"How do people decide?": A Model for Software Library Selection

Minaoar Hossain Tanzil
minaoar@gmail.com
University of Calgary
Calgary, Alberta, Canada

Gias Uddin
guddin@yorku.ca
York University
Toronto, Ontario, Canada

Ann Barcomb
ann.barcomb@ucalgary.ca
University of Calgary
Calgary, Alberta, Canada

ABSTRACT

Modern-day software development is often facilitated by the reuse of third-party software libraries. Despite the significant effort to understand the factors contributing to library selection, it is relatively unknown how the libraries are selected and what tools are still needed to support the selection process. Using Straussian grounded theory, we conducted and analyzed the interviews of 24 professionals across the world and derived a model of library selection process which is governed by six selection patterns (i.e., rules). The model draws from marketing theory and lays the groundwork for the development of a library selection tool which captures the technical and non-technical aspects developers consider.

KEYWORDS

third-party, software library, adoption process, decision pattern, open-source software, grounded theory, interview study, qualitative research

1 INTRODUCTION

In 2011, Marc Andersson noted that *"Software is eating the world"* [2]. With such growing demands, software companies seek to build and deploy their products quickly and efficiently by utilizing reusable components such as software libraries [29]. The libraries can be open source software (OSS) or proprietary, e.g., Application Programming Interfaces (APIs) from cloud vendors. Software developers often prefer the reuse of a library over the re-implementation of a feature from scratch [36]. According to a 2017 report by the European Commission, the use of OSS saves European companies approximately €456 billion per year through increased productivity, efficiency, and direct cost savings [15]. However, such benefits do not always accrue; Spinellis outlines many potential concerns of relying on an OSS library, among them: the project might not be maintained, the code may not be of high quality, the license may not allow the desired use, and the documentation may be poor [30].

The selection factors of software libraries have been the subject of several studies. In particular, technical criteria related to the selection of OSS tools are studied and cataloged extensively [9, 10, 14, 20, 23, 23–25, 34–39, 45]. The factors are diverse, as the selection process requires balancing a number of considerations, both those inherent to the library itself and those relating to the context in which it will be used. [30, 43]. Factors such as the fit between the functional capability of available software and the acceptance (or rejection) of the technology by peers may also influence the selection decision [12, 13]. We find that the current literature regarding software library selection is limited in describing the selection process. While the literature offers an understanding of various library selection factors, we do not know how such factors are consulted and what rules are followed during the selection of a library. As is often the case in any organization, rules are needed to guide the completion of a task.

In this paper, we address this gap by developing a model of library selection process. We developed the model and the guidelines by conducting semi-structured interviews of developers from 24 large, medium, and small international software companies across the world. We asked the participants to offer us insights about the overall process they follow in their company during the selection of a third-party library. We asked several semi-structured questions that we formulated based on our review of literature and the observations from the interviews themselves. We analyzed the interview responses, which produced a model of API selection process in the companies that we found to be governed by six selection patterns (i.e., rules). The adoption of the rules was influenced by seven barriers. The developers found our model of selection process to be an accurate representations of their library selection practices in their companies.

The contributions of the paper are as follows:

- We developed a comprehensive model of the library selection process which incorporates complex interplay among organizational conditions and barriers that were neglected in previous studies [23, 24, 30, 39] that focused mostly on library selection factors. Our library selection model provides a detailed selection process influenced and obstructed by organizational, technical, and individual conditions, barriers, and decision patterns by drawing on mature models from the marketing domain.
- We provide five actionable recommendations for organizations to improve the library selection process.

2 RELATED WORK

In this section, we cover buyer behavior models in the marketing domain, technology adoption models for businesses, and factors and processes in the evaluation of libraries.

2.1 Buyer Behavior Models

We use buyer behavior as the lens through which we examine software library adoption decisions. In contrast to technology adoption models, consumer and business buyer behavior models drawn from marketing theories provide more in-depth insights into how a consumer or a business organization makes a decision to procure a product. In their seminal textbook, Kotler and Armstrong defined consumer/organization-specific concerns as 'influences' and separated these from product-specific attributes (which we will henceforth refer to as factors, consistent with the software engineering literature) while explaining the influences in buying process [21]. Moreover, the product-specific factors are also a foundational element of the Fishbein Multiattribute Model which calculates the weighted average of all product-specific factors to define which product the consumer will select [16]. Since it was introduced almost 60 years ago, this model has been "extensively used by consumer researchers" [3]. In addition to influential conditions, and important product factors, the buyer behavior models also provided steps of buying decision process and actors involved or influencing the product adoption. Though these models provide a more holistic approach compared to technology adoption models, there is no study of how such models are applicable to technology adoption, specifically, the library adoption process.

2.2 Technology Adoption Models

The process by which a tool comes to be adopted within an OSS community was found to consist of several phases: knowledge, individual persuasion, individual decision, individual implementation, organizational adaptation, and organizational acceptance [22]. In marketing, psychology, and technology literature, a number of technology adoption models have been proposed. Theory of Reasoned Action (TRA) [17] has become the foundational base for investigating personal technology usage [31]. Two notable derivations of TRA model are the Theory of Planned Behavior (TPB) [1] which added perceived behavioral control and the Technology Acceptance Model (TAM) [7, 8]. The TAM model considered perceived usefulness, perceived ease of use, and attitude toward use for technology adoption by individual consumers. While these models explained consumer behavior to accept a technology product, TOE (Technological, Organizational, and Environmental) framework [33] describes organizational technology adoption and has been criticized as too generic [46].

2.3 Selection Factors for Software Libraries

Larios-Vargas et al. conducted a comprehensive study of factors considered for library selection, identifying 26 technical, human, and economic factors through semi-structured interviews, which were subsequently validated through a survey of 116 developers [23]. In contrast to our study, they divided the library selection process just in two stages: up-front selection and prototype stage. Their simplistic selection process does not explain how the internal external conditions change the selection process, information sources or selection factors.

Several research studies focused on specific selection factors such as security [40, 41, 44], documentation [34], and non-functional requirements [25].
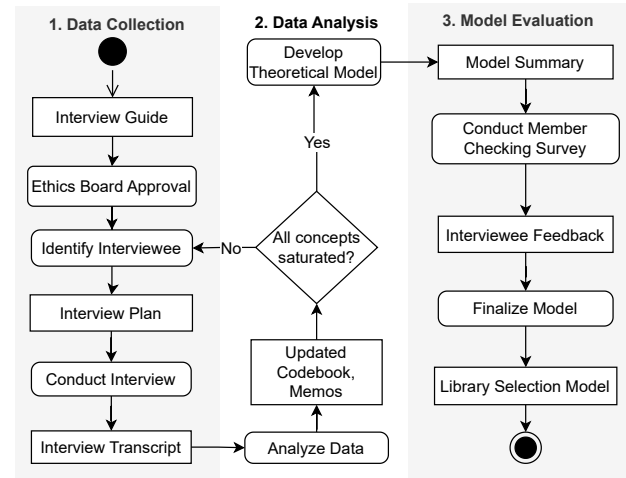

Figure 1: Overall research method for developing theoretical model and conceptual tools. Grounded theory is applied from data collection to the model evaluation. The rounded boxes denote the activity and the rectangles denote the outcome.

