The Invisible Adoption Tax: How Non-Technical Frictions Shape Developer Tool Adoption

Ujunwa Njoku

2026

Abstract

Developer tool adoption is often explained by technical qualities such as performance or features. However, many tools that work well technically still fail to gain or retain users. This paper studies how non-technical factors shape developer adoption decisions. We conducted an online survey of 230 software developers who had tried at least one new developer tool in the past 12 months. The survey measured four types of non-technical friction: cognitive friction (learning effort), social friction (community and maintainer interactions), institutional friction (perceived stability and governance) and narrative friction (clarity of purpose and use cases). Using logistic regression while controlling for technical adequacy, we find that these non-technical frictions are strongly associated with tool abandonment and together explain a substantial share of adoption outcomes ($R^2 = 0.42$, $p < 0.001$). Institutional friction shows the strongest association with abandonment, followed by cognitive friction. We introduce the Invisible Adoption Tax (IAT) Framework to describe how these non-technical costs accumulate during tool evaluation and onboarding. Our findings suggest that successful adoption depends not only on technical quality, but also on how tools are presented, supported, and governed.

1 Introduction

Developer tools such as programming languages, testing frameworks, build systems, and IDEs are essential to software development. But many tools fail to gain users even when they work well and offer clear technical improvements over existing options [1, 2]. This suggests that technical quality alone does not explain why developers choose to adopt or abandon tools.

Research on technology adoption shows that decisions are influenced by more than technical features, including social and organizational factors [3, 4]. In software engineering research, some studies have examined parts of this problem, such as documentation quality [5], developer communities [6], and learning effort [7]. However, these factors are often studied separately and adoption is still commonly explained mainly in terms of technical performance.

This paper focuses on how non-technical difficulties affect developer tool adoption. We introduce the Invisible Adoption Tax (IAT) Framework, which describes adoption as being shaped by several kinds of non-technical effort that developers must deal with when trying a new tool. The framework includes four types of friction: cognitive friction (how hard the tool is to learn), social friction (how helpful the community and maintainers are), institutional friction (how stable and trustworthy the tool appears), and narrative friction (how clearly the tool explains what it is for and when to use it). We use the term "tax" to highlight that these efforts add up and create a real cost for developers, even when a tool is technically sound.

To study these frictions, we conducted an online survey of 230 software developers who had tried at least one new developer tool in the past 12 months. Participants reported their experiences

with each type of friction, rated the technical quality of the tools they tried, and stated whether they continued to use the tool or stopped using it. We used logistic regression to examine which factors were linked to tool abandonment while accounting for technical quality.

Our results show that non-technical frictions together are strongly linked to whether a tool is abandoned ($R^2 = 0.42$, $p < 0.001$). Among the four types, institutional friction such as concerns about stability and long-term support has the strongest relationship with abandonment (OR = 2.87). Technical quality remains important, but it does not fully explain the adoption outcomes on its own.

This work makes three contributions. First, it provides evidence that non-technical difficulties play a major role in developer tool adoption. Second, it introduces a simple framework for understanding how these difficulties add up during tool evaluation and onboarding. Third, it offers guidance for tool builders by showing that improving adoption requires more than just technical improvements.

2 Background and Related Work

2.1 Technology Adoption Frameworks

Research on technology adoption comes from several fields. Well-known models such as the Technology Acceptance Model (TAM) [9] and the Unified Theory of Acceptance and Use of Technology (UTAUT) [3] explain adoption mainly through two ideas: how useful a system seems and how easy it is to use. Rogers' Diffusion of Innovation theory [4] adds that adoption is influenced by factors such as perceived advantage over existing options, compatibility with current practices, complexity, and the ability to try and observe a technology before fully committing to it.

These models provide helpful starting points, but they were largely developed to explain the adoption of enterprise systems and consumer technologies. Developer tools differ in important ways. Developers are not only end users, but also technical evaluators, often making judgments about design, architecture, and long-term maintainability. Adoption decisions may be made by individuals, teams, or organizations, and tools often require significant initial effort to learn, integrate, and maintain [8]. These characteristics suggest that adoption models developed for other types of technology may not fully capture how developers evaluate and adopt tools.

2.2 Developer Tool Adoption

Previous studies on developer tