterns"

Quality attributes: Accessibility; Usability;

GENERATE INDEX [25]

Solution: "Automatically determine which words and phrases in the component documentation are most significant and generate an index from them."

Pattern category: "Search Patterns"

Quality attributes: Automatability; Usefulness;

GENERATE KEYWORD CROSS-REFERENCE [25]

Solution: "Automatically parse class names to create a minimal list of cross-reference keywords."

Pattern category: "Search Patterns"

Quality attributes: Automatability; Usefulness;

P39

MAINTAIN CONSISTENCY [25]

Solution: "Make the visual appearance of the documentation consistent."

Pattern category: "Presentation Patterns"

Quality attributes: Consistency; Usability;

DISTRIBUTION VIA THE WEB [25]

Solution: "Use Web pages on the Internet to provide access to the most current version of the documentation."

Pattern category: "Presentation Patterns"

Quality attributes: Accessibility; Usability;

ICONS, HEADINGS, AND COLORS MARK NODES [25]

Solution: "Use pictorial icons, standard headings, and/or colors to mark the context and purpose of hypertext nodes in the documentation."

Pattern category: "Presentation Patterns"

Quality attributes: Readability; Usability;

READABILITY THROUGH TYPOGRAPHY [25]

Solution: "Graphically represent the textual information clearly and effectively."

Pattern category: "Presentation Patterns"

Quality attributes: Readability; Usability;

TYPOGRAPHY-ENCODED DETAILS [25]

Solution: "Use typographic conventions to encode syntactic details where programming language syntax is directly included in the documentation."

Pattern category: "Presentation Patterns"

Quality attributes: Readability; Usability;

STRUCTURE FOR PHYSICAL MANAGEMENT [25]

Solution: "Design the physical repository of the documentation to enable its easy versioning and administration as a unit."

Pattern category: "Presentation Patterns"

Quality attributes: Up-to-dateness; Usefulness;

SYNTACTICALLY SCOPED DOCUMENTATION [25]

Solution: "Use syntactic rules to associate embedded documentation with the programming-language construct it documents."

Pattern category: "Embedding Patterns"

Quality attributes: Consistency; Usefulness;

DOCUMENT IN SMALL CHUNKS [25]

Solution: "Structure the embedded comment text so that software artifacts are documented in small increments."

Pattern category: "Embedding Patterns"

Quality attributes: Consistency; Usefulness;

ALLOW TYPOGRAPHIC CONTROL [25]

Solution: "Program the extraction tool to support typographic markup within the manually authored documentation text."

Pattern category: "Embedding Patterns"

Quality attributes: Automatability;

READABLE EMBEDDED DOCUMENTATION [25]

Solution: "Design a documentation-embedding syntax that enhances the readability of the source code."

Pattern category: "Embedding Patterns"

Quality attributes: Readability; Usefulness;

EMBEDDED IMAGES [25]

Solution: "Centralize file management by embedding ASCII-encoded images within the source code for later extraction, decoding, and inclusion in the final documentation."

Pattern category: "Embedding Patterns"

Quality attributes: Standardization;

How Much is Just Enough? Some Documentation Patterns on Agile Projects

---

FAKE DOCUMENTATION [21]

Problem: "How do you co-ordinate the timing of documentations produced across Agile and non-Agile projects?"

Solution: "Time the production of a minimal amount of traditional documentation to co-ordinate with non-Agile teams."

Quality attributes: Cost; Effort; Sufficiency;

TIME STAMP [21]

Problem: "How do you clarify who initiated a change request?"

Solution: "Document change decisions with timestamp on a wiki."

Quality attributes: Standardization; Up-to-dateness;

E-BACKUP [21]

Problem: "How do you avoid losing all your story cards and task post-its?"

Solution: "Make electronic backups of paper artifacts."

Quality attributes: Cost; Effort; Standardization;

PROJECT DICTIONARY [21]

Problem: "How do you translate customer's business requirements into technical tasks?"

Solution: "Engage your customers to document business terms, their meanings, and contexts of use into a project dictionary."

Quality attributes: Consistency; Correctness; Understandability;

ADVANTAGE AGILE [21]

Problem: "How do you demonstrate the Agile advantage over successful non-Agile projects measured using traditional metrics?"

Solution: "Document positive customer feedback."

Quality attributes: Usefulness;

Agile Documentation - A Pattern Guide to Producing Lightweight Documents for Software Projects

---

TARGET READERS [31]

Problem: "How can the project team ensure that the documents they produce will be appreciated?"

Solution: "First and foremost, each document must have a target readership, and must address these readers in order to prove useful."

Pattern category: "Finding the Right Topics"

Quality attributes: Relevance; Usefulness;

FOCUSED INFORMATION [31]

Problem: "How can documents be prevented from meandering and getting nowhere fast?"

Solution: "A clear and identifiable focus on a particular topic makes a document concise and straightforward. The straightforward document offers the information relevant to this topic, but no more than that."

Pattern category: "Finding the Right Topics"

Quality attributes: Consistency; Relevance;

INDIVIDUAL DOCUMENTATION REQUIREMENTS [31]

Problem: "How can unnecessary documentation requirements be avoided?"

Solution: "The most effective approach towards documentation is for each project to define its documentation requirements individually."

Pattern category: "Finding the Right Topics"

Quality attributes: Sufficiency;

DOCUMENTATION PORTFOLIO [31]

Problem: "How can teams reuse the knowledge about which documents might be required in their projects?"

Solution: "A documentation portfolio describes which documents might be necessary in a software project, and their scope. If an organisation sets up such a portfolio, projects can choose those documents they need, checking the necessity of each candidate document individually."

Pattern category: "Finding the Right Topics"

Quality attributes: Cost; Effort; Standardization;

FOCUS ON LONG-TERM RELEVANCE [31]

Problem: "How can projects avoid producing documentation that expires too soon?"

Solution: "There is much value in documentation that focuses on issues with a long-term relevance – issues that will play a role in a later project phase or in future projects."

Pattern category: "Finding the Right Topics"

Quality attributes: Up-to-dateness;

SPECIFICATION AS A JOINT EFFORT [31]

Problem: "How can development projects ensure that they head in the direction the customer wants?"

Solution: "Every development project requires a specification, which reflects the requirement analysis done jointly by the project team and the customer."

Pattern category: "Finding the Right Topics"

Quality attributes: Relevance; Usefulness;

DESIGN RATIONALE [31]

Problem: "How can the team make sure that the foundations are laid for future design changes?"

Solution: "Design documents become a valuable source of information if they aren't restricted to describing the actual design, but also focus on the rationale behind the design and explain why the particular design was chosen."

Pattern category: "Finding the Right Topics"

Quality attributes: Completeness;

THE BIG PICTURE [31]

Problem: "How can people be introduced to a project without being confronted with a deluge of technical details?"

Solution: "A good feel for a project is best conveyed through a description of the 'big picture' of the architecture that underlies the system under construction."

Pattern category: "Finding the Right Topics"

Quality attributes: Accessibility; Relevance; Understandability; Usefulness;

SEPARATION OF DESCRIPTION AND EVALUATION [31]

Problem: "How can authors prevent loss of credibility?"

Solution: "Authors gain credibility if, in their documents, they clearly separate description from evaluation."

Pattern category: "Finding the Right Topics"

Quality attributes: Consistency;

REALISTIC EXAMPLES [31]

Problem: "How can abstract material be explained in a comprehensible way?"

Solution: "Project documents are much more convincing if they include realistic examples from the project's context."

Pattern category: "Finding the Right Topics"

Quality attributes: Accessibility; Completeness; Understandability;

STRUCTURED INFORMATION [31]

Problem: "How can information be presented in an easily accessible way?"

Solution: "Most project documents are best organised as sequential yet well-structured text. This begins with well-chosen chapters and sections, but may well extend to using textual building blocks consistently throughout a document."

Pattern category: "Structuring Individual Documents"

Quality attributes: Accessibility; Consistency;

JUDICIOUS DIAGRAMS [31]

Problem: "How can authors provide an overview of structures and processes in a convenient way?"

Solution: "Diagrams can provide excellent overviews, while an accompanying text explains details to the extent that is necessary."

Pattern category: "Structuring Individual Documents"

Quality attributes: Completeness; Understandability;

UNAMBIGUOUS TABLES [31]

Problem: "How can authors present systematic information in a precise way?"

Solution: "Tables offer a compact format for the concise and unambiguous presentation of information."

Pattern category: "Structuring Individual Documents"

Quality attributes: Consistency; Relevance;

GUIDELINES FOR READERS [31]

Problem: "How can potential readers be informed whether they should read a document, and if so, on which parts they should focus?"

Solution: "Some brief guidelines at the beginning of each document can inform potential readers of the purpose the document serves and explain how different groups of readers should approach the document."

Pattern category: "Structuring Individual Documents"

Quality attributes: Accessibility; Understandability;

THUMBNAIL SKETCHES [31]

Problem: "How can readers get an overview of the topics dealt with in a document?"

Solution: "Thumbnail sketches provide brief descriptions of the sections of a document, including the section's general goals, as well as its major ideas."

Pattern category: "Structuring Individual Documents"

Quality attributes: Relevance; Understandability;

TRACEABLE REFERENCES [31]

Problem: "How can documents be linked to each other?"

Solution: "A document should include references to other documents only if readers can obtain those documents without much effort."

Pattern category: "Structuring Individual Documents"

Quality attributes: Accessibility; Consistency;

GLOSSARY [31]

Problem: "How can authors make sure that readers understand the vocabulary used in a document?"

Solution: "A glossary can explain technical terms as well as the terms specific to the application domain."

Pattern category: "Structuring Individual Documents"

Quality attributes: Readability; Standardization; Understandability;

DOCUMENT HISTORY [31]

Problem: "How can confusion be avoided between versions of a document?"

Solution: "A document history can explain the differences to previous versions of a document, and can relate the document to versions of the software it describes."

Pattern category: "Structuring Individual Documents"

Quality attributes: Up-to-dateness;

TEXT ON 50% OF A PAGE [31]

Problem: "How much space on a page should be devoted to text?"
Solution: "About 50% of the page should be devoted to text."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability; Usability;

TWO ALPHABETS PER LINE [31]

Problem: "What is the optimum line width?"
Solution: "Approximately two lowercase alphabets of the standard typeface should fit on one line."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability;

120% LINE SPACING [31]

Problem: "What is the optimum line spacing?"
Solution: "The best line spacing is roughly 120% of the type size."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability;

TWO TYPEFACES [31]

Problem: "How many typefaces are appropriate, and which?"
Solution: "In most cases, two typefaces per document are appropriate – a serif typeface for the body text and a sans-serif typeface for the headings."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability; Usability;

CAREFUL USE OF TYPE VARIATIONS [31]

Problem: "How can parts of a text be emphasised?"
Solution: "Type variations can be used for emphasis, but they should be used with care."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability; Usability;

CAREFUL RULING AND SHADING [31]

Problem: "How can table cells be separated?"
Solution: "Careful ruling and shading leads to highly legible tables."
Pattern category: "Layout and Typography"
Quality attributes: Accessibility; Readability; Usability;

ADJACENT PLACEMENT [31]

Problem: "How can tables and diagrams be integrated into text?"

Solution: "Diagrams and tables are best placed close to the text from which they are referenced."

Pattern category: "Layout and Typography"

Quality attributes: Accessibility; Usability;

COHERENT PAGES [31]

Problem: "What options exist to avoid awkward pagination that tears related information apart?"

Solution: "The reading flow is supported by coherent pages – pages that make sure a minimum of related information appears on either side of a page break."

Pattern category: "Layout and Typography"

Quality attributes: Accessibility; Readability;

DOCUMENT LANDSCAPE [31]

Problem: "How can team members get a good overview of what documentation exists in a project?"

Solution: "The project documentation can be represented as a kind of landscape that team members can use as a mental map when they retrieve or add information. A document landscape that roughly forms a tree suits human intuition best."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Accessibility; Usability; Usefulness;

DOCUMENT ARCHIVE [31]

Problem: "How can projects avoid the loss of any document versions?"

Solution: "Archiving project documentation offers the advantage that versions can be retrieved when necessary."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Up-to-dateness;

WIKI [31]

Problem: "How can documentation be given a more interactive edge?"

Solution: "A Wiki offers access to the project documentation via an intranet server, and in addition allows the team to post notes and messages to others as necessary."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Accessibility; Usefulness;

CODE-COMMENT PROXIMITY [31]

Problem: "What is an easy way to maintain documentation that refers to the actual code?"

Solution: "Documentation of the code, to the extent that a project team considers it necessary, is best done through source code comments. Separate documents should be reserved for higher-level issues such as overviews, requirements, design and architecture."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Consistency; Standardization;

READER-FRIENDLY MEDIA [31]

Problem: "Which is more appropriate: documents intended for on-line use, or documents intended for print?"

Solution: "The choice of a medium must reflect a document's typical usage. The rule of thumb is: print is good for reading, on-line is good for looking things up."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Usability;

SEPARATION OF CONTENTS AND LAYOUT [31]

Problem: "How can the layouts of text documents be changed and reused easily?"

Solution: "Layout styles can be defined and assigned to content portions. These layout styles can easily be changed and can be reused across documents."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Consistency;

SINGLE SOURCE AND MULTIPLE TARGETS [31]

Problem: "How can multiple views of a document be created without doubling maintenance?"

Solution: "The documentation infrastructure can employ mechanisms that take source documents and automatically generate additional views. Such mechanisms avoid double maintenance and ensure consistency."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Automatability;

IMPORT BY REFERENCE [31]

Problem: "How can different documents use the same diagram or table consistently?"

Solution: "Artefacts that need to appear in multiple contexts can be imported by reference into the documents that include them."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Consistency;

SEPARATION OF PROCESSING AND PRINTING [31]

Problem: "How can projects produce useful, printable documents?"

Solution: "If a team chooses to deliver the project documentation in a print format that is widely available, all readers are able to print the documents, independent of the platform they use."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Accessibility; Usefulness;

DOCUMENT TEMPLATES [31]

Problem: "How can all project documents acquire a reasonable structure and a good layout at little cost?"

Solution: "Document templates, once they have been properly designed, impose their structure and layout on all documents that are produced using them."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Consistency;

FEW TOOLS [31]

Problem: "How can projects minimise the effort spent on the introduction and use of documentation tools?"

Solution: "Almost all projects can manage with a small set of documentation tools."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Effort;

ANNOTATED CHANGES [31]

Problem: "How can authors avoid confusion over changes they have made?"

Solution: "While a document is under development, authors can use automatic annotations to identify those parts of the document that have changed recently."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Automatability;

NOTIFICATION UPON UPDATE [31]

Problem: "How can readers be prevented from using outdated versions?"

Solution: "Whenever there is a significant change in a project document, all potential readers should be notified of the new version. The notification should roughly explain what has been changed, but should not include the updated material itself."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Up-to-dateness;

REORGANISATION UPON REQUEST [31]

Problem: "How can the documentation infrastructure be maintained?"

Solution: "Frequent reorganisation makes things worse, not better. Reorganisation of the documentation infrastructure should take place only when it is requested by the members of the project team."

Pattern category: "Infrastructure and Technical Organization"

Quality attributes: Consistency; Standardization;

A DISTINCT ACTIVITY [31]

Problem: "How should resources be assigned to documentation activities?"

Solution: "When documentation is considered a distinct project activity, and not just the by-product of coding, it can be assigned its own budget, priority and schedule. Documentation can then be weighed against other project activities."

Pattern category: "Management and Quality Assurance"

Quality attributes: Cost;

ONE RESPONSIBLE AUTHOR [31]

Problem: "How many people should be responsible for a document?"

Solution: "For each project document, there must be one person who accepts responsibility for it. This person need not write the document alone, but must coordinate the contributions from other people."

Pattern category: "Management and Quality Assurance"

Quality attributes: Standardization;

CONTINUING DOCUMENTATION [31]

Problem: "When should project documentation be written?"

Solution: "Project documentation, when it evolves continuously as the project goes on, offers the advantage that it reflects the last stable state of the project."

Pattern category: "Management and Quality Assurance"

Quality attributes: Effort;

WRITING AND REFLECTION [31]

Problem: "How can documentation and other project activities stimulate each other?"

Solution: "To get the best out of documentation, team members have to spend time on the actual writing, as well as in reflection on what they have written, preferably in an undisturbed environment."

Pattern category: "Management and Quality Assurance"

Quality attributes: Effort; Standardization;

REVIEW CULTURE [31]

Problem: "How can the quality of the project documents be improved?"

Solution: "Documentation can profit a lot from reviews, provided a review culture has been established in which both authors and reviewers feel comfortable."

Pattern category: "Management and Quality Assurance"

Quality attributes: Usefulness;

REVIEW BEFORE DELIVERY [31]

Problem: "How can authors receive the right feedback at the right time?"

Solution: "Early reviews are fine as they help the author shape the scope and the structure of a document. But before a document is officially distributed, or delivered to the customer, a review is mandatory."

Pattern category: "Management and Quality Assurance"

Quality attributes: Standardization;

CUSTOMER REVIEW [31]

Problem: "How can a team use documentation to increase customer involvement?"

Solution: "Customer reviews can improve the quality of a document, especially as far as the domain expertise is concerned, and at the same time add to team building and integration."

Pattern category: "Management and Quality Assurance"

Quality attributes: Usefulness; Completeness;

A DISTANT VIEW [31]

Problem: "How can authors obtain unbiased feedback?"

Solution: "Authors can obtain unbiased feedback from reviewers who are interested in the topic and who are generally knowledgeable in the field, but who are not involved in the actual work described in the document."

Pattern category: "Management and Quality Assurance"

Quality attributes: Usefulness;

INFORMATION MARKETPLACE [31]

Problem: "How can good documents be prevented from going sadly unnoticed?"

Solution: "Documents gain more attention if the intended readers are actively invited to read them."

Pattern category: "Management and Quality Assurance"

Quality attributes: Readability; Relevance; Usefulness;

KNOWLEDGE MANAGEMENT [31]

Problem: "How can future projects profit from a successful project?"

Solution: "Only when project documentation is made available organisation-wide do future projects have a chance of drawing on the expertise gained."

Pattern category: "Management and Quality Assurance"

Quality attributes: Accessibility;

Writing Internal Documentation

SEPARATE AND INTERRELATE DOCUMENTATION AND PROGRAM [37]

Problem: "A pile of documentation in one hand and a pile of code in the other - how can one find the documentation relevant for a given part of the code?"

Solution: "Separate documentation and program, and use typed links to provide selective and mutual navigation between documentation and program."

Pattern category: "Structural patterns"

Quality attributes: Consistency; Relevance;

DOCUMENT STRUCTURE FOLLOWS PROGRAM STRUCTURE [37]

Problem: "Getting started on writing documentation as well as structuring the documentation is not easy, especially for the inexperienced documentation writer - how can one get started writing and structuring documentation?"

Solution: "Let the documentation structure follow the program structure when presenting static program structure and program entities."

Pattern category: "Structural patterns"

Quality attributes: Understandability; Usefulness;

TRANSVERSE ISSUES [37]

Problem: "It is not feasible to document all aspects of a program, but which aspects should at least be considered?"

Solution: "Describe and explain program relationships such as collaborating program entities or code dependencies. Relate such transverse issues to relevant program fragments."

Pattern category: "Structural patterns"

Quality attributes: Understandability;

EXTRACT COMMONLY USED INFORMATION [37]

Problem: "The documentation often becomes a mess because explanations of program entities become long and complex. It becomes difficult to find relevant explanations, and consequently difficult to update - how can the documentation become more flexible? "

Solution: "Different aspects of a program entity should be explained separately, if these aspects are important in more than one context."

Pattern category: "Structural patterns"

Quality attributes: Sufficiency; Usefulness;

PROGRAM HISTORY [37]

Problem: "Often programmers inadvertently try solutions that have already been proven useless because they have forgotten previous encountered problems - how can this be avoided?"

Solution: "Record the choices made during program development in the documentation. This includes relevant solutions that have been tried but proven useless. In addition, consider recording relevant alternatives such as solutions that are more flexible, optimal or even more aesthetic than the chosen solution."

Pattern category: "Structural patterns"

Quality attributes: Relevance; Understandability; Usefulness;

CONCEPTUAL WRITING [37]

Problem: "Different programmers often use the same words for different things and different words for the same thing – how can we avoid that the documentation inherits such inconsistencies?"

Solution: "Special terminology used about the program must be defined in a central part of the documentation."

Pattern category: "Structural patterns"

Quality attributes: Consistency; Understandability;

INTERTWINE PROGRAMMING AND WRITING DOCUMENTATION [37]

Problem: "Documentation must be written, but when is it time to write it?"

Solution: "Document the overall code design before coding, and document the design details after the coding. In addition, maintain relevant issues that arise during coding by alternating between writing code and documentation."

Pattern category: "Temporal patterns"

Quality attributes: Usefulness;

DOCUMENTATION REFACTORING [37]

Problem: "The program evolves, and documentation gradually falls behind. Keeping the documentation up-to-date according to every change is simply not feasible – when is it time to update the documentation and how can this be done without constantly changing the entire documentation?"

Solution: "Refactor the documentation on a regular basis either by 1) providing updates to the documentation based on a hot list of changes that need immediate propagation, or 2)revise the entire documentation."

Pattern category: "Maintenance patterns"

Quality attributes: Up-to-dateness;

DOCUMENTATION REVIEW [37]

Problem: "Documentation varies from programmer to programmer in terms of quality and quantity - how can the overall quality of documentation be evaluated and ensured during coding?"

Solution: "The documentation should be reviewed regularly in order to ensure the adequacy of the documentation. Documentation reviews can for example be performed in connection with code reviews."

Pattern category: "Maintenance patterns"

Quality attributes: Consistency;

DOCUMENTATION TEMPLATES [37]

Problem: "Different programmers have different styles when writing and structuring documentation. Depending on which programmer wrote what in the documentation, some aspects of the program are well explained whereas others are not - how can we ensure some kind of uniformity of the documentation?"

Solution: "Identify overall aspects that should (or can) be addressed in the documentation and create templates for these."

Pattern category: "Stylistic patterns"

Quality attributes: Consistency; Standardization;

Appendix B

Project Metrics

This appendix includes all the metrics extracted, and all the data points analyzed for each project.

B.1 Angular

Angular is a TypeScript web framework maintained by Google. The framework exists since 2016 as a successor to AngularJS.

Table B.1: Documentation data points analyzed for Angular

Data point	Days since creation
2015-03-05	168
2015-07-19	304
2015-12-09	447
2016-04-21	581
2016-09-15	728
2017-02-01	867
2017-06-21	1007
2018-04-01	1291
2021-04-30	2416

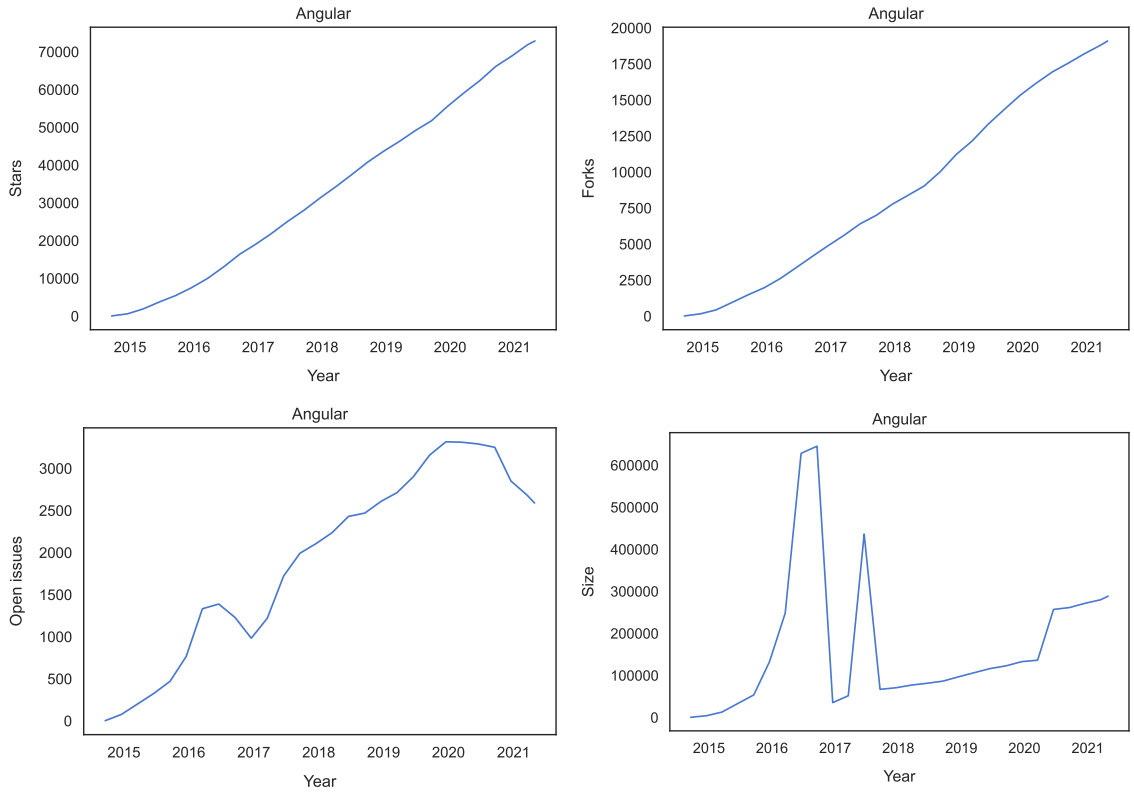

Figure B.1: Repository metrics for Angular

B.2 ASP.NET Core

ASP.NET Core is a framework written in C# to create web apps and services. The framework was first released in 2016 and is a successor to ASP.NET.

Table B.2: Documentation data points analyzed for ASP.NET Core

Data point	Days since creation
2015-05-03	418
2015-09-27	565
2016-02-11	702
2016-11-08	973
2017-08-07	1245
2018-05-04	1515
2021-04-30	2607

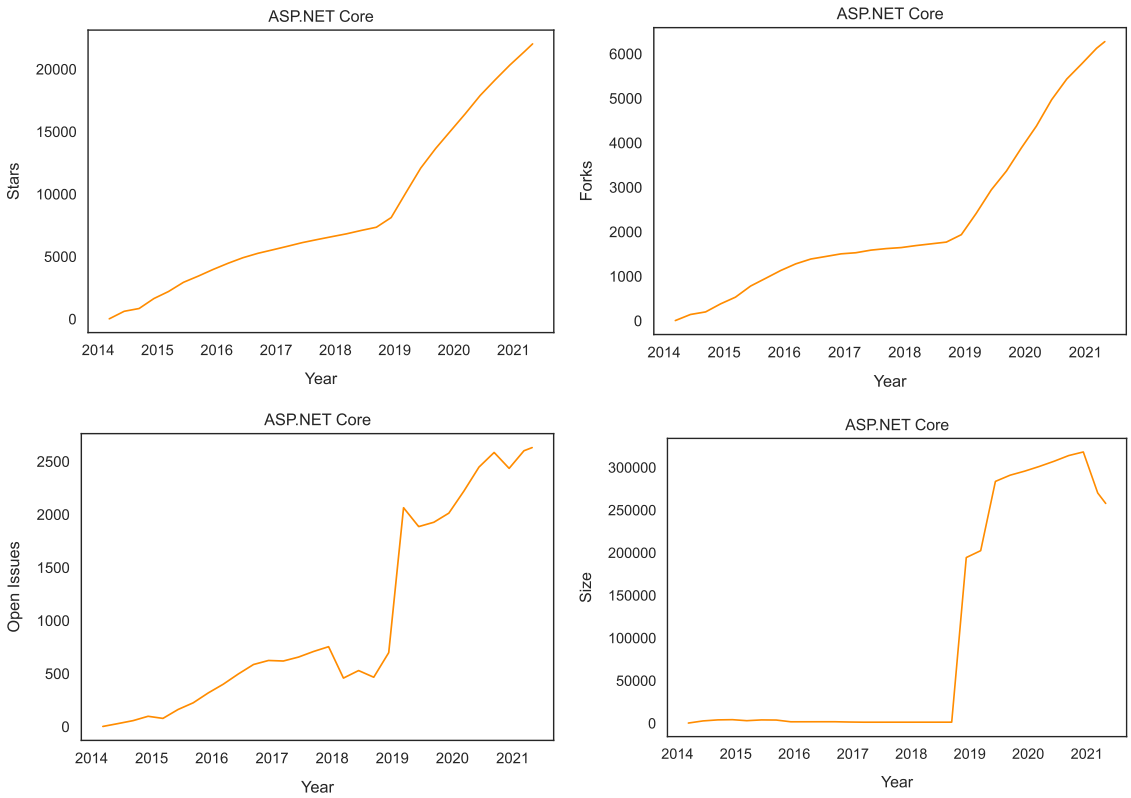

Figure B.2: Repository metrics for ASP.NET Core

B.3 Gatsby

Gatsby is a React-based web framework to build singe-page applications. It was first released in 2015 by Kyle Mathews. Currently is maintained by himself and a few open source contributors.

Table B.3: Documentation data points analyzed for Gatsby

Data point	Days since creation
2015-07-22	62
2016-02-18	273
2016-05-31	376
2016-09-09	477
2017-03-11	660
2017-07-09	780
2018-06-10	1116
2018-10-19	1247
2019-02-28	1379
2019-11-19	1643
2020-08-09	1907
2020-12-19	2039
2021-04-30	2171

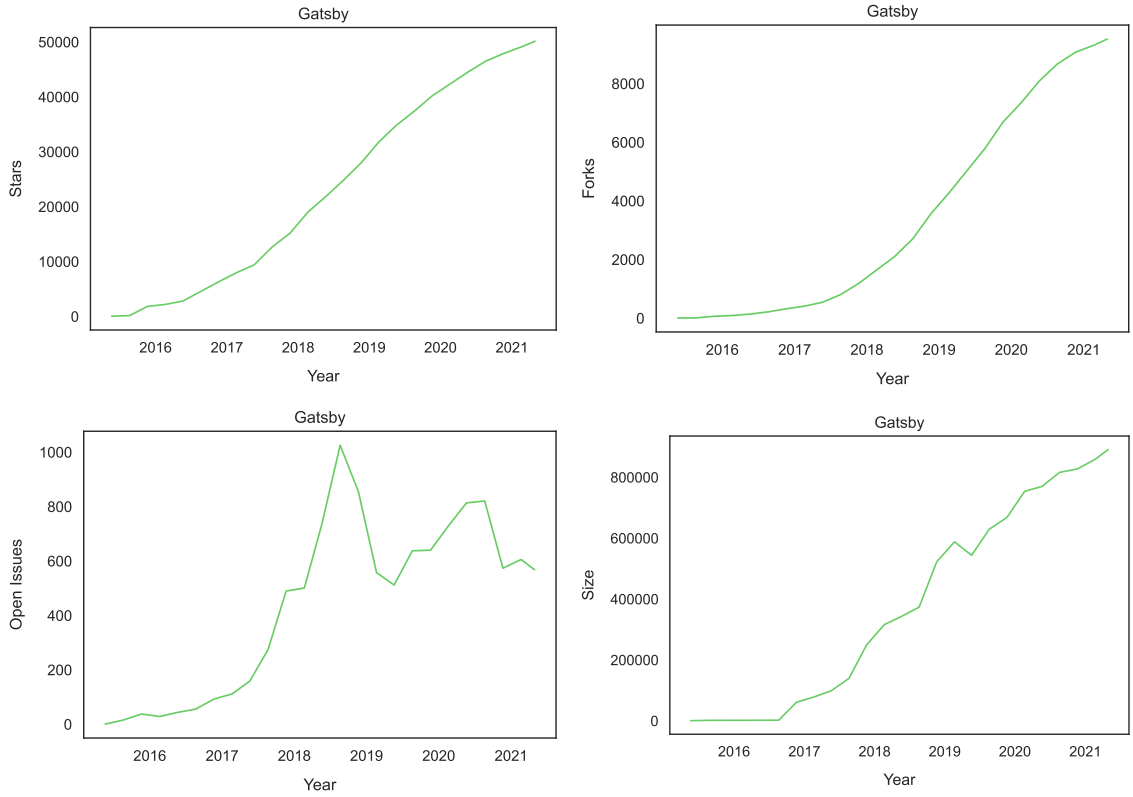

Figure B.3: Repository metrics for Gatsby

B.4 React.js

React.js is a JavaScript web framework to build user interfaces. It was first released in 2013 by Jordan Walke, and is maintained by Facebook.

Table B.4: Documentation data points analyzed for React.js

Data point	**Days since creation**
2013-11-13	107
2015-10-11	804
2016-04-01	977
2016-10-04	1163
2017-09-05	1499
2018-02-18	1665
2018-08-04	1832
2019-01-17	1998
2019-07-03	2165
2019-12-17	2332
2020-06-01	2499
2021-04-30	2832