Besides the research studies, the software engineering textbooks also outline the advantages and risks of reusing [29] and maintaining dependency on external components such as libraries [42]. However, they do not provide the complete library selection process along with the barriers and the selection patterns modern technology teams are facing.

Comparison process of reusable OSS components (other than libraries) has been proposed based on Qualification and Selection Open Source (QSOS) and Open Business Readiness Rating (Open-BRR) [11, 28, 39]. Studies also identified evaluation factors for OSS applications [24] and proposed information collection and investigation criteria [6]. While these attempts to codify the decision process do address human and technical factors, they are primarily focused on the selection of software applications, not libraries.

3 STUDY METHODOLOGY

Given the absence of a model of library selection process in companies, we adopted Straussian grounded theory [4] to derive such a model. We conducted semi-structured interviews of industry developers to learn about the model because such interviews allow us to elicit unexpected information and to evolve our questions as the study progressed [19, 27]. Figure 1 provides an overview of the overall research method that was applied to the study.

3.1 Data Collection

We conducted 24 interviews between June 2022 and January 2023. Table 1 provides an overview of participants. The average number of years of professional experience of the participants was 14 years. Their roles spread across the spectrum of engineering and leadership roles in nine different countries from five continents. There was also wide variability in technologies used (e.g., C/C++, Java, Python, Android, iOS, .NET, and machine learning). Many of our participants worked in large corporations (e.g., Google, Microsoft,

Table 1: Interview participants by years in industry, role (Arc-Architect, EM-Engineering Manager, Cons-Consultant, SDE-Software Development Engineer, MLE-Machine Learning Engineer, CTO-Chief Technology Officer), primary technology (JV-Java, PY-Python, A/IOS-Android/iOS, PE-Perl, JS-JavaScript, RoR-Ruby on Rails, Any-not limited by technology) geographic location (AS-Asia, AU-Australia, EU-Europe, NA-North America, SA-South America), and tech. people size.

P#	Yrs	Role	Tech	GEO	Industry	Size
P01	12	Arc	JV	EU	Automotive	500
P02	6	SDE	PY	NA	Cloud Service	80,000
P03	12	SDE	A/IOS	NA	Automotive	600
P04	20	CEO	.NET	AS	Broadcast Media	54
P05	16	EM	.NET	AU	Financial	12
P06	17	SDE	PE	EU	Tech	20
P07	9	CTO	JS	NA	Data Analytics	6
P08	9	EM	Any	NA	Cloud Service	10,000
P09	13	Arc	PY	NA	Web	100
P10	15	EM	JS	EU	Energy	300
P11	7	MLE	PY	NA	Data Analytics	30
P12	22	Cons	PE	AS	Tech	1,000
P13	15	Arc	JV	NA	Retail	200,000
P14	6	SDE	A/IOS	AS	Financial	100
P15	22	CTO	.NET	AS	Enterprise	300
P16	9	SDE	JV	AS	Cyber Security	300
P17	15	CTO	RoR	EU	Custom Software	6
P18	27	CEO	C++	NA	Financial	40
P19	15	EM	RoR	NA	Cloud Service	75,000
P20	10	SDE	A/IOS	NA	Food Service	10
P21	13	SDE	RoR	NA	CI/CD	900
P22	30	Arc	JV	NA	Operating Sys.	9,000
P23	7	MLE	PY	SA	Custom Software	750
P24	6	MLE	PY	NA	Medical	80

etc.) and OSS projects. Among all the interviewees, three experts identified themselves as women. Our participants covered 16 application domains including specialized regulated areas as such health, finance, cybersecurity, and broadcast media.

3.1.1 Theoretical Sampling for Recruitment. We started with an architect (participant P1) from our professional network who had twelve years of experience including developing a payment system from scratch and using numerous libraries. Analysis of this interview revealed the need for information regarding licensing. Hence, we selected the next participant (P2) to investigate licensing concerns. We used both convenience sampling and snowballing to recruit interviewees from direct and extended professional networks after validating their relevance through screening calls or emails. The motivation for selecting every participant is provided in the appendix in the replication package [32].

3.1.2 Concept Saturation over the Interviews. An integral part of grounded theory is concept saturation when further data collection and analysis does not provide additional information about a concept. After each interview, we analyzed the transcript and

performed open coding to identify all the concepts discussed in that interview. We also compared the discussed concepts and their dimensions with previous interviews. If we found that no new dimension of a concept was discussed consecutively for few interviews, we considered the concept saturated. By the time we interviewed P24, no new concepts were discussed and all previous concepts were saturated.

3.2 Data Analysis

All interviews were conducted and recorded online using Microsoft Teams for easy transcription. Because of the lack of accuracy of auto-transcription, each interview was manually corrected. We used the qualitative data analysis tool MaxQDA [18] for coding. The coding process consisted of three steps: open coding and memoing, axial coding, and theoretical integration. The steps are described below.

3.2.1 Open Coding and Memoing. During each interview, we continuously took field notes so that we could identify concerning points. After each interview, we made summary memos with the new concepts that emerged, the properties of existing concepts that became saturated, and the questions that arose. After the first couple of interviews, the concepts had been identified and we were able to begin open coding after each interview.

3.2.2 Axial Coding. After coding and analyzing the first four interviews, we started to discover the intersection points of different concepts using axial coding. For example, we identified that some selection factors are related to technical issues such as *ease of use*, *performance*, and *compatibility*, whereas some factors are not dependent on the software of the library itself, such as *active development*, *community support*, *paid customer support*, which mostly relate with central concepts of *support and maintenance*. Similarly, *company culture* and *company technology* had a central theme of *organizational influence*, which differs from developer's *personal background*, *experience level* or *personal motivation* which fall under the *individual influence* category. Using axial coding analysis, we created another layer of categories to group similar factors, conditions, sources, processes, barriers, and selection patterns.

3.2.3 Theoretical Integration. Diagrams and memos helped us conduct theoretical integration to generate the core concept of our research. For example, we started the interviews and analysis to explore how the developers adopt libraries. As the analysis progressed, we attempted to generate a core concept by sorting the memos and drawing interactive diagrams. We realized that unless we deeply understood why developers use libraries, we could not generate a core concept. After we identified developers' motivations and concerns for third-party libraries, the core concept emerged as the decision patterns of software library adoption which in an organization can guide a developer to make a decision by employing the library adoption steps and by considering the factors and conditions that influence the steps. For example, some developers wanted to talk to people when they were under tight deadlines *"the way of choosing libraries was actually talking to peers because we were in a rush to deploy" (P16)*. So we thought $Meet\ deadline$ was a core concept. However, we also found that even when there was no deadline, some developers still reached out to their peers, because they did not want to spend time searching the library: *"They [friends*

or colleagues] already did it, right? They can just tell you do this." *(P11)* They wanted to *make their life easy*. Connecting these two motivations, we came to the conclusion that both of them followed a common decision pattern of *Just Do It*. In similar way, theoretical integration helped us develop the library selection model along with the barriers and the rest of the selection patterns.

4 LIBRARY SELECTION PROCESS

Through our interviews of software developers in companies, we identified a selection process of third-party software libraries that involves five steps, where a suite of factors are considered along the steps during the selection process.

4.1 Steps Followed

The selection of a third-party library consists of five steps: **Search**, **Compare**, **Review**, **Integrate**, and **Maintain**. Each step consists of one or more action items.

The step **Search** entails three actions: *identifying the problem*, *talking to people*, and *performing an online search*. *"The very first step is I'm reaching out to my colleagues. If I cannot find the answer within the team from the colleagues, then I will Google it."* (P11) Next, developers **Compare** available libraries by *comparing alternatives*, *exploring candidates*, and *selecting the outstanding library*. Online information sources like Stack Overflow are consulted extensively for comparison. The third step **Review** involves multiple actions by the developers and the teams: *team discussion*, *design/code review*, *consent process*, *convince developers*, *convince non-developers*, and *stakeholder consultation*. Depending on the company size and practices, developers may request approval from dedicated teams who review the security and license issues of third-party libraries. Once a library is selected, the fourth step is to **Integrate** a library into the current software by developing *proof of concept development*, *integrating* with production system, and by *gradual adoption*. The last step is **Maintain**, where developers may *scale up usage* by using more functionalities of the library, perform *post integration maintenance* by upgrading, and even *migration* to new libraries: *"You can opt-in for new versions or you can ignore the new versions. Usually, major versions might have something which can break your code. But you don't upgrade without any reason."* (P05)

In total, there were 18 actions associated with the steps. Figure 2 summarizes each of steps and the action items associated with it, illustrating the step with a trace from the interviews.

4.2 Information Sources Consulted

Developers extract information about libraries from a total of 15 different *information sources* categorized under five different categories (see Figure 3): *human sources*, *online search and articles*, *question answer sites*, *repositories*, and *organizational sources* internal to the company.

Other than generic *online searching*, the most common online information source is Stack Overflow: *"Stack Overflow is a huge resource for seeing what different people recommend…and also seen a lot on things like Quora and Reddit where you say what's the best library for doing X and people will list out a couple of different*

options there." (P07). The latest novel sources are ChatGPT[1] and CoPilot[2]: *"I'm giving a prompt to ChatGPT or my team is giving a prompt to the CoPilot, not Googling first… that's an interesting way of finding relevant packages or APIs. "* (P16) Finally, based on the *company maturity* and availability of internal *organizational sources*, namely *existing source code*, *internal wikis* developers can turn to such *organizational content*: *"[We have an] internal GitHub. Then there are internal search engines, also there are some question answers like Stack Overflow. I think that is true for all these big corporations."* (P02)

4.3 Selection Factors Considered

We identified 28 library-specific *factors* under four categories: *software factors*, *commercial factors*, *maintenance factors*, and *external factors* which are shown in Figure 4.

In the category of technical *software factors*, we see *compatibility*, *stability*, *flexibility*, *capability*, *security*, *performance*, *ease of use*, *ease of installation*, *size of library*, and *interesting interface* as primary concerns. Several of these considerations are provided by P15: *"We have to understand how much memory it is gonna use. How much time does it take to execute? Or how much CPU is gonna use? Is this library compatible with the development environment? How is the thread safety within the library?"* (P15)

Based on the *investment capacity* or *regulatory requirements*, developers may have to consider *commercial factors* such as *license*, *cost*, *dependency*, *roadmap*, *open source*, *documentation*, and *demo availability*.

For long-term projects, developers consider *Maintenance factors* that include whether the library source code is under *active development*, enjoys *community support*, is *supported by a reputed organization*, has a *large community*, offers *customer support*, and is *supported by own organization*. Finally, *external factors* can come into play. These include *popularity*, *search engine ranking*, *familiarity*, *used by reputed organizations*, and *detailed benchmark*. The following quotation demonstrates familiarity affects selection:

> *"Oftentimes what happens is that the decision or the choice gets influenced by an individual's bias or familiarity or previous experience with one particular product or service"* - P08

5 LIBRARY SELECTION PATTERNS

During our interviews with the software developers, we observed that there are six selection patterns that could guide the adoption of the software library selection process in their companies (see Section 5.1). We found six types of conditions that influence the usage of specific selection factors while adopting a pattern (see Section 5.2) and six barriers that can influence the adoption of a pattern itself and its priority within a company (see Section 5.3).

5.1 The Selection Patterns (SP)

All of the attributes of the library selection process described in Section 4 have complex interactions which we captured in the six selection patterns as shown in Figure 5. All of these patterns
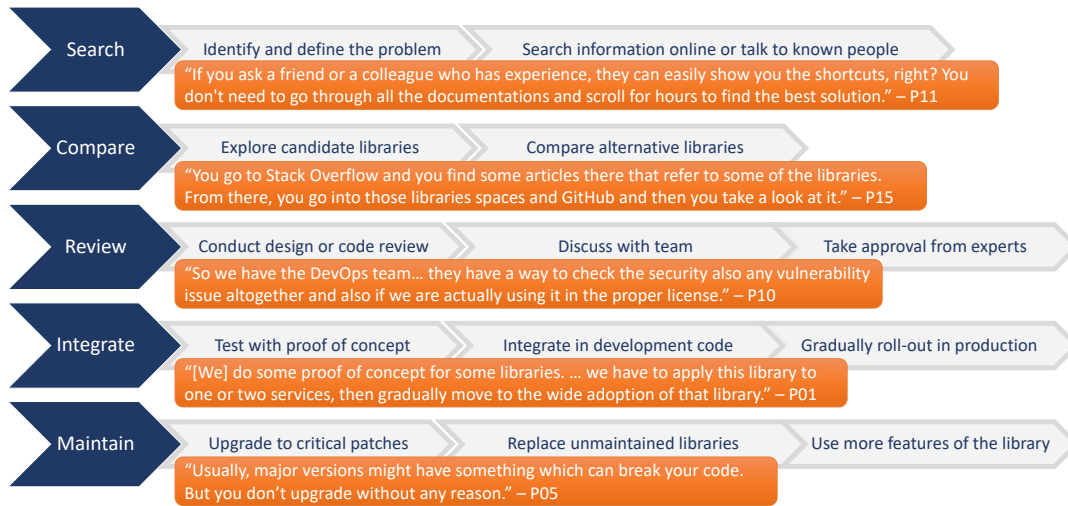
---

