



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Minerando Discussões em Migração de Software: Um estudo da lista de emails Boost sobre evolução de código em C++

Pedro Victor Rodrigues de Carvalho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador

Prof. Dr. Rodrigo Bonifácio de Almeida

Brasília
2024

Dedicatória

Eu gostaria de dedicar este trabalho aos meus pais, Marly e Tiago. Sem dúvida alguma, este trabalho só existe pelo apoio continuado que me ofereceram durante todos esses anos em estudos e até durante uma pandemia global. Agradeço por terem me instruído e encorajado a buscar uma educação de alta qualidade, e a ter resiliência para terminar as tarefas às quais me propuser fazer.

Também gostaria de dedicá-lo aos meus avós, em especial à minha avó materna, Raimunda. Ela mostrou à geração anterior à minha a importância do estudo e da educação mesmo quando isso não era e não seria almejado amplamente por muitos anos a seguir. Sem sua visão de mundo extremamente avançada para seu tempo, certamente o presente da minha família teria sido extremamente diferente.

Agradecimentos

Eu gostaria de demarcar meus agradecimentos ao professor Rodrigo Bonifácio por me permitir trabalhar neste projeto sob sua orientação, que me permitiu adquirir muitas habilidades e conhecimentos novos. Agradeço pelo convite para assistir às aulas de estudos em aprendizado de máquina e por ter me apresentado ao grupo de estudos focados em engenharia de software.

Gostaria de agradecer também aos colegas do grupo de estudos - Alana, Ricardo e Walter - pela companhia durante esse projeto e pela colaboração em desenvolvimentos relacionados ao trabalho. Especialmente, agradeço ao Walter por ter dedicado seu tempo para me auxiliar com ajustes e instruções para uma escrita digna de publicação e por ter compartilhado de muito de seu conhecimento durante esse projeto.

Por fim, agradeço aos meus amigos de longa data - Marco, Lucas, Heitor, Jorge, Felipe, João, Victor e Wagner - pelo companheirismo e por sempre estarem comigo durante esses anos. Também agradeço aos amigos que fiz durante esse curso - Gabriel Matheus, Lucas, Pedro Augusto, Carol, Gabriel Preihs, Guilherme e Gabriel Moretto - que levarei para a vida inteira.

Meus mais sinceros agradecimentos,
Pedro Victor.

Resumo

Este trabalho apresenta uma pesquisa conduzida para investigar como o fenômeno de evolução de linguagens de programação impacta desenvolvedores de software. O lançamento acelerado de novas versões de linguagens de programação dificulta o objetivo de manter sistemas de software atualizados e modernos. Devido a isso, este estudo busca investigar como esse fenômeno afeta o desenvolvedor (ou grupo de desenvolvedores) em seu trabalho de manter o software, a fim de prover melhor entendimento sobre os desafios principais enfrentados ao desempenhar essas atividades. Investiga-se especificamente a comunidade *Boost* de desenvolvedores em C++ e como as discussões relacionadas ao tema de *Migração de Software* foram conduzidas durante o período de existência da organização. Dentre os resultados, encontrou-se a dificuldade de conciliar diferentes objetivos de maior abrangência, como o desejo de produzir inovações na linguagem e o desejo de manter código já utilizado para garantir satisfação de usuários de longa data. Também foi feita uma análise temática que apresenta os desafios mais pertinentes enfrentados pelos desenvolvedores na comunidade Boost. A pesquisa foi aprovada pelo professor orientador e submetida ao Simpósio Brasileiro de Qualidade de Software, onde encontra-se sob processo de revisão. Trabalhos futuros poderiam investigar outros grupos ou projetos, além de estudar como a estrutura dessas organizações ou grupos impacta no processo de adoção de novos padrões de linguagens de programação.

Palavras-chave: Migração de Software, Evolução de Linguagens de Programação, Boost, C++

Abstract

This document presents a research effort conducted in order to understand how programming language evolution affects software developers. The increased pace at which programming language versions are released are an obstacle to the objective of keeping a software system updated and modern. This study, then, aims to investigate how this phenomenon impacts the software developer (or group of developers) in the task of software maintenance, in order to provide insight into the key challenges faced when dealing with these tasks. Specifically, the Boost community of C++ developers is the study's focus, regarding how the discussions related to *Software Migration* were conducted during the organization's lifespan. The results suggest that there was difficulty in balancing different overarching objectives, such as the desire to produce innovation in the language and the desire to maintain already deployed software to provide reliability to long-time users. A thematic analysis was also conducted, and it presents the most prevalent challenges found by the developers at Boost. The study was approved by the supervising professor, and was submitted to the Brazilian Symposium of Software Quality. Future works could study other groups or software projects, or even investigate how the organizational structure among these groups affect the process of adhering to newer programming language standards.

Keywords: Software Migration, Programming Language Evolution, Boost, C++

Sumário

| | | |
|----------|---|-----------|
| 1 | Introdução | 1 |
| 2 | Artigo: “Mining Discussions on Software Migration: A study of the Boost mailing list regarding C++ code evolution” | 3 |
| 2.1 | Abstract | 3 |
| 2.2 | Introduction | 4 |
| 2.3 | Background and Related Work | 5 |
| 2.3.1 | Software Maintenance | 6 |
| 2.3.2 | Software Migration | 6 |
| 2.3.3 | Related Work | 7 |
| 2.4 | Study Settings | 9 |
| 2.4.1 | Data Collection and Filtering | 10 |
| 2.4.2 | Data Analysis | 12 |
| 2.5 | Analysis Results | 14 |
| 2.5.1 | Prevalence | 14 |
| 2.5.2 | Topics | 16 |
| 2.5.3 | Discussion Themes | 18 |
| 2.6 | Discussion | 23 |
| 2.6.1 | Answers to our Research Questions | 23 |
| 2.6.2 | Threats to validity | 24 |
| 2.7 | Conclusion | 25 |
| 3 | Conclusões | 26 |
| | Referências | 27 |

Lista de Figuras

| | | |
|------|--|----|
| 2.1 | Fragment of message discussing a breaking change in C++ standard requirements | 9 |
| 2.2 | Dataset distribution of messages | 11 |
| 2.3 | Steps of the analysis processes and their results | 12 |
| 2.4 | Distribution of software migration messages found by year | 15 |
| 2.5 | Percentage of software migration messages found by year | 16 |
| 2.6 | Distribution of software migration messages found by year, colored by most recent C++ standard | 17 |
| 2.7 | Percentage of software migration messages found by year, colored by most recent C++ standard | 18 |
| 2.8 | Count distribution of C++ standards mentioned | 19 |
| 2.9 | Thematic analysis results | 21 |
| 2.10 | Fragment of message about versioning and discontinuation | 22 |
| 2.11 | Fragment of message about how legacy support holds back C++ progress | 22 |
| 2.12 | Fragment of message about supporting users of older standards | 22 |
| 2.13 | Fragment of message expressing dissatisfaction with resistances to migration | 23 |

Capítulo 1

Introdução

Este documento serve de forma de apresentação do artigo produzido durante a condução das atividades que compõem Trabalho de Conclusão de Curso de Engenharia de Computação. O aluno (autor) foi responsável pela condução do estudo que levou aos métodos e resultados expostos no artigo em questão, também contando com a participação de e colaborando com colegas alunos do departamento de Ciência da Computação da Universidade de Brasília em atividades relacionadas a esse estudo.

A questão abordada pelo artigo lida com o fenômeno da evolução acelerada de linguagens de programação, que incentiva desenvolvedores a, entre outras estratégias, praticar esforços de *migração de software* para aderir a novos padrões de linguagem e utilizar *features* presentes em novas versões. Esse processo, como abordado no artigo, não é uma atividade simples para sistemas de complexidade considerável, e há uma lacuna na literatura sobre como o desenvolvedor e seu grupo percebe e lida com essas atividades. O objetivo desse trabalho é, principalmente, de entender melhor a natureza das discussões conduzidas por desenvolvedores de software sobre migração de software, relacionadas também com a evolução da linguagem C++. Entretanto, também espera-se que essa pesquisa fomente novas ideias e trabalhos relacionados a esse fenômeno de evolução de linguagens de programação, em vista de entender quais questões são mais relevantes, e projetar soluções e mudanças na intenção de melhorar a qualidade do trabalho de desenvolvedores de software.

O trabalho conduzido é, portanto, uma pesquisa sobre a percepção dos desenvolvedores de software sobre a evolução de linguagens de programação e de esforços de migração de software e como esses fatores impactam seu trabalho. A pesquisa focou no grupo *Boost* de desenvolvedores em C++, que possui um portal de comunicação entre desenvolvedores que é de acesso público - a chamada *Boost mailing list*, ou lista de mensagens do Boost. Durante a pesquisa, foram utilizadas técnicas de mineração de dados, aprendizado de máquina e processamento de linguagem natural para se filtrar um grupo de mensagens a

ser estudado. Esse grupo, contendo mensagens que foram postadas num intervalo de mais de duas décadas, foi analisado manualmente e classificado com relação aos padrões de C++ mencionados, argumentos utilizados e seu tempo de postagem. Essa análise manual foi conduzida para se encontrar temas que eram frequentemente utilizados nas discussões entre desenvolvedores, e se discutir sobre quais fatores se puseram mais relevantes, a fim de se obter um melhor entendimento dos desafios enfrentados e soluções propostas nesse âmbito de discussões. Os resultados obtidos sugerem que desenvolvedores de software costumam ter fortes diferenças de opinião sobre adoção de padrões novos de linguagem, e que as discussões possuem alto potencial de se estenderem demasiadamente, ao ponto de poderem caracterizar um fenômeno chamado *bikeshedding*. Os achados também sugerem que, além da adoção de padrões, os interesses gerais dos desenvolvedores podem ser uma grande fonte de divisão, como a incompatibilidade de objetivos entre desejar produzir código para inovações na linguagem e manter código já em uso, ainda que antigo, para garantir sua qualidade e a satisfação dos clientes que o utilizam. Por fim, também concluiu-se que essas discussões apresentam um problema que não é de solução trivial, e muitas vezes inspira propostas de mudança estrutural na organização em questão.

O artigo da pesquisa foi escrito em língua inglesa, seguindo a recomendação do Simpósio Brasileiro de Qualidade de Software (SBQS), em vista do objetivo de se submeter o documento para publicação na edição de 2024. O artigo foi submetido ao SBQS e posteriormente aceito para publicação. Uma transcrição completa encontra-se no capítulo a seguir.

Capítulo 2

Artigo: “Mining Discussions on Software Migration: A study of the Boost mailing list regarding C++ code evolution”

2.1 Abstract

Programming languages are evolving faster than ever before. New versions of mainstream programming languages (e.g., C++, Java, and JavaScript) are being released with increasing frequency, posing an elevated challenge for software developers as their systems are more easily affected by obsolescence. Software migration is far from trivial. Although there is literature on software migration methods and how developers deal with the software aging and obsolescence, little research exists on how developers perceive and are affected by rapid programming language evolution. To understand how C++ developers discuss these issues and the nature of their discussions, we mined the mailing lists of the Boost organization—one of the most important C++ open-source communities. We found that software migration is a significant concern for this community, with a lasting presence in their message boards. Furthermore, most discussions related to the challenges of the migration process, with many conflicting opinions on related matters, suggesting these issues are not easily solvable.

2.2 Introduction

Programming languages like C++, Java, and Python have been releasing new versions more frequently in recent years. For example, C++ was a “single-standard” type of language for over a decade, with only a couple of versions available to the public (in this case, C++98 and C++03, up until 2011). After this period, the C++ committee decided to make version standard releases a regular occurrence, releasing new standards every three years, starting from C++11 to C++23. This radically changed the C++ development landscape. In only a span of six years, developers had seen more new standard versions of the language released than what was available before for over a decade. This sort of practice has become common in recent years and in many different languages and applications.

In this landscape of rapid development changes, the constant passage of time means software developers and their teams are at more frequent risk of having their software become obsolete, which might lead to a decline in the system’s external qualities, affecting how it interacts with other systems and potentially rendering it obsolete in the face of competition from newer softwares [1].

To prevent or remedy this, developers might benefit from software migration techniques, such as *Source Code translation*, that involves converting source code from one programming language to another [2, 3, 4]; *GUI Migrations*, which consists of updating or transforming the graphical user interface of applications to improve user experience and compatibility with current standards [5, 6]; *Library Migration* that entails moving from one set of libraries to another, which may involve replacing outdated libraries with more modern alternatives that offer better functionality or support [7, 8]; *Source code rejuvenation*, which consists of evolving the source code of the programs through code transformations to support the new constructions of the same programming language [9].

These software migration efforts are far from trivial in most real cases and are often documented in very different manners within the literature on the subject [1], even though legacy systems are still seeing widespread usage and migration efforts are also commonplace [10, 11, 12, 13, 14].

In addition to the rapid evolution of programming languages, another pressing concern is the lack of empirical studies on how developers react to these changes.

This constant evolution can lead to challenges in maintaining existing codebases, as well as in ensuring that developers stay up-to-date with the latest advancements [11]. The scarcity of empirical research in this area leaves a gap in understanding the practical implications of these changes for professional developers. Addressing this gap is crucial for developing effective strategies to support developers in the evolving programming landscape and can provide valuable insights into the impacts of programming language

evolution on software development processes and outcomes, thereby helping to mitigate the risks associated with rapid technological change.

Previous studies mine code repositories [11, 15, 16] to understand how the migration efforts take place, but the high-level discussions had not been the focus. Therefore, in this paper, our goal is to understand how the kind of discussion related to software migration is conducted in the mailing lists of a comprehensive open-source C++ organization. More specifically, we address a specific migration effort that aims to update the version of the language (e.g., from C++11 to C++14) used in the Boost C++ libraries. To deal with this lack of research into developers' perspectives towards migration efforts, this article presents a study that mines messages from the Boost mailing list and conducts a thematic analysis to understand what their discussions on the topic presented themselves as and what influence this topic might've had on developers.

In summary, the contributions of this paper are as follows:

- **Insight into Developer Discussions and Decision-Making:** Our research provides valuable insights into how developers engage in discussions about migration efforts, particularly in the context of adopting new programming language standards like C++11. It highlights the complexities and challenges developers face, including differing opinions and the potential for extensive debates, sometimes hindered by the "bikeshedding" effect.
- **Identification of Key Challenges and Competing Interests:** Our study identifies the primary challenges associated with software migration, such as balancing the desire for technological advancement with the need to maintain code reliability for long-time users. It captures the tension between pushing for software migration and preserving stability, illustrating the competing interests within the developer community.
- **Proposal for Organizational Changes to Support Modernization:** Our findings suggest that to overcome resistance and facilitate the adoption of modern technologies, there may be a need for structural changes within organizations. This contribution underscores the importance of organizational adaptation in supporting technological evolution and ensuring that software migration efforts can proceed more smoothly.

2.3 Background and Related Work

In this section we will provide some context into the concepts used throughout this document. Additionally, we will comment on how our research effort compares to existing

literature and research, in order to provide insight into what guided our efforts.

2.3.1 Software Maintenance

Maintenance is considered one of the most important phases in a software's life cycle, significantly influencing the total cost of a system [17]. Previously associated with poor development practices, software maintenance is now regarded as evolutionary development [18]. According to ISO/IEC 12207, software maintenance activities can be categorized into Corrective Maintenance, Adaptive Maintenance, and Perfective Maintenance. Corrective maintenance aims to fix identified errors of any kind. Adaptive maintenance involves modifications when there is a change in the context to which the software must adapt. Lastly, perfective maintenance includes changes made to evolve the software to meet new user needs [19], which can involve techniques like software migration.

2.3.2 Software Migration

Software migration is a widely used term for many different applications. For our research purposes, we will base our definitions on the understanding given by Bragagnolo et. al [1]. In their research, they give extensive context into software migration and many related concepts and practices.

The main reason proposed for the necessity of modernization efforts, to which migration pertains, is given by two forms of system decline: Decadence and Obsolescence. The first one is a consequence of the continuous deterioration of inherent internal elements of the software system. This could be, for example, highly tangled code and unreliable documentation. The lack of active maintainers acting on a part of a system can be a cause for this effect. The second one, Obsolescence, is related to the ever evolving technological environment in which the system exists, and how its inherent external qualities are perceived in relation to it. New technologies that enable developers to make more efficient pieces of code, and deprecation of dependent technologies are some of the causes of obsolescence.

An important distinction to make between the two is that while Obsolescence justifies and incentivizes evolution of software systems, Decadence actively hampers it, due to the unexpected complexity in dealing with and solving its issues.

As proposed by Bragagnolo et. al, solutions for modernizing software systems can be separated into two larger categories: *Reengineering* and *Replacement*. The latter regards efforts that require abandoning legacy systems and establishing new ones in place of those. Our research is focused on the Reengineering category, which is defined as "all processes based on modifying a previously existing system". This category encompasses

Modernization and Renovation efforts, which are the types of efforts our research is focused on.

The researchers distinguish between modernization efforts, aimed at recovering a system from obsolescence, and renovation processes, which address system decadence. Within these broader categories, they identify specific subcategories that are the focus of our study: Adaptation, which involves updating a system to use new technological environments without abandoning current technologies; Migration, which entails moving a system from one technological environment to another that is mutually exclusive with the original; and Restructuring, which focuses on source code operations to enhance the quality and understanding of the system's internal structure.

Our interest resides mostly on investigating how developers faced the challenge of adopting different C++ standards, especially newer ones. Being a community that develops C++ libraries that often use each other's utilities, any library developer needs to consider other libraries' usage of their own before making a breaking change in their code, like seen in Figure 2.1. This, within the Boost development environment, can pertain to Adaptation, Migration and Restructuring.

From here onwards, we will refer to these categories solely as "Software Migration" for the sake of simplicity in reporting our findings. We considered these three specific categories to compose our whole focus when looking for developers discussing migration efforts.

2.3.3 Related Work

The work detailed in this document is similar in nature to other research already done in the field of mailing list/internet forum thread mining. Additionally, we consider our work similar to other work in the field of source code rejuvenation studies, due to similarities and overlap between code rejuvenation and software migration.

Lucas et al[11] conducted a study on how KDE contributors practice rejuvenation in their projects. The study focused on specific features contained in C++11, C++14 and C++17, and how developers utilized them. The analysis was conducted in 272 KDE programs and libraries written in C++ which projects had started after 2010 and had had at least one commit made in 2022, so as to focus on changes on currently maintained code related to the features contained in post-C++11 versions. The research employed an automated method of detecting increases in use of modern C++ features while the number of actual statements remained constant, in order to determine at what points in time the project could've likely applied a rejuvenation effort. The results indicated that the reasoning was often related to improving readability and conciseness of the code, as well as slowing software aging and attracting new contributors to the projects. It's

also suggested that the benefits of rejuvenation are perceived by developers, and that programming language designers would benefit from knowing about how modern features are embraced by developers.

We referenced this study to set out a general guideline of what to look for regarding the intricacies and importance of software migration, not only to the performance and usage of the software in question, but to whoever develops, maintains and manages it. In general, our aim is to understand the main reasons as to why a migration effort is or is not applied. Additionally, our focus on the Boost community is similar to their sole focus on the KDE community, in our case mainly due to the easily accessible body of data contained in their mailing list and their reputation involving C++ development throughout decades.

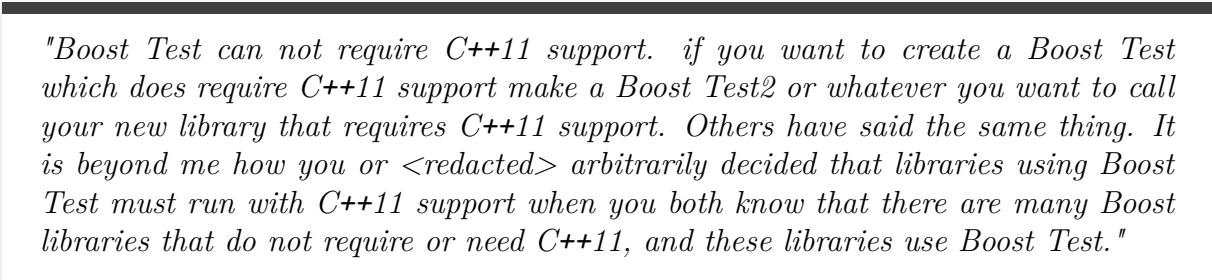
Swillus et al[15] conducted an analysis of Stack Overflow posts regarding software testing. While also focusing on a widely applied and influential topic on software development, the analysis focused on how this practice is perceived through the developer's perspective. For this, the Stack Overflow messages were mined and subsequently filtered through a semi-automated method and then by a systematic qualitative data analysis. The strategy employed for the manual data analysis was based on the Socio-Technical Grounded Theory[20]. Their results suggest that software testing motivation increased with the complexity of the project, and that the practice itself is seen as something to aspire to. We referenced this research in order to employ a somewhat similar manual analysis method focusing on broader interpretative topics.

Tahir et al[16] conducted a similar style of research investigating how developers at Stack Overflow, Stack Exchange and Code Review discussed code smells and anti-patterns. Their results suggest that developers don't have a clear definition of code smells and anti-patterns, yet they seem to have negative feelings towards these practices in general. Similarly to the previous research mentioned, we also referenced their work for employing a similar thematic-focused approach.

In our study, we aim to distinguish our objectives from previous works by focusing specifically on discussions about software migration within the Boost community. While related studies often utilize data mining to collect datasets for analysis, our approach emphasizes understanding the context of how developers respond to the rapid evolution of programming languages. Unlike a comprehensive search for all migration-related discussions, our goal is to capture a broad spectrum of these discussions to analyze developers' reactions and decision-making processes.

2.4 Study Settings

This research aims to comprehend the themes that emerged from the Boost mailing list discussions, particularly in relation to software migration concepts. Boost is one of the main C++ open-source organizations, significantly contributing to the evolution of the C++ language specification. Currently, hundreds of C++ developers contribute to the implementation of Boost libraries, which are widely used and often integrated into the C++ standard library. Indeed, the success of some Boost libraries can be partially attributed to a formal review process that relies on mailing list discussions. Considering that the Boost foundation contributes to the evolution of C++ standards, it is expected that Boost libraries would be up-to-date with new versions of the language. However, migrating a Boost library to a new version of the C++ programming language is anything but trivial and should be carefully discussed among Boost members. Figure 2.1 shows a snippet of a message from the Boost mailing list on this subject. These careful discussions are necessary because migrating a library might not only lead to (unexpected) breaking changes but also require the clients of that library to update their C++ code, dependencies, compilers, and tools as well.



"Boost Test can not require C++11 support. if you want to create a Boost Test which does require C++11 support make a Boost Test2 or whatever you want to call your new library that requires C++11 support. Others have said the same thing. It is beyond me how you or <redacted> arbitrarily decided that libraries using Boost Test must run with C++11 support when you both know that there are many Boost libraries that do not require or need C++11, and these libraries use Boost Test."

Figura 2.1: Fragment of message discussing a breaking change in C++ standard requirements

In line with this objective, we devised two main questions that summarize what we want to understand about this topic, and drove our research methods in order to answer them. The questions are as follows:

1. **What is the prevalence of software migration discussions in the Boost mailing list?**
2. **What was the nature of the discussion's contents?**

By answering the first research question, we aim to measure the extent to which software migration is present in the Boost mailing list data. Since programming language evolution is a continuous phenomenon, we hypothesize that it inevitably impacts software development. Furthermore, we intend to investigate the patterns regarding the frequency

and timing of these discussions and determine the specific topics addressed in messages about software migration. By answering the second research question, we hope to identify the motivations and challenges associated with upgrading a large codebase of C++ libraries to a new version of the language standard. We believe that some of the motivations and challenges related to software migration that we find in Boost might also occur in other open-source organizations.

2.4.1 Data Collection and Filtering

Our focus is on the Boost community of developers, and the most readily available data for our research is contained in their public message board: the *Boost MailMan Archive*¹ (referred to hereafter as the *Boost mailing list*). This archive is a message board hosted on a static HTML website, making it easy to collect data using a web scraper [21]. We built a scraping application² that goes through every post in the mailing list and captures key information such as the date of posting, author name, post title, and message body. The scraper stores the collected data in a relational database for later access. In March 2023, we collected all the data available on the Boost mailing list at the time: a total of 253,548 messages, spanning more than two decades of mailing list history (from February 1998 to March 2023).

To better handle the volume of data collected, we decided to employ an automated effort to reduce the scope of our manual data analysis procedures. For that, our decision was to explore the Support Vector Machine (SVM) algorithm [22] to classify the messages that were more likely to contain software migration discussion.

SVMs require a labeled dataset for both training and testing of the model. Due to this necessity, we employed a manual search effort in order to find a reasonable amount of messages we deemed relevant to the topic. We went through many different points in the mailing list’s history and searched through the threads for possible software migration discussion. Once we found a message containing this topic, we stored its URL and relevant information in a spreadsheet for future reference. With this effort, we found 58 messages discussing software migration, of which 32 were used to train the model and the remaining 26 were used for testing.

Figure 2.2 shows a graphic visualization for how the messages were organized in the following process. The training dataset was composed of 32 manually selected messages and 300 randomly selected messages from the mailing list. The manually selected messages were labeled as “*containing software migration*” and the 300 randomly selected were labeled “*not containing software migration*”. Similarly, the testing dataset was composed

¹<https://lists.boost.org/Archives/boost/>

²Blind review

of 26 manually selected messages and 20 randomly selected from the mailing list. They were labeled the same way as the training dataset.

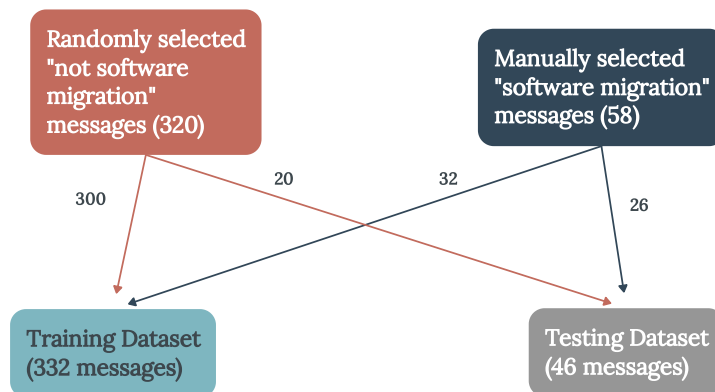


Figura 2.2: Dataset distribution of messages

First, our code performs the vectorization of the training dataset, using a TF-IDF [23] transformation to more accurately represent relevant tokens in the analysis. We employed a Natural Language Processing [24] approach called *Bag of Words* [25] for the vectorization step of our process. Afterwards, the vectorized data is given as input to the learning algorithm. This results in a classifier model that we can leverage to classify new messages.

To test the resulting model, the testing dataset is vectorized in the same way as the training dataset, then we use the model to classify this new dataset’s entries. Since this dataset was previously classified by ourselves, we can measure the model’s performance with metrics such as precision and recall. We obtained a score of 71% precision and 100% recall for messages “*not containing software migration*”. We also got 100% precision and 69% recall for messages “*containing software migration*”.

Our objective with this step was to find more messages “*containing software migration*” than what we already had, employing an automated approach. For that, 69% recall in a small corpus of 46 test messages was likely to return enough true positives when applied to the full dataset of 250 thousand messages. In light of this, we considered this model acceptable to search for more migration-related messages. This also means, however, that this effort will not be an exhaustive search, due to the low amount recalled.

Running the whole mailing list dataset through the classifier model returned 967 messages labeled as “*containing software migration*”. This body of messages was used as a basis for the manual analysis processes that followed.

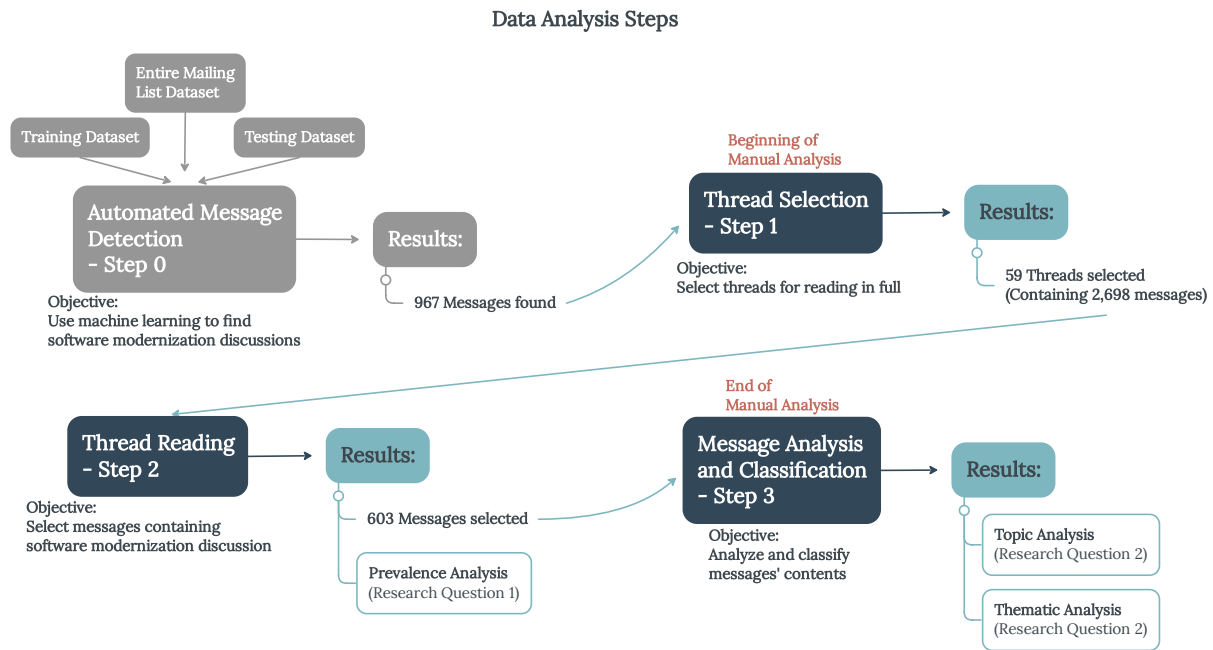


Figura 2.3: Steps of the analysis processes and their results

2.4.2 Data Analysis

With the results from the classifier available, we decided to not limit ourselves only to the 967 messages we got back from the model. We would use these results as pointers towards where to look for more relevant messages, with the intention to find more than what was already found by the classifier. For that, we took the following steps:

1. Thread Selection

Look through every message found by the classifier and analyze the thread to which they belong to. Determine if this thread may have more migration-related messages and if so, note them down in a spreadsheet for future reading.

2. Thread Reading

Having a list of selected threads from the previous step, read through every message contained in each of them and note down any message that contains software migration discussion. This step provides results for answering research question (1).

3. Message Analysis and Classification

This step is taken as soon as a message is selected in the *thread reading* step. We analyze the message on its context, noting down what topics it mentions and what arguments it uses. This step is fundamental for answering research questions (1) and (2).

These steps were devised as we matured the research ideas, and they seek to provide a more structured approach and improve the overall scope reached by the analysis.

The objective of the "*thread selection*" step is to increase the scope of analysis in a guided manner so we don't rely solely on an automated analysis for our results. By looking into whole threads, beyond increasing our understanding of the context for each message (which improves our analysis capabilities), we also reduce our dependence on the classifier's results, for we may be able to detect and select more true positive messages and disregard false positives. This is true especially considering nuanced messages, which aren't easy to classify correctly in this automated method we used.

The "*thread reading*" step shares the same objectives, and additionally, by reading the threads in full, we do an exhaustive search for software migration messages inside these selected threads.

The final step, "*message analysis and classification*", has the intent of quantifying and recording our observations on the discussion's contents. Additionally, by noting down the arguments used, we devise a list containing recurrent arguments that allows us to see what are the most popular ones, and afterwards use the arguments list and count to construct a thematic analysis approach to understanding the discussions.

Following is an explanation of our body of data under study during parts of the process. A diagram of this information is present in Figure 2.3.

After *thread selection* we had a list of 62 threads that we selected for further analysis. Adding up the number of messages contained in each thread, we had a set of 2,920 messages to read through. However, during *thread reading* we noted that three threads were not related to software migration, instead they talked about an almost opposite, yet related topic: code backporting. Those threads were excluded from our set, and that left us with 2,698 messages in 59 threads.

After *thread reading* and *message analysis and classification* we had selected 603 messages deemed as "*containing software migration discussion*". Each one of these was analyzed and classified regarding the topics mentioned and arguments used.

The final analysis efforts were to sift through all our collected data and classifications and interpret the data in order to answer research questions. For research question (1), regarding prevalence, we looked at the total amount of messages and what distribution of yearly posting they represented. Again for research question (1), now regarding topics of discussion, we looked at topic occurrence counts and overall topic prevalence. For research question (2) we looked at argument count and argument meaning, ultimately grouping arguments into *themes*. These themes allowed us to make a more human analysis of the discussion's contents.

2.5 Analysis Results

The analysis was guided by the previously mentioned research questions. The study process was very much a flexible and evolving effort, and therefore, there are many data points related to the analysis process itself that could be discussed along with the results. For the purposes of brevity and objectiveness, this section will focus mainly on the final results.

2.5.1 Prevalence

To answer the first part of research question (1), we must examine the number of messages we found that discuss software migration. Through our research method, we found 603 messages that were considered to contain software migration discussion in the Boost mailing list. **This represents 0.24% of the whole mailing list.** Since this was not an exhaustive search for those messages, we will consider this, at most, a lower bound.

Analyzing further, from these 603 messages, we found the distribution of occurrences present in Figure 2.4. Derived from this distribution, in Figure 2.5, each bar indicates what percentage those messages represented of the total number of messages posted in that year. These distributions present an average of 23.2 migration messages per year and 0.48% presence per year.

From manual analysis, we determined that the discussions before and after 2011 were fundamentally different in nature. Before 2011, migration discussions were basically only about compiler support. The messages from 2011 onwards are not only more frequent but rarely revolve around compilers—the new migration goals were then defined mostly by C++ standard versions.

Analyzing the same data but limiting the scope to 2011 onwards, we get an average of 43.2 messages per year and an average of 0.94% presence per year.

Moreover, from 2011 onwards the total number of messages posted in the mailing list is 80,043, which is significantly smaller, less than one third of the whole dataset. However, considering the estimates calculated previously, the number of messages selected manually drops only to 562, which represents **a lower bound of 0.7% of presence after 2011.**

These message distributions, however, pose a contradicting trend. Even though total code migration-related message quantities declined after 2018, the percentage those messages represent from the year's total have an increasing trend.

This might be explained by an overall trend of reduced usage of the Boost mailing list³. While reading threads manually, we encountered messages where developers discussed why the mailing list was seeing a decline in usage, and the consensus seemed to be that most

³<https://bit.ly/boost-reduced-usage-trend>

Software migration messages selected per year

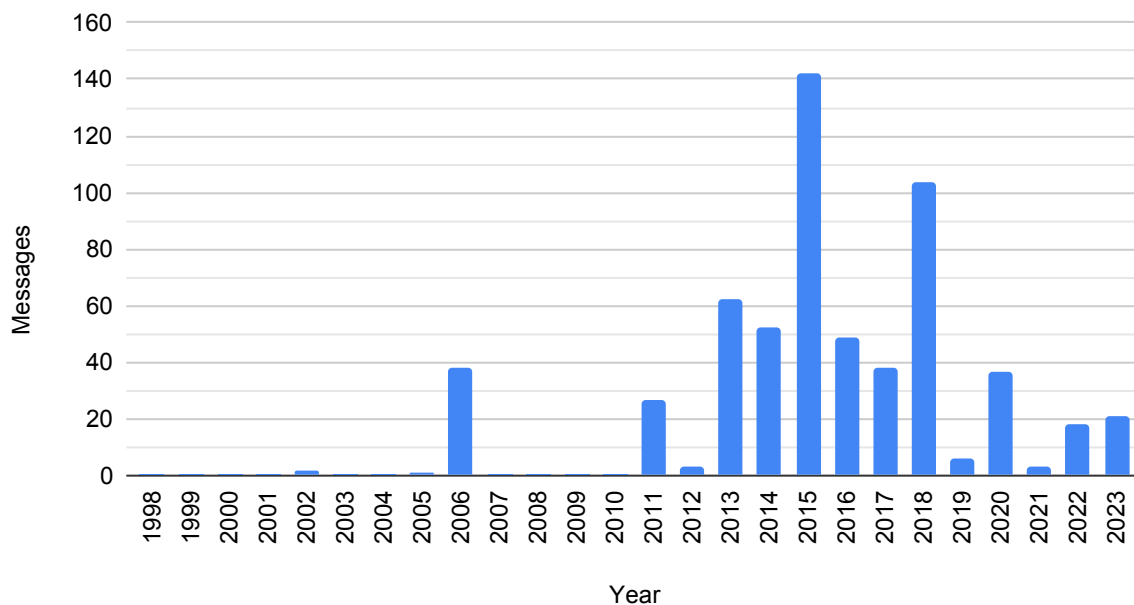


Figura 2.4: Distribution of software migration messages found by year

discussions regarding Boost’s development of individual libraries was being conducted in portals other than the mailing list, like *Github* and *Slack*. This observation might also explain the dramatic reduction in the total number of messages in the mailing list from 2011 onwards.

We can safely conclude, then, that **software migration discussion increased in presence even though the mailing list saw decrease in usage.**

Furthermore, in Figures 2.6 and 2.7 we can clearly see peaks in 2015 and 2018. In these figures, the red color represents C++14 as the most recent standard released that year, and the green color represents C++17. We can see those peaks in 2015 and 2018 seem to align with C++ releases with a one year delay. This does not happen equally for C++11 (yellow) nor C++20 (blue) however, although there are peaks in their respective sections.

This might suggest a correlation between increased occurrence of software migration discussion and C++ standard releases. The one-year delay observed can be explained due to resistance in adhering to new standards as soon as they’re released, either due to bugs in the standard or low expected immediate adherence from C++ developers. Similar results were found in the KDE community [11] where researchers observed that developers take, on average, three to five years to adopt modern features from C++11 onwards.

Software migration discussion (%) per year

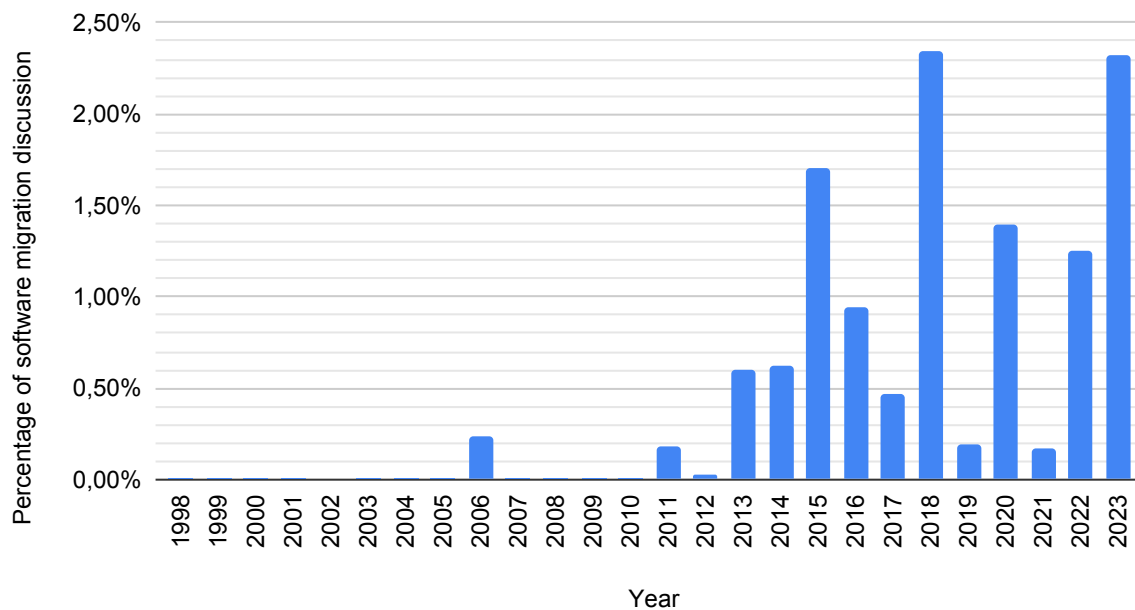


Figura 2.5: Percentage of software migration messages found by year

2.5.2 Topics

To answer the second point of interest in research question (1), we derived a list of topics that were recurring during the manual analysis. We found three main topics:

- **C++ Standard Support**

Discussions related to dropping or keeping support (migrating or not) to specific C++ standards. **This category was the most prevalent one, especially after 2011, being present in 403 messages (66.8% of total).** The standards included were: *C++98*, *C++03*, *C++11*, *C++14*, *C++17* and *C++20*.

- **Compiler Support**

Discussions related to dropping or keeping support (migrating or not) to specific C++ compilers or compiler families. This category was more prevalent in messages before 2011, and was found in 118 messages (19.5% of total).

- **General Topics**

These are topics not entirely related to software migration, but still related to modernization as a whole. This category was found in 135 messages (22.4% of total). It includes:

Software migration messages selected per year

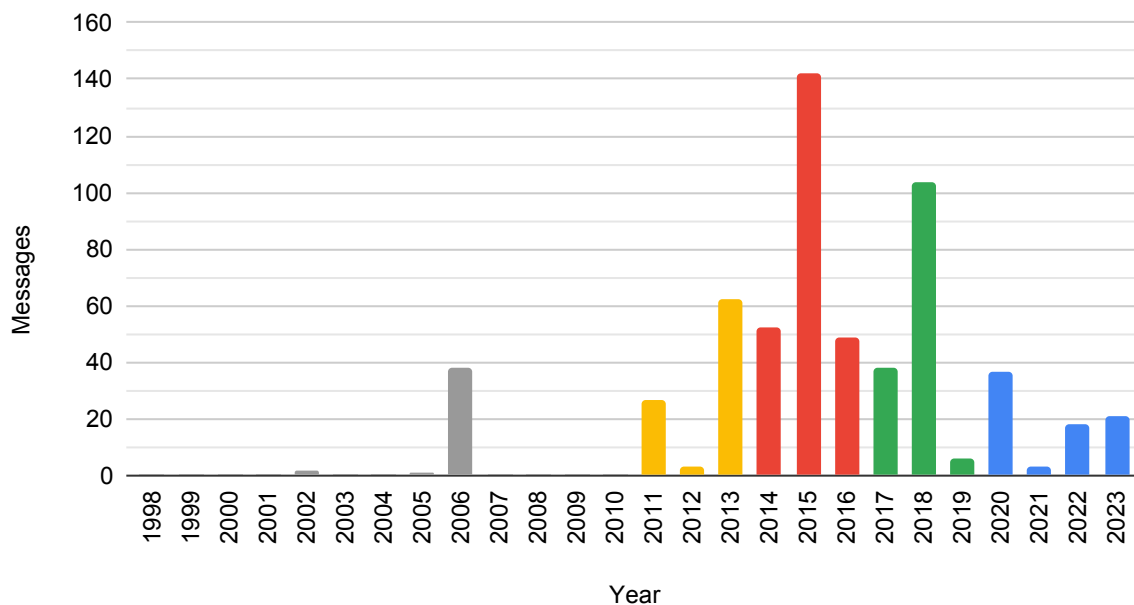


Figure 2.6: Distribution of software migration messages found by year, colored by most recent C++ standard

- *Organizational Changes*

Discussions regarding management of the Boost Library development community, containing topics such as autonomy for breaking changes and modularity of the release.

- *Non-Specific Opinion*

These are messages that don't argue on specific standards or compilers, nor talk about the management of the group. Instead, they simply show approval or refusal for previously proposed migration and modernization efforts.

Diving further into the most prevalent topic (*C++ Standard Support*), we analyzed what specific standards were most mentioned. As can be seen in Figure 2.8, the most mentioned C++ standards were C++11 and C++03, by a large margin. **C++11 was mentioned in 307 (50.9%) of the 603 messages analyzed, and C++03 was mentioned in 243 (40.3%) of them.** All other C++ standards had less than 100 mentions (around 15% presence) each.

While analyzing the mentions of each of these C++ standards, we encountered similarities in C++03 and C++11's occurrence distributions, which corroborated manual analysis experience. This inspired a further investigation of both standards in the discussions. We analyzed how many mentions of C++03 were accompanied by C++11, and found they were

Software migration discussion (%) per year

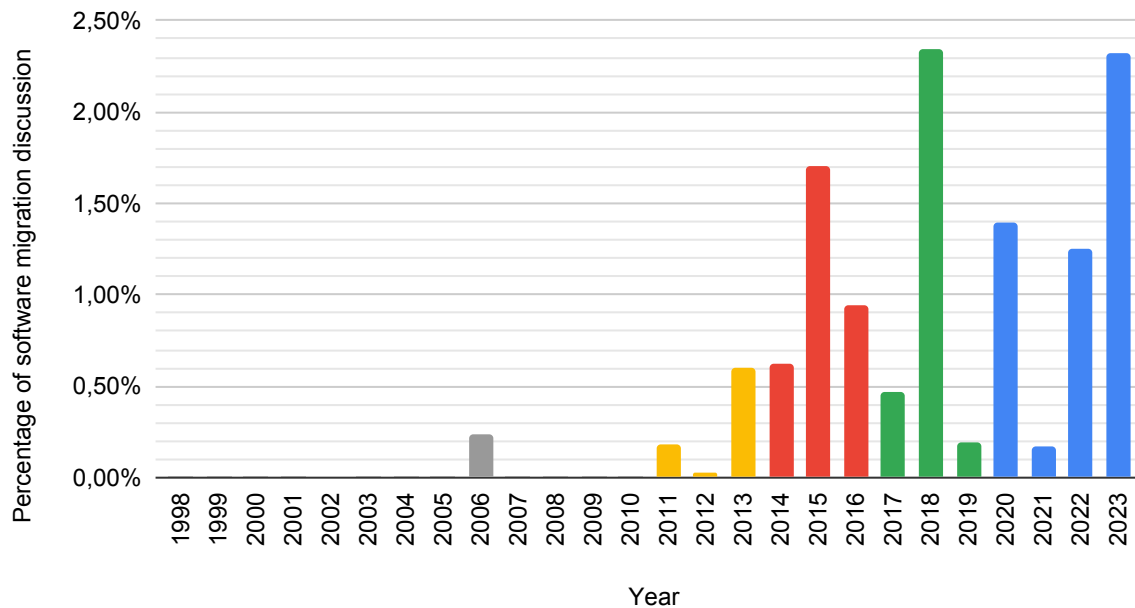


Figura 2.7: Percentage of software migration messages found by year, colored by most recent C++ standard

mentioned together in 188 messages. This means that 77.4% of C++03's occurrences were alongside C++11, while 61.2% of C++11's occurrences were alongside C++03. **This suggests C++03 and C++11 were deeply related topics and were mostly mentioned together in these discussions.**

We decided to compute every mutual occurrence of C++ standards and found the only strong relation between two topics was between C++03 and C++11. Other topics did get mentioned some amount with each other, but analyzing the proportion from both sides (as was done for C++03 and C++11) showed other pairs had an imbalanced relation. This suggests one topic often appeared alongside another, but not vice-versa.

All of this suggests the Boost mailing list talked mostly about C++ standards in discussions regarding software migration. The two most mentioned topics were C++11 and C++03, which were majoritarily mentioned together.

2.5.3 Discussion Themes

To answer research question (2), we started by noting down recurring argument types contained within the analyzed messages. This list was updated throughout the manual analysis process as we saw new arguments be employed and as we resigned an argument

Distribution of C++ standards mentioned

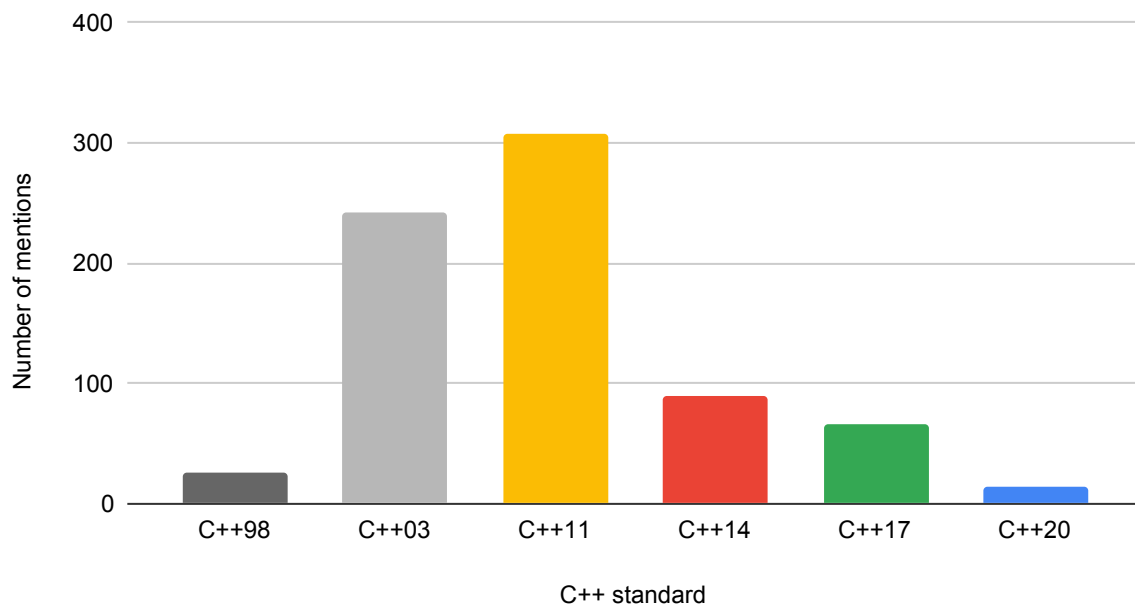


Figura 2.8: Count distribution of C++ standards mentioned

classification when needed. Following this process, we derived themes from these argument categories, taking into account their overall similarities, grouping and functionality.

In the end, we found three main themes related to software migration. Following is a description of the themes, along with subthemes and the amount of messages they were found in. The same themes are seen in Figure 2.9.

- **Drivers (23.0% presence)**

Reasons developers presented for wanting to migrate, or arguing the community should migrate to newer versions of C++ standards.

- *Desire to invest in C++ innovation* (83)
- *Cost of maintaining support of older standards* (74)
- *Arguing older standards are obsolete* (19)
- *Optimism with userbase's adoption of newer standards* (18)
- *Desire to support users of newer standards* (15)

- **Deterrents (16.5% presence)**

The opposite of the drivers, these are reasons developers presented for not wanting to migrate, arguing the community shouldn't migrate or not being able to migrate to newer C++ standards.

- *Desire to support users of older standards* (85)
- *Cost of updating to newer standards* (29)
- *Pessimism with userbase's adoption of newer standards* (24)
- *Claiming self incapable of updating to newer standards* (9)
- *Problem / Bug in newer standards* (3)

- **Challenges (60.5% presence)**

Every type of struggle or conflict found to be the result of the division between developers who want to migrate and developers who didn't want to migrate to newer C++ standards.

- *Conflicting opinions on discontinuing support of older standards* (220)
- *Conflicting opinions on software versioning* (104)
- *Inter-library dependence problems* (73)
- *Conflicting opinions on retrocompatibility* (60)
- *Modernizing on an organizational level* (55)
- *Conflicts between developers in the community* (38)

From the thematic analysis we can easily see that **the majority (more than 60%) of discussion regarding software migration was centered around its challenges.** More specifically, the majority of what was said regarded calls for either discontinuing support for older C++ standards or keeping it, followed by discussions on versioning. The manual analysis revealed that, in this context, versioning was often related to the discontinuation issue (like in the example in Figure 2.10), either as a way of keeping multiple versions of a library in different C++ standards or discussing the whole library set release as a whole.

Furthermore, **many of the subthemes found are mirrors of each other.** That is, the "*Drivers*" and the "*Deterrents*" categories are fundamentally opposed, and many of their characteristics mirrored each other, having the same type of argument, only arguing for opposite reasons. One example of this is the "*Cost of maintaining support of older standards*" and "*Cost of updating to newer standards*" pair of subthemes, which express the difficulty of interacting with a given C++ standard, but one from a perspective of facilitating migration and another from a perspective of opposing it. This can be seen in the "Challenges" category as well, as all subthemes defined as "conflicting opinions" expressed ideas both in favor of and against migration-related concepts.

The most popular arguments driving software migration reveal that **there was an interest in spearheading C++ development, and that keeping support of older**

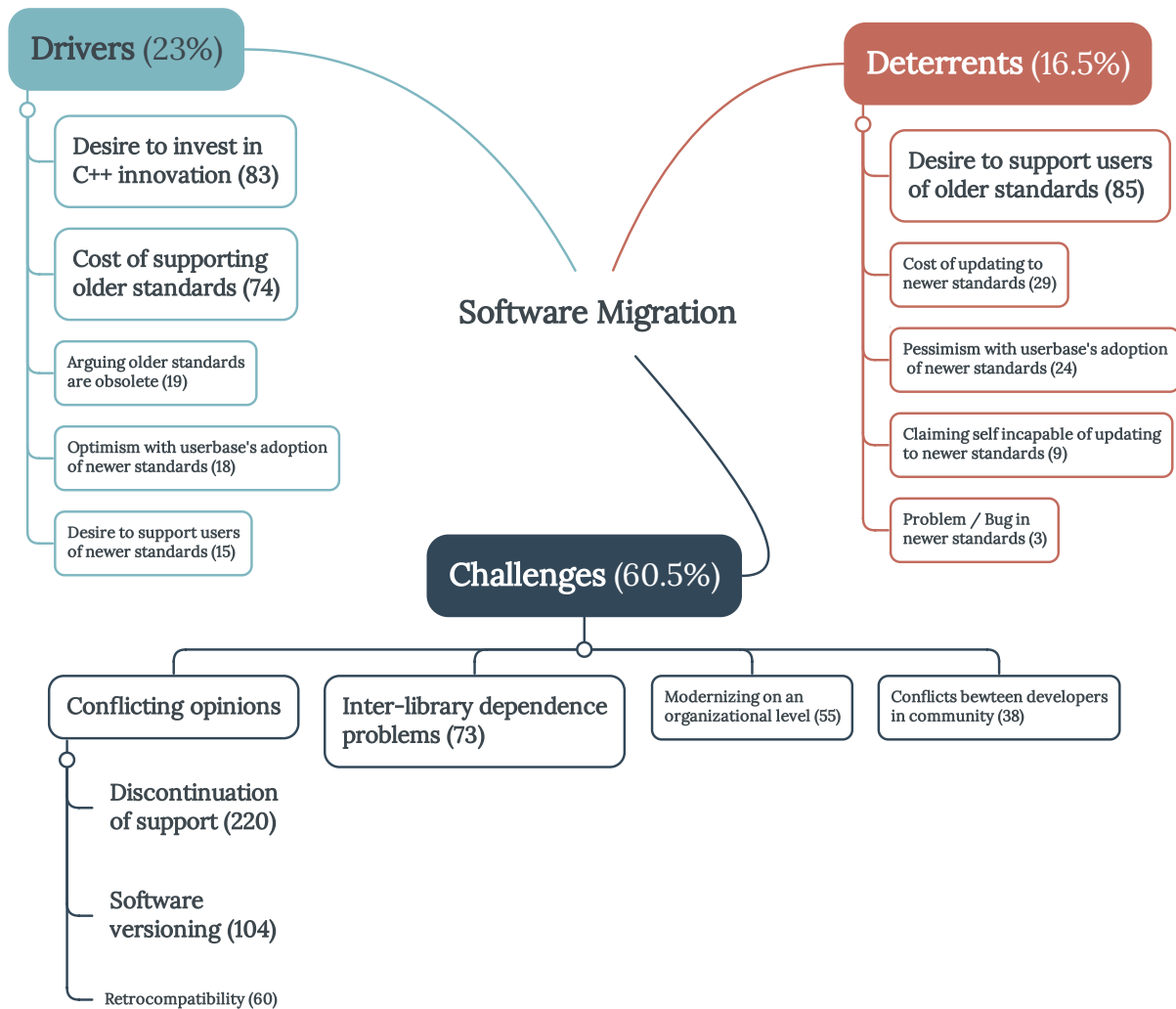


Figura 2.9: Thematic analysis results

C++ standards holds this progress back. This is corroborated by manual analysis experience, in which we saw a call for returning Boost to its old reputation as C++ pioneers. This sort of discussion often led to discussing Boost’s general goals as an organization, like in Figure 2.11, usually in very lengthy threads.

On the other hand, the most popular arguments deterring software migration reveal that many developers think that it is very important to keep support for the userbase the Boost libraries already have - considering users of older standards, often C++03. This, coupled with the opinion that updating to newer C++ standards is costly and troublesome, reveals that **some developers were more focused in keeping the reliability of Boost’s libraries and services to those who already use it.** This can be seen in the example in Figure 2.12.

As "*Driver*" arguments were seen 40% more than "*Deterrent*" arguments, we see an imbalance in presence in favor of migrating to newer C++ standards. However, **the amount**

"As <redacted> suggested so many years ago now, it's long overdue for a separate Boost v2.x release which is shorn of the backwards compatibility and undermaintained libraries. (...)"

Figura 2.10: Fragment of message about versioning and discontinuation

"Compiler vendors are not going to focus on improving legacy support for outdated standards. Why should Boost? (...) The message should be, 'upgrade your compiler, or get out of the way. You're holding everyone else up with your buggy old non-conforming compiler.' (...) C++ needs to move forward. Slavish devotion to backwards compatibility is a burden it can not afford."

Figura 2.11: Fragment of message about how legacy support holds back C++ progress

of discussion mentioning "Challenges" paints software migration as an extensively debated issue, which means it likely was not easily solved or quick to reach an agreement to. This corroborates the manual analysis experience, in which we saw many lengthy discussions over migration efforts that often branched out in many directions (once even mentioning and discussing the *bikeshedding* concept⁴).

In general, it seemed that Boost's overall monolithic approach to releasing its libraries often frustrated developers who wanted to update existing libraries to newer standards. This extensive and difficult to solve issue tended to lead whoever is proposing breaking changes to either be negated permission or to be frustrated with delays and resistance. This was expressed in the *"Conflicts between developers in the community"* subtheme, in which, alongside questioning the validity and relevance of each other's motivations for proposing or opposing software migration, developers expressed dissatisfaction and cynicism regarding the overall resistance to migrating that was offered to their proposals, like shown in Figure 2.13. This sentiment often led to proposals of changes in organizational aspects, such as making a more modular release process or changing how strictly policies regarded "breaking changes", such as migrating C++ standards, which is what is represented in the challenge of *"Modernizing on an organizational level"*.

⁴<https://lists.boost.org/Archives/boost//2013/10/207482.php>

"Note also that many corporate users cannot yet switch to C++11 for many reasons (...) arguing that users can use older versions of Boost is unhelpful since that denies those users access to new libraries, or fixes to current libraries, even if those libraries support C++03. (...)"

Figura 2.12: Fragment of message about supporting users of older standards

*"(...) I'll freely admit I have given up on trying to make any substantial changes to Boost. I prototyped (...) a Boost-lite transition layer suitable for a clean Boost fork (...). Nobody was interested. The community *likes* things just the way they are: serving the Boost community, and to hell with the entire C++ community. A shame, and a waste, and I suspect in the long term self defeating."*

Figura 2.13: Fragment of message expressing dissatisfaction with resistances to migration

2.6 Discussion

In this section we provide answers to our research questions and then highlight possible decisions of our research that might threaten the validity of our work.

2.6.1 Answers to our Research Questions

The following were our answers to our proposed research questions. They give an overview of what we observed through our assessments.

Research question (1): What is the prevalence of software migration discussions in the Boost mailing list?

The messages we found that were considered as containing software migration discussion represented 0.24% of the entire message quantity in the mailing list from February of 1998 up until May of 2023. The discussions were much more prevalent after 2011, in which period they represented 0.7% of the whole mailing list. During this period, the discussions averaged 43.2 messages a year and also averaged 0.94% presence in messages posted in each year. We also found that software migration discussion increased in presence even though the mailing list saw a decrease in usage. Furthermore, we found a correlation between the increase in software migration discussion and C++ standard releases.

The most discussed topic was *"C++ Standard Support"*. It was present in over two thirds of all messages analyzed, being in 403 of them. Of the C++ standards mentioned, C++11 was mentioned in 50.9% of messages and C++03 was mentioned in 40.3%, making these the most discussed standards. Other standards had less than 15% presence each. We also found that C++03 and C++11 were mentioned together more often than not.

Research question: (2) What was the nature of the discussion's contents?

We found that discussions revolved around three main overarching themes: *"Drivers towards software migration"*, *"Deterrents to software migration"* and *"Challenges of software migration"*. The majority of the discussion (*over 60%*) mentioned the challenges of migrating. This, even though there was an imbalanced presence in favor of drivers over deterrents, suggests that software migration was an extensively debated issue, and likely had no easy solution or quick agreement reached between developers in the community.

This even goes as far as causing conflicts between developers in discussions, in which they expressed dissatisfaction and cynicism regarding resistance to migrating. These sentiments often led to proposals of changes on organizational levels, in order to facilitate migration and reduce the impact of challenges presented.

Regarding the other themes, the drivers revealed that some developers held an interest in innovating in C++ development, which was hindered by the continued support of older C++ standards. The deterrents, on the other hand, showed some developers were more interested in keeping the reliability of Boost’s libraries and services to those who already use it. Between these two themes, we also saw mirrored arguments that essentially presented the same point, but from opposing perspectives.

2.6.2 Threats to validity

Throughout the study, we believe our methods were successful in facilitating and reaching our objectives. However, there are some points that probably impact our results with biases coming from our study design decisions.

Our choices to focus our analysis on the Boost community and to reduce our scope solely to their public mailing list present an important point to consider regarding the *transferrability* of our findings. That is, our results speak to how a subsection of Boost’s message exchanges presented discussions on software migration, and while it may represent fairly how the organization behaved regarding these efforts and challenges, the same may not reasonably apply to other developer groups. We believe these results may be, within reason, extended to other open-source communities of developers, however, we cannot say the same to closed-source projects and groups whose projects differ considerably from the type of software the Boost Libraries are. Moreover, our choice to analyze only the mailing list represents a risk of representativeness of Boost itself, as we learned that many development-specific discussions moved from the mailing list to other channels like Slack and Github. We argue, however, that our choice was reasonable enough given that these other channels are not publicly displayed by Boost, nor are of as easy outside access as the mailing list is.

Regarding our automated search methods, our initial effort for composing a training and testing dataset was a crucial step on the whole research. We acknowledge that these datasets were small and could’ve been *biased* towards messages that mentioned C++ standards directly. Our manual analysis steps that increased the scope of search were an attempt to mitigate how impactful this would be to our results. This factor probably impacted how our messages found were distributed in time, being much more prevalent after 2011. However, we argue that the focus on this section of time represents more

clearly modern issues with software migration, and therefore are still very much relevant to the recent increased pace of programming language evolution.

Finally, it's important to consider that, when conducting this type of interpretative research effort, there is always a risk of compromising *neutrality* regarding the results. We cannot eliminate this risk in its entirety due to the subjective nature of what was studied, which is human behavior regarding a topic. That said, we tried to tailor our methods to better encapsulate all discussion found related to the topic, *minimizing value judgements* regarding expressed opinions in messages. Another point to be considered is that we did not follow strict existing methods for thematic analysis, like the work presented by Hoda et. al [20]. We were inspired by works that employ these techniques to analyze our data in a similar manner, which was simpler in nature, but still yielded the comprehensive thematic structure presented in section 4.3. We present this point to argue that we believe our results are *credible*, as the numbers and themes found showed an overall view of the discussions that is in line with manual experience gathered during the reading of more than 2,700 messages.

2.7 Conclusion

In this paper, we presented the results of a thematic analysis based on discussion threads from the Boost mailing list, which aims to expand our understanding of how developers discuss migration efforts that target upgrading the C++ language standard version. After analyzing 603 messages, we collected evidence that this particular kind of software migration is one of considerable importance to the Boost community of developers. The discussions are definitely present and seem to increase in presence over time.

Throughout these discussions, we could see that C++03 and C++11 specifically had major influence on the discussions between developers at Boost. We saw that the challenges of software migration were avidly discussed, and arguments driving migration often mirrored arguments deterring it. We found competing interests in the form of desires to spearhead C++ development and desires to maintain existing code reliable for long-time users. These discussions were often lengthy and inspired calls for change in the organization for facilitating modernizing proposals.

Future works to expand these understandings regarding software migration could focus on analyzing other developer community groups, in order to find similarities and/or differences in our results compared to those communities. Additionally, further research could investigate the impact of organizational structures on the adoption of new standards, as well as the potential role of automated tools and frameworks in mitigating software migration challenges.

Capítulo 3

Conclusões

Nessa pesquisa foi apresentado um processo de coleta e análise de discussões de desenvolvedores sobre migração de software. Mais de 600 mensagens foram analisadas e classificadas, e uma análise temática foi feita para melhor se apresentar a natureza das discussões.

Os resultados sugerem que o assunto foi amplamente discutido pelos desenvolvedores, gerando discussões muito extensas e muitas vezes desviando de seu objetivo principal. Além disso, foi observado que os padrões de C++ mais pertinentes às discussões desse grupo foram C++03 e C++11, sendo mencionados em mais de 40% de todas as mensagens analisadas. Dos desafios enfrentados, a característica mais presente foi a de conflitos de opinião com relação ao objetivo principal dos desenvolvedores. Isso é, havia desenvolvedores que desejavam poder utilizar versões de C++ mais novas para produzir inovações na linguagem, mas também havia desenvolvedores cujo interesse era de manter código já em uso para prover estabilidade e melhorias para seus usuários de longa data. Esses interesses distintos foram observados como os principais motivadores e dissuasores, respectivamente, com relação à migração para novos padrões de C++.

Trabalhos futuros nessa via de estudo poderiam expandir os achados de algumas formas. Uma delas seria a de reproduzir o processo com foco em outras comunidades ou grupos de desenvolvedores de software, a fim de encontrar paralelos e/ou diferenças entre ambos, e também expandir o conhecimento do impacto da evolução de linguagens em desenvolvedores. Outra maneira seria investigar qual impacto as estruturas hierárquicas organizacionais têm nesses processos de migração de software. Outra, ainda, seria investigar a possibilidade de utilizar ferramentas automatizadas para auxiliar no processo de migração de software.

Referências

- [1] Bragagnolo, Santiago, Nicolas Anquetil, Stéphane Ducasse, Abderrahmane Seriai e Mustapha Derras: *Software Migration: A Theoretical Framework*. Tese de Doutorado, Inria Lille Nord Europe-Laboratoire CRISAL-Université de Lille, 2021. 4, 6
- [2] Brant, John, Don Roberts, Bill Plendl e Jeff Prince: *Extreme maintenance: Transforming delphi into c#*. Em Marinescu, Radu, Michele Lanza e Andrian Marcus (editores): *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*, páginas 1–8. IEEE Computer Society, 2010. <https://doi.org/10.1109/ICSM.2010.5609731>. 4
- [3] Kontogiannis, Kostas, Johannes Martin, Kenny Wong, Richard Gregory, Hausi A. Müller e John Mylopoulos: *Code migration through transformations: an experience report*. Em MacKay, Stephen A. e J. Howard Johnson (editores): *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research, November 30 - December 3, 1998, Toronto, Ontario, Canada*, página 13. IBM, 1998. <https://dl.acm.org/citation.cfm?id=783173>. 4
- [4] Martin, Johannes e Hausi A. Müller: *C to java migration experiences*. Em *6th European Conference on Software Maintenance and Reengineering (CSMR 2002), 11-13 March 2002, Budapest, Hungary, Proceedings*, páginas 143–153. IEEE Computer Society, 2002. <https://doi.org/10.1109/CSMR.2002.995799>. 4
- [5] Moore, Melody M., Spencer Rugaber e Phil Seaver: *Knowledge-based user interface migration*. Em Müller, Hausi A. e Mari Georges (editores): *Proceedings of the International Conference on Software Maintenance, ICSM 1994, Victoria, BC, Canada, September 1994*, páginas 72–79. IEEE Computer Society, 1994. <https://doi.org/10.1109/ICSM.1994.336788>. 4
- [6] Verhaeghe, Benoît, Anne Etien, Nicolas Anquetil, Abderrahmane Seriai, Laurent Deruelle, Stéphane Ducasse e Mustapha Derras: *GUI migration using MDE from GWT to angular 6: An industrial case*. Em Wang, Xinyu, David Lo e Emad Shihab (editores): *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, páginas 579–583. IEEE, 2019. <https://doi.org/10.1109/SANER.2019.8667989>. 4
- [7] Zhou, Hong, Jian Kang, Feng Chen e Hongji Yang: *OPTIMA: an ontology-based platform-specific software migration approach*. Em *Seventh International Conference*

- on *Quality Software (QSIC 2007)*, 11-12 October 2007, Portland, Oregon, USA, páginas 143–152. IEEE Computer Society, 2007. <https://doi.ieeecomputersociety.org/10.1109/QSIC.2007.40>. 4
- [8] Cossette, Bradley e Robert J. Walker: *Seeking the ground truth: a retroactive study on the evolution and migration of software libraries*. Em Tracz, Will, Martin P. Robillard e Tevfik Bultan (editores): *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*, página 55. ACM, 2012. <https://doi.org/10.1145/2393596.2393661>. 4
- [9] Pirkelbauer, Peter, Damian Dechev e Bjarne Stroustrup: *Source code rejuvenation is not refactoring*. Em *SOFSEM 2010: Theory and Practice of Computer Science: 36th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 23-29, 2010. Proceedings 36*, páginas 639–650. Springer, 2010. 4
- [10] Lucas, Walter, Rodrigo Bonifácio, Edna Dias Canedo, Diego Marcilio e Fernanda Lima: *Does the introduction of lambda expressions improve the comprehension of java programs?* Em Carmo Machado, Ivan do, Rodrigo Souza, Rita Suzana Pitangueira Maciel e Cláudio Sant'Anna (editores): *Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, Salvador, Brazil, September 23-27, 2019*, páginas 187–196. ACM, 2019. <https://doi.org/10.1145/3350768.3350791>. 4
- [11] Lucas, Walter, Fausto Carvalho, Rafael Campos Nunes, Rodrigo Bonifácio, João Saraiva e Paola Accioly: *Embracing modern c++ features: An empirical assessment on the kde community*. *Journal of Software: Evolution and Process*, página e2605, 2023. 4, 5, 7, 15
- [12] Overbey, Jeffrey L. e Ralph E. Johnson: *Regrowing a language: refactoring tools allow programming languages to evolve*. Em Arora, Shail e Gary T. Leavens (editores): *Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA*, páginas 493–502. ACM, 2009. <https://doi.org/10.1145/1640089.1640127>. 4
- [13] Nicolini, Thiago, André C. Hora e Eduardo Figueiredo: *On the usage of new javascript features through transpilers: The babel case*. *IEEE Softw.*, 41(1):105–112, 2024. <https://doi.org/10.1109/MS.2023.3243858>. 4
- [14] Khadka, Ravi, Belfrit V Batlajery, Amir M Saeidi, Slinger Jansen e Jurriaan Hage: *How do professionals perceive legacy systems and software modernization?* Em *Proceedings of the 36th International Conference on Software Engineering*, páginas 36–47, 2014. 4
- [15] Swillus, Mark e Andy Zaidman: *Sentiment overflow in the testing stack: Analyzing software testing posts on stack overflow*. *Journal of Systems and Software*, 205:111804, 2023. 5, 8

- [16] Tahir, Amjed, Jens Dietrich, Steve Counsell, Sherlock Licorish e Aiko Yamashita: *A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites*. Information and Software Technology, 125:106333, 2020. 5, 8
- [17] Bennett, Keith H e Václav T Rajlich: *Software maintenance and evolution: a roadmap*. Em *Proceedings of the Conference on the Future of Software Engineering*, páginas 73–87, 2000. 6
- [18] Layzell, Paul J. e Linda A. Macaulay: *An investigation into software maintenance - perception and practices*. J. Softw. Maintenance Res. Pract., 6(3):105–120, 1994. <https://doi.org/10.1002/smr.4360060302>. 6
- [19] Singh, Raghu: *International standard iso/iec 12207 software life cycle processes*. Software Process Improvement and Practice, 2(1):35–50, 1996. 6
- [20] Hoda, Rashina: *Socio-technical grounded theory for software engineering*. IEEE Transactions on Software Engineering, 48(10):3808–3832, 2021. 8, 25
- [21] Khder, Moaiad Ahmad: *Web scraping or web crawling: State of art, techniques, approaches and application*. International Journal of Advances in Soft Computing & Its Applications, 13(3), 2021. 10
- [22] Mammone, Alessia, Marco Turchi e Nello Cristianini: *Support vector machines*. Wiley Interdisciplinary Reviews: Computational Statistics, 1(3):283–289, 2009. 10
- [23] Bafna, Prafulla, Dhanya Pramod e Anagha Vaidya: *Document clustering: Tf-idf approach*. Em *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, páginas 61–66. IEEE, 2016. 11
- [24] Chowdhary, KR1442 e KR Chowdhary: *Natural language processing*. Fundamentals of artificial intelligence, páginas 603–649, 2020. 11
- [25] Qader, Wisam A, Musa M Ameen e Bilal I Ahmed: *An overview of bag of words; importance, implementation, applications, and challenges*. Em *2019 international engineering conference (IEC)*, páginas 200–204. IEEE, 2019. 11