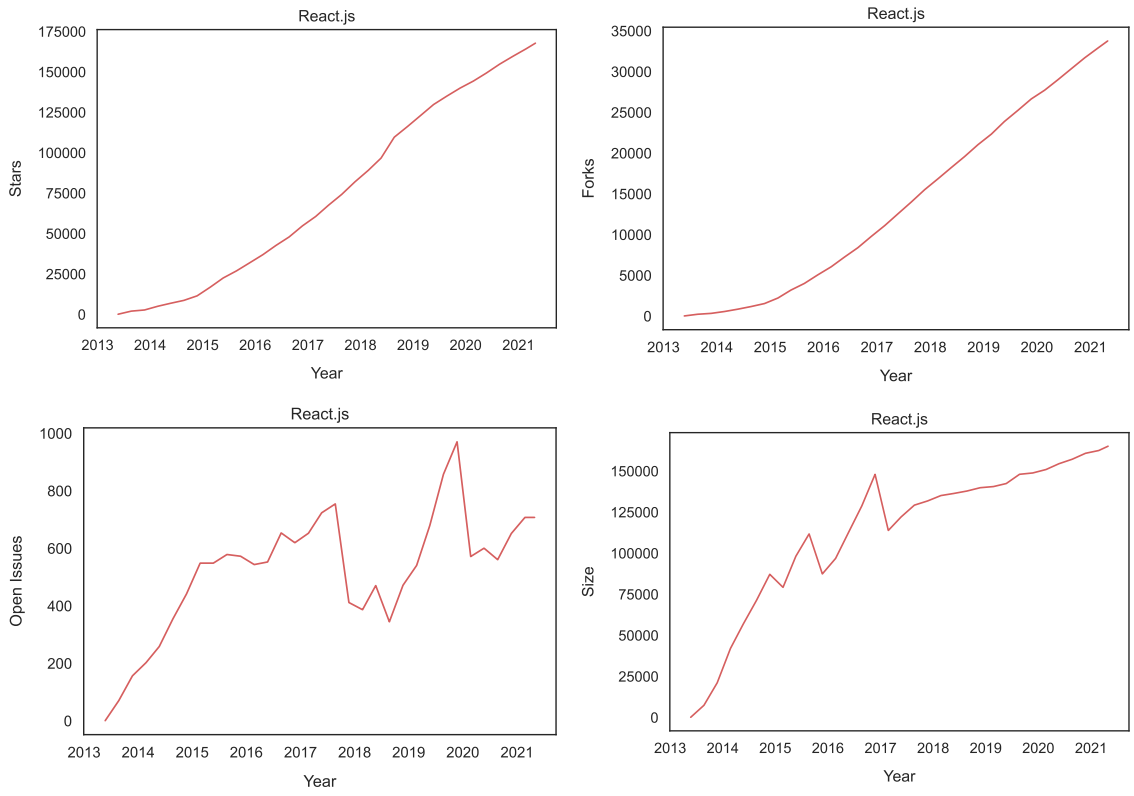

Figure B.4: Repository metrics for React.js

B.5 Vue.js

Vue.js is a JavaScript web framework to build user interfaces. It was first released in 2014 by Even You, and is maintained by himself and a dedicated team.

Table B.5: Documentation data points analyzed for Vue.js

Data point	Days since creation
2014-02-07	193
2015-01-01	521
2015-06-17	688
2015-11-27	851
2016-05-08	1014
2016-10-21	1180
2017-09-19	1513
2018-03-01	1676
2018-08-14	1842
2019-07-10	2172
2021-04-30	2832

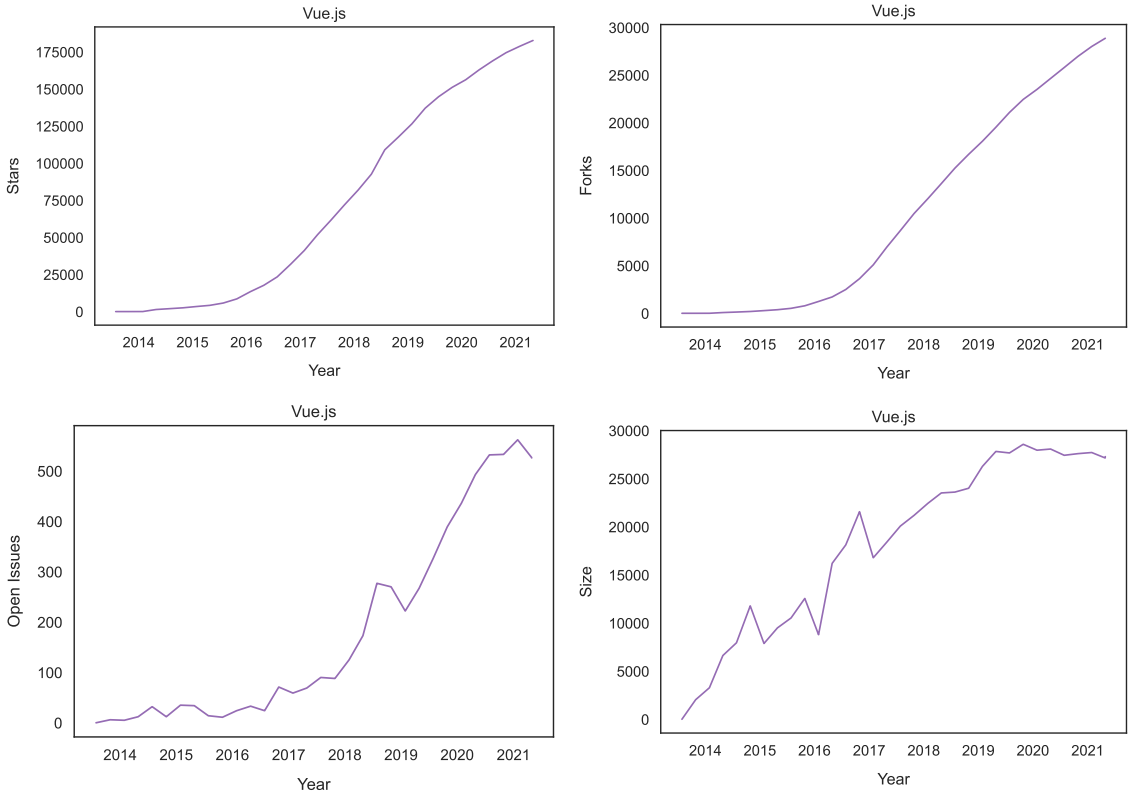

Figure B.5: Repository metrics for Vue.js