[1]https://openai.com/blog/chatgpt
[2]https://github.com/features/copilot

Figure 2: Majors steps taken during the library selection process. In every step, there are multiple actions. Each row denotes a step.
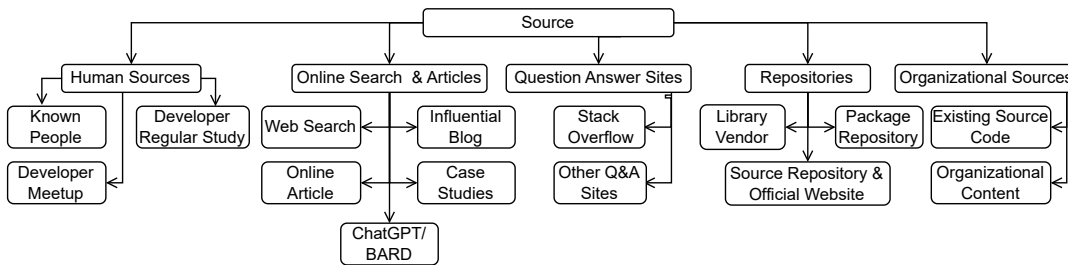
Figure 3: Information sources from where developers collect library-related data, reviews, or opinions.
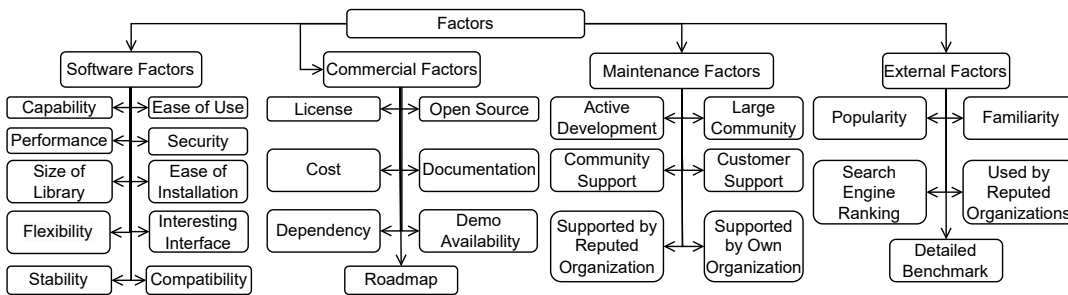
Figure 4: Technical and non-technical aspects of libraries considered by developers during library selection

represent an abstract solution to a recurring problem [26] based on the scenarios in the industry:

> *"It's not that always you have to choose a library or framework which is technically the best. Rather capabilities of the library and the organization, the domain, the people, the timing, everything influences that decision."* - P01

On the basis of internal and external conditions, certain factors of libraries become relevant to developers, certain steps of the selection process become more important to them, and they follow a decision pattern accordingly. For example, inexperienced developers can skip *review* or *maintenance* considerations and highly emphasize *easy to use*, *easy to install*, or *popular* libraries when following $Just\ Do\ It$ pattern. Based on the emphasis on the factors, their source of information also changes. For example, developers using the $Ensure\ Compliance$ pattern will have to consult with internal organizational sources or legal resources if they care about the *license* factor.

	Scenario	Dev Team Response	Consequence
Just Do It (Developer)	Faster delivery. Prefer less effort.	(F) Easy to use/install, familiar library. (S) Ask known people. (P) Skip code review or maintenance plan.	Can lead to performance bottleneck, security or legal issues
	"For startups, a lot of it [priority] is just speed to market and how much resources is gonna eat up using any specific library." - P07		
Reuse Robust Component (Senior Dev)	Matured org, stable code. Dev concerned of quality & maintainability.	(F) Open source, community trusted, stable. (S) Library repository, technical blog. (P) Thorough code review, gradual adoption.	Stable, maintainable code. Over analysis can cause delivery delay.
	"[this corporation] as a whole is actually built on the open source libraries that are suitable for our use cases. If you find a library, use it; only build if you can't find anything." - P13		
Avoid Structural Change (Architect)	Large scale software with custom framework.	(F) Flexible library, small sized. (S) Library repository, internal spec. (P) Thorough design review. Wrap library in API.	Less external dependency. Avoids libraries if possible.
	"Look at the structural aspect of the application and make sure you do want to avoid changing it or maybe it's the time to change it." - P22		
Empower the Team (Team Lead)	Strong learning culture. Tech limitations/strengths of dev team.	(F) Documented, customer support, popular. (S) Collect opinion from online/community. (P) Thorough team discussion.	Compromise solution for team's interest. Motivates team.
	"So, looking at community popularity helps because then it helps to hire people. It helps to retain people. They like to use technologies that are transferable." - P19		
Ensure Compliance (Experts)	Matured org, regulated industry. Presence of security/legal experts.	(F) Compatible License. Secure. (S) Library documentation. (P) Expert consent/approval for library.	Protects from security and legal risk. Approval can be delayed.
	"There was a licensing issue in a library and we had to replace half of all libraries. We are now very concerned about adding any external library, because without proper license, it will be a legal problem." - P09		
Maintain Continuous Stability (DevOps)	Long term application.	(F) Actively maintained. (S) Commit/issue/usage history from repo. (P) Software bill of materials, maintenance plan.	Prevents vulnerability, system breakage. More resource needed.
	"When we integrated the updated version our whole interface broke. We had to change all the interceptors, interfaces, everything. This maintenance is quite hard, a full-time work to always keep updated." - P14		

Figure 5: Six library selection related decision patterns found in the industry. Each selection pattern is presented using the scenario, dev team response, and consequences. The scenario is further explained using the conditions. The dev team response is explained by providing the consideration factors (F), specific sources (S), or special processes (P). The primary actor of each selection pattern is shown under the pattern name in parentheses. Orange boxes show example traces from the interviewees.

5.1.1 Startup Patterns. We found one selection pattern $Just\ Do\ It$ was more prevalent in early-stage companies or products. However, this pattern was also observed in $critical\ production\ scenarios$ to provide hot fixes.

SP1: Just Do It. When developers are concerned primarily with time-to-market or are not that interested in long-term maintenance, they may opt to make use of third-party libraries which are $easy\ to\ use$ and $easy\ to\ install$, as illustrated by the following quotes.

> "It was going to solve a particular promotion or something, and it was going to be retired. So usually the long-term maintainability was not a factor." - P06

5.1.2 Patterns in Mature Companies. A couple of patterns were more common in organizations with experienced developers building long-term products or large-scale complex systems.

SP2: Reuse Robust Component. Developers working on long-term or complex products prefer to choose $stable$ libraries which are $actively\ maintained$ for a long period and supported by a $large\ community$. The selection patterns is $Reuse\ Robust\ Component$. Examples of its use are:

> "I want to use as much as already developed, tested, and robust software in my solution…the main thing is that reusability and

having stability in the application inherently out-of-the-box by using a stable, robust library" - P22

SP3: Avoid Structural Change. When existing large-scale applications are built on a custom in-house framework or technology stack, software designers tend to avoid any structural change unless absolutely necessary to improve the architecture. They also tend to integrate only small and specialized third party libraries which does deteriorate their architecture. P18 worked in a large project that had less than 20 libraries in a two million lines of source code. Even when they $integrate$ libraries, they would wrap it under their own structure:

> "We will create just a thin wrapper, that ends up looking like the rest of our platform. The simple source file [wrapper API] is protecting the two million lines of code from the idiosyncrasies of this one particular library." - P18

5.1.3 Common Patterns. Other three selection patterns were commonly found in either early-stage or matured organizations depending on other internal-external conditions.

SP4: Empower the Team. Adopting third-party libraries also empowers the development teams. Open-source libraries provide developers with working experience with popular development
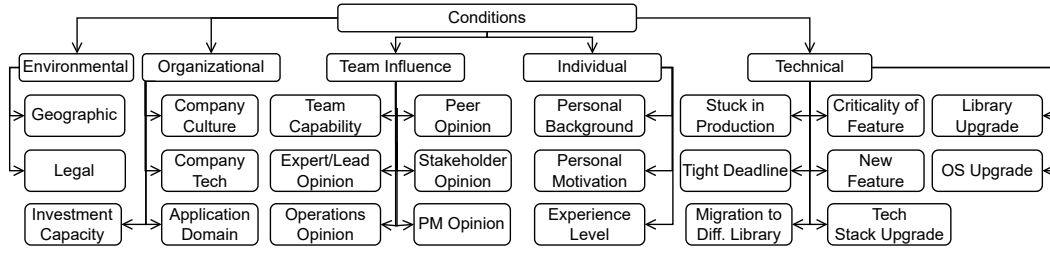
Figure 6: Internal and external Conditions influencing library selection process

components to acquire transferable knowledge whereas proprietary solutions developed internally can create bottlenecks.

SP5: Ensure Compliance. Third-party libraries can come with licensing issues, privacy concerns, or security vulnerabilities. However, not all development teams equally consider the impact of compliance issues. *Legal environment* of the organization or their customers are the primary driver for compliance principle.

SP6: Maintain Continuous Stability. Aside from compliance issues, the biggest risk of using a third-party library is that the library won't be maintained.

To protect themselves from abandonment and stability risks, developers look for libraries where a *large community* and contributors are involved. To *search information* about *active development*, they check *source repositories*:

> *"We have to check whether this library maintenance is active or not. We can determine that by checking their GitHub repository when the last push was given?"* - P04

5.2 The Conditions Affecting the Choice of Factors in Patterns

The library selection process can be influenced by 23 *environmental*, *organizational*, *team influence*, *individual*, and *technical* conditions as shown in Figure 6. These conditions affect the specific selection steps and the library aspects.

Environmental conditions include the *geographic* and *legal* landscapes. For example, regulatory requirements enforce a thorough review process before the integration of libraries: *"In Germany you have to report a security breach in your company... you have to pay two percent of the revenue if a security breach happens and your data gets leaked."* (P17)

Geography can promote the usage of certain types of libraries: *"... everywhere it's not still functional programming, it's not widely adopted. I recently migrated from Asia to Europe and I never saw this trend widely adopted in our previous companies. But here [in Europe], I've seen a lot of people are very interested in the functional programming paradigms. So, to choose the libraries, ... all of these things can have some impact."* (P01)

In some organizations, *team capability* can influence library selection: *"I went for Vue because most of the developers in my company were mostly back-end developers and I found Vue is very back-end developer friendly."* (P04)

Individual developers' *personal background*, *motivation*, and *experience level* can have an impact: *"The excitement of trying some*

new library was also fairly motivating to keep my skills up to date..." (P06)

Finally, there can be *technical* conditions, such as a severe issue in production. *"If the feature is too business critical, then it goes through an even more rigorous decision-making process than the other where, for example, you are just trying to choose a library to show an image and crop it. So that is not so business critical."* (P15)

How do the conditions impact the decision patterns and what library factors developers consider for each decision pattern is shown in Figure 7. Overarching conditions such as *company culture* can have diverse impact of library selection patterns. For example, if the culture promotes faster delivery over long-term maintenance, developers choose the *Just Do It* pattern. On the other hand, if the culture of development team emphasizes on source code quality, they will choose *Reuse Robust Component* pattern. Different conditions again can converge to same patterns. For example, *Reuse Robust Component* pattern can also be followed based on *company technological* conditions such as long-term and large scale products. This pattern can also be practiced based on senior *developer experience*.

5.3 The Barriers Affecting the Choice of the Selection Patterns (B)

In addition to the presence of various library selection patterns, there are certain entry barriers that an organization or a developer may face while integrating a third-party library. Such obstacles can be caused by organizational policy, or by technology infrastructure, or even by developers' mindset, and experience. We identified a total of eight challenges that originate from three major sources of the organizational culture, individual traits, and technical limitations as shown in Table 2. Example quotations illustrating the barriers are also shown in the table.

B1. Lack of Supporting Process. To manage third-party open source libraries, many companies *lack supportive process* such as an OSS program office, legal or security teams. In some cases, organizations may be unclear about what their developers can use and cannot use from third parties, and instead of embracing external libraries, they can be rather rigid.

B2. Inclusivity Barrier. Though library selection is a team decision, sometimes *lack of inclusivity* hinders the consideration of all opinions. Developers seek information from a variety of sources. If the organization does not have a very welcoming, inclusive culture, critical analysis of libraries can be ultimately influenced and driven by outspoken people.
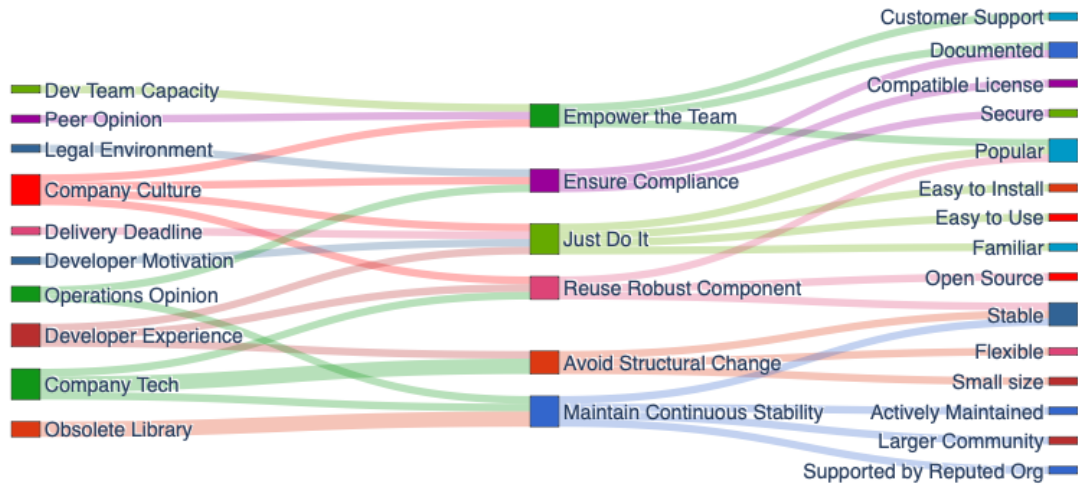
Figure 7: Impact of internal and external conditions (left side) on the choice of selection patterns (in the middle). Selected decision patterns also govern the library-related factors (on the right) that developers consider.

Table 2: Barriers faced in library adoption

Category	Barrier	Exemplary Quote
Organizational	Lack of Supporting Process	*"You usually you would spend 100 hours developing something instead of one hour trying to fill out some legal form"* - P12
	Lack of Inclusivity	*"Often times what happens is that the decision or the choice gets influenced by developers' bias who are more vocal."* - P08
	Lack of a Learning Culture	*"…encourage them to get training materials, and read books so that they can also contribute to team discussions."* - P05
Individual	Lack of Experience	*"I can write just some code doing this, and investing the time to learn the third party library seems too much investment"* - P12
	Change-averse Mindset	*"Some developers are like regular. They're not too much serious, they're moderate, but they don't go the extra mile."* - P05
Technical	Infra/Tech Limitation	*"C++ libraries are notorious for the amount of work to integrate them, to get them to compile on"* - P18
	Reliability of Sources	*"Maybe this is a freelancer who has just put up their blog post so we cannot really evaluate these persons' experience."* - P08
	Lack of Tools	*"Then there is analysis paralysis when you have an abundance of choices and then you can't make a choice... I don't see any tools really for that."* - P12

B3. Lack of a Learning Culture across the development team also creates an unbalanced team discussion in situations like library selection. Even when the culture promotes openness, development teams often have few enthusiastic developers who love to explore and whose opinions might have a disproportionate weight in discussions.

B4. Lack of Experience. *Developers' lack of experience* in software development can motivate them to solve some problems by themselves for which there are already robust libraries. They cannot estimate how much work would be needed to implement by themselves:

> *"Then there is the 'not invented here' syndrome. So many people think that they can do it better."* - P12

B5. Change-Averse Mindset. The analysis also reveals that there can be a *mindset of developers* that can hinder them from learning new technologies or libraries and stick to their own development.

B6. Infrastructural or Technological Limitations. Some organizations *"try to limit the access to the Internet."* (P12) which makes it difficult to install third-party libraries from repositories such as NPM used in Node JS. Also, in some cases, integrating a library to a different environment can be complicated. A participant who developed a C++ application in the Windows environment for over 27 years shared their experience:

> *"If the library was designed for Linux, it's really hard to get them actually to compile with Visual Studio, to get your compile flags all right, to get all your dependencies right."* - P18

B7. Lack of Reliable Source is often a problem for developers when they try collect information or opinions regarding third-party libraries. There can be many online articles and blogs written by some developers that analyze a certain library or tech stack. Without knowing the credibility of those bloggers, developers have

to concentrate on the content and methodology of the opinionated analysis. Finding quality articles can be difficult in the real world.

B8. Lack of Tools. *Lack of availability of comparison tools* for library analysis or other supporting tools can make developers confused about choosing a library after searching online. With the organization's support, policy, and team's willingness to collaborate and explore libraries, there are always challenges of finding out the appropriate library by going through numerous articles, documents, and reviews online. There is still a lack of guiding tools that can support library selection (or in general, technology selection) by summarizing a large amount of data according to the team's priorities. Developers cannot rely on the existing summarization research outcomes since the detailed reference and quality assurance of those tools still are not considered worthy of industrial usage, as illustrated by this quotation which influenced our title:

> *"**How do people decide?** There are like 50 things implementing the same thing. Which one should I use? Then there is analysis paralysis when you have an abundance of choices and then you can't make a choice... I don't see any tools really for that. Maybe there are. I guess there should be or I don't know."* - P12

In practical scenarios, developers are always facing multiple, even conflicting situations, and are balancing between selection patterns to make their final decisions. Moreover, the systematic barriers present in the organizations, or inside the developers can also hinder adoption of certain selection patterns or can motivate towards adopting some less-optimal patterns. For example, a lack of library adoption policy (B1) will motivate less-experienced developers (B5) to adopt the *Just Do It* selection pattern. Conversely, lack of learning culture (B4) will block less-experienced developers (B5) from adopting the *Reuse Robust Component* selection pattern.

6 DISCUSSION

Putting the components of our model, shown in figures 2, 3, 4, 5, 6 together, we arrive at the complete model, shown in figure 8. Comparing the model with the classical Business Buyer Behavior Model describing buyer behavior [21], we observe multiple similarities. In fact, the major building blocks of the models are identical, suggesting that this approach can allow us to draw from mature marketing research in developing a library selection tool.

Based on this model and our interviews, we further propose five recommendations (denoted as R#) for companies trying to formalize the library selection process.

R1: Setting Up Open Source Program Office. We have observed that lack of organizational policy or expertise can hinder adopting libraries. Industry experts recommended setting up open source program office (OSPO) or at least a library adoption policy that developers can easy follow:

> *"Larger companies now are going towards an open source program office where they basically checking the licenses, seeing whether the software can be used within the context of the product, potentially even doing a code or security review."* - P06

Another industry expert (P21) shared a recommended library integration policy from a renowned organization that software companies can follow for setting up their policies[3].

---

[3]https://docs.gitlab.com/ee/development/gemfile.html

> **R1.** Organizations should formalize third-party library policy, and create streamlined processes to support it.

R2: Ownership of Upgrading Libraries. Participants noted that developers should not only focus on solving imminent problems but also plan for future maintenance of a library.

> *"There is aversion to [library] upgrades because of how risky it is. But again, like my experience at [large corporation] has been the best so far where they have all the systems in place to continuously keep libraries up to date... In fact, at [large corporation], every directory has a designated set of owners. So it's the owner's job to make sure that they comply with the update requirements. "* - P19

> **R2.** Organizations should have a library maintenance strategy along with dedicated maintainer for upgrading libraries.

R3: Not Compromising with License and Security. Marketing theories consider cut-off factors as those factors of the product whose absence will bar the consumer from buying it [3]. During our interviews, participants shared that all developers consider *capability* and *compatibility* of libraries as such cut-off factors. They also noted that developers in *small* or *early stage* organizations, or *early career* developers may be unaware of the severity of security and license issues. Industry experts strongly recommended considering such compliance issues as cut-off factors.

> *"I think it's [licensing] even important for midsize companies as well. Even a startup can become famous within a year. And then if this kind of legal problems happen and they get sued. So, I would say, if the company is really small, the tech lead should take the responsibility and be careful in choosing those libraries. Even if the tools are not present there, they should be careful and read the licensing documentations. "* - P03

> **R3.** Organizations should consider *security* and *license* issues, irrespective of *organizational* or *environmental* conditions.

R4: Building an Inclusive Team Culture. We found from the interviews that software library selection is a team decision in most cases. If the organization does not have a very welcoming, inclusive culture, critical analysis of libraries can be ultimately influenced and driven by outspoken people. It would be the responsibility of the leaders to let normally silent members communicate their ideas in whatever preferred (verbal, written) way possible.

> **R4.** Organizations should encourage openness and inclusivity to support developers' diverse communication styles.

R5: Encouraging Developers for Learning New Technologies. Teams can make better decisions when all members develop a habit of regular studies and contribute to the team discussions equally. Providing sanctioned, scheduled opportunities for learning allow experimentation with new technologies and libraries without jeopardizing the production system. Being an engineering manager, P05 emphasized on developing this learning culture:

> *"You can encourage the team so that they can spend some time on research and learning. And also you establish the culture*
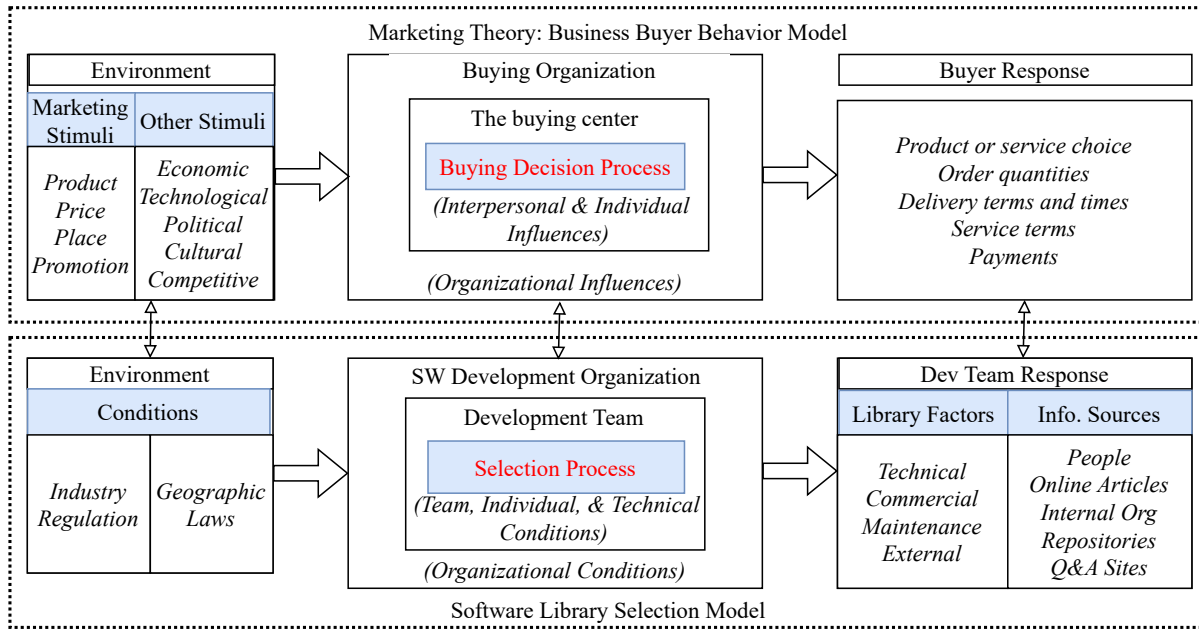
Figure 8: Similarity between library selection model and business buyer behavior model redrawn verbatim from marketing theories [21]. The similar blocks between the two models are mapped using the open point double-sided arrow. The major building blocks of the two models are found identical.

of engineering meeting every week or so, so that people can talk about any technology, not necessarily it is related to your company or a future tech. " - P05

> **R5.** Organizations should promote a culture of technical exploration and discussion through study circles or hackathons.

7 LIMITATIONS

In this section, we evaluate the quality and applicability of our grounded theory research for the development of the library selection model. Corbin and Strauss did not recommend using the terms 'validity' and 'reliability' when discussing qualitative study [4], because qualitative methodologies cannot be assessed using quantitative criteria. They defined 17 measures researchers can use to evaluate the quality and applicability of their grounded theory research. We present the complete evaluation as part of our replication package [32], here we present five major evaluation criteria.

7.1 Industry Fitness

This criterion concerns industry credibility. We conducted member checking [5] by presenting a ten-page summary of our findings, which was sent to 18 of our interviewees who agreed to further contact. This communication included a link to a survey that asked their opinion as to whether the summary reflected their experience and was useful to them. Thirteen participants completed the survey, all of whom opined that the summary was accurate. Five provided detailed feedback on the findings: *"I think the summary captures different aspects of the library selection process very well. It outlines a*

generic, industry-wide pattern with enough details, and also provides exceptional factors that impact that pattern."*

7.2 Industrial Application

Industrial application answers the question of whether the findings provide insight into situations and provide knowledge that can be applied to develop policy, change practice, and add to the knowledge base of a profession [4]. In member checking, one participant shared their feedback, demonstrating that they found our work applicable: *"One interesting thing that I learned from your research is, different developers have different processes and priorities for picking a library, and not everybody is considering all the steps that need to be taken, so I would recommend your paper to all developers to just widen their horizons."*

7.3 Usefulness

To address this measure, we looked at if there are suggestions for practice, policy, teaching, and application [4]. The presentation of guiding principles in the form of patterns, which are widely used in industry, makes the results more accessible to practitioners. Participants also appreciated the suggestions: *"My favorite part of the summary is the takeaway action items that I believe would be useful in building a better culture for adopting the right tools and technologies."*

7.4 Explainability of Theory

The way to assess this measure is to determine if variation is built into the theory [4]. Our conceptual framework is based on six selection patterns which depend on different contexts and lead to

different decisions for the developers to choose the appropriate factors or selection steps. This allows the theory to support a wide variety of circumstances.

7.5 Saturation of Categories

"How is saturation explained, and when and how was it determined that categories were saturated?" is the criterion for evaluating this measure [4]. We provided detailed information about how we performed theoretical sampling to achieve saturation and described how the major categories achieved saturation throughout the progression of the study. Additional figures in the replication package [32] contain a heatmap of concept saturation.

8 CONCLUSION

In this paper, we conducted 24 semi-structured interviews to explore the major steps developers follow in the adoption of a software library. We present a novel library selection model that consists of the process that developers follow to adopt a library, and a set of conditions, information sources, factors, selection patterns that influence the process and barriers that developers face. We proposed five recommendations derived from the concerns that developers identified in interviews. Our study provides researchers with the opportunity to investigate specific adoption steps in more detail. The factors can be used to develop comparative analysis tools. Additionally, the industry can make use of the existing patterns and recommendations to guide decisions about third-party library selection. Our future work focuses on the development of a toolkit to support the automatic comparison of software libraries based on our derived library adoption model.

DATA AVAILABILITY

Codebook, study evaluation, and interview script (no interview transcription is available based on the approved research ethics application) are available at [32].

REFERENCES

[1] Icek Ajzen. 1991. The theory of planned behavior. *Organizational Behavior and Human Decision Processes* 50, 2 (1991), 179–211. https://doi.org/10.1016/0749-5978(91)90020-T Theories of Cognitive Self-Regulation.
[2] Marc Andreessen. 2011. *Why Software Is Eating The World*. The Wall Street Journal, https://www.wsj.com/articles/SB10001424053111903480904576512250915629460. Last accessed on 02 February 2023.
[3] R.D. Blackwell, P.W. Miniard, and J.F. Engel. 2001. *Consumer Behavior*. Harcourt College Publishers, California, USA. https://books.google.ca/books?id=eocrAQAAMAAJ
[4] Juliet Corbin and Anselm Strauss. 2014. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications, CA, USA.
[5] John W Creswell and Cheryl N Poth. 2016. *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, CA, USA.
[6] David Cruz, Thomas Wieland, and Alexander Ziegler. 2006. Evaluation criteria for free/open source software products based on project analysis. *Software Process: Improvement and Practice* 11, 2 (2006), 107–122.
[7] F.D. Davis. 1985. *A Technology Acceptance Model for Empirically Testing New End-user Information Systems: Theory and Results*. Massachusetts Institute of Technology, Sloan School of Management. https://books.google.ca/books?id=hbx8NwAACAAJ
[8] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. http://www.jstor.org/stable/249008
[9] Fernando López de la Mora and Sarah Nadi. 2018. An empirical study of metric-based comparisons of software libraries. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. Association for Computing Machinery, New York, NY, USA, 22–31.
[10] Fernando López De La Mora and Sarah Nadi. 2018. Which library should i use?: A metric-based comparison of software libraries. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. IEEE, Association for Computing Machinery, Gothenburg, Sweden, 37–40.
[11] Jean-Christophe Deprez and Simon Alexandre. 2008. Comparing assessment methodologies for free/open source software: OpenBRR and QSOS. In *Product-Focused Software Process Improvement: 9th International Conference, PROFES 2008 Monte Porzio Catone, Italy, June 23-25, 2008 Proceedings 9*. Springer, Springer, Berlin, Germany, 189–203.
[12] Mark T Dishaw and Diane M Strong. 1998. Supporting software maintenance with software engineering tools: A computed task–technology fit analysis. *Journal of Systems and Software* 44, 2 (1998), 107–120.
[13] Andreas Eckhardt, Sven Laumer, and Tim Weitzel. 2009. Who influences whom? Analyzing workplace referents' social influence on IT adoption and non-adoption. *Journal of information technology* 24, 1 (2009), 11–24.
[14] Rehab El-Hajj and Sarah Nadi. 2020. LibComp: An IntelliJ plugin for comparing java libraries. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, New York, NY, USA, 1591–1595.
[15] European-Commission. 2015. The economic and social impact of software & services on competitiveness and innovation (SMART 2015/0015). https://op.europa.eu/en/publication-detail/-/publication/480eff53-0495-11e7-8a35-01aa75ed71a11. Accessed: 2021-04-22.
[16] Martin Fishbein. 1967. *Attitude and the prediction of behavior*. Wiley.
[17] M. Fishbein and I. Ajzen. 1975. *Belief, Attitude, Intention, and Behavior: An Introduction to Theory and Research*. Addison-Wesley Publishing Company. https://people.umass.edu/aizen/f&a1975.html
[18] VERBI GmbH. 2023. *The #1 software for qualitative and mixed methods data analysis*. VERBI GmbH, https://www.maxqda.com. Last accessed on 03 January 2023.
[19] Siw Elisabeth Hove and Bente Anda. 2005. Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. In *11th IEEE International Symposium on Software Metrics*. IEEE Computer Society, Washington, DC, United States, 23.
[20] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell them apart: distilling technology differences from crowd-scale comparison discussions. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Association for Computing Machinery, New York, NY, USA, 214–224.
[21] P. Kotler and G. Armstrong. 2014. *Principles of Marketing*. Pearson, London, UK. https://books.google.com/books?id=TA3OoQEACAAJ
[22] Martin F Krafft, Klaas-Jan Stol, and Brian Fitzgerald. 2016. How do free/open source developers pick their tools?: A Delphi study of the Debian project. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, New York, NY, USA, 232–241.
[23] Enrique Larios Vargas, Maurício Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. 2020. Selecting third-party libraries: The practitioners' perspective. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. Association for Computing Machinery, New York, NY, USA, 245–256.
[24] Xiaozhou Li, Sergio Moreschini, Zheying Zhang, and Davide Taibi. 2022. Exploring factors and metrics to select open source software components for integration: An empirical study. *Journal of Systems and Software* 188 (2022), 111255.
[25] Mingwei Liu, Xin Peng, Andrian Marcus, Shuangshuang Xing, Christoph Treude, and Chengyuan Zhao. 2021. API-Related Developer Information Needs in Stack Overflow. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4485–4500.
[26] Dirk Riehle, Nikolay Harutyunyan, and Ann Barcomb. 2021. Pattern Discovery and Validation Using Scientific Research Methods. *Transactions on Pattern Languages of Programming* (2021).
[27] Carolyn B. Seaman. 1999. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. Software Eng.* 25, 4 (1999), 557–572.
[28] Raphaël Semeteys. 2008. Method for qualification and selection of open source software. *Open Source Business Resource* May 2008 (2008).
[29] I. Sommerville. 2016. *Software Engineering, Global Edition*. Pearson Education. https://books.google.ca/books?id=W_LjCwAAQBAJ
[30] Diomidis Spinellis. 2019. How to select open source components. *Computer* 52, 12 (2019), 103–106.
[31] Hamed Taherdoost. 2018. A review of technology acceptance and adoption models and theories. *Procedia Manufacturing* 22 (2018), 960–967. https://doi.org/10.1016/j.promfg.2018.03.137 11th International Conference Interdisciplinarity in Engineering, INTER-ENG 2017, 5-6 October 2017, Tirgu Mures, Romania.
[32] Minaoar Tanzil. 2023. *"How do people decide?": A Model for Software Library Selection*. University of Calgary, https://github.com/minaoar/How-do-people-decide-A-Model-for-Software-Library-Selection. Last updated on 14 September 2023.

[33] L.G. Tornatzky, M. Fleischer, and A.K. Chakrabarti. 1990. *The Processes of Technological Innovation.* Lexington Books. https://books.google.ca/books?id=EotRAAAAMAAJ

[34] Gias Uddin, Olga Baysal, Latifa Guerrouj, and Foutse Khomh. 2019. Understanding how and why developers seek and analyze API-related opinions. *IEEE Transactions on Software Engineering* 47, 4 (2019), 694–735.

[35] Gias Uddin and Foutse Khomh. 2017. Automatic summarization of API reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, IEEE Press, Urbana-Champaign, IL, USA, 159–170.

[36] Gias Uddin and Foutse Khomh. 2017. Opiner: an opinion search and summarization engine for APIs. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, IEEE Press, Urbana-Champaign, IL, USA, 978–983.

[37] Han Wang, Chunyang Chen, Zhenchang Xing, and John Grundy. 2020. DiffTech: A tool for differencing similar technologies from question-and-answer discussions. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* Association for Computing Machinery, New York, NY, USA, 1576–1580.

[38] Han Wang, Chunyang Chen, Zhenchang Xing, and John Grundy. 2021. DiffTech: Differencing Similar Technologies from Crowd-Scale Comparison Discussions. *IEEE Transactions on Software Engineering* 48, 7 (2021), 2399–2416.

[39] Anthony I Wasserman, Xianzheng Guo, Blake McMillian, Kai Qian, Ming-Yu Wei, and Qian Xu. 2017. OSSpal: finding and evaluating open source software. In *IFIP International Conference on Open Source Systems.* Springer, Cham, Springer International Publishing, Cham, 193–203.

[40] Dominik Wermke, Jan H. Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. 2023. "Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain. In *In Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P '23).* IEEE Computer Society. https://www.ieee-security.org/TC/SP2023/program-papers.html

[41] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. 2022. Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects. In *43rd IEEE Symposium on Security and Privacy.* https://www.ieee-security.org/TC/SP2022/index.html

[42] Titus Winters, Tom Manshreck, and Hyrum Wright. 2020. *Software engineering at google: Lessons learned from programming over time.* O'Reilly Media.

[43] Thomas Wolter, Ann Barcomb, Dirk Riehle, and Nikolay Harutyunyan. 2022. Open Source License Inconsistencies on GitHub. *Transactions on Software Engineering and Methodology* (2022). https://oss.cs.fau.de/wp-content/uploads/2022/10/wolter-2022-open.pdf

[44] Elizabeth Wyss, Alexander Wittman, Drew Davidson, and Lorenzo De Carli. 2022. Wolf at the Door: Preventing Install-Time Attacks in npm with Latch. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security.* Association for Computing Machinery, New York, NY, USA, 1139–1153.

[45] Litao Yan, Miryung Kim, Bjoern Hartmann, Tianyi Zhang, and Elena L Glassman. 2022. Concept-Annotated Examples for Library Comparison. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology.* Association for Computing Machinery, Bend, OR, USA, 1–16.

[46] Kevin Zhu and Kenneth L Kraemer. 2005. Post-adoption variations in usage and value of e-business by organizations: cross-country evidence from the retail industry. *Information systems research* 16, 1 (2005), 61–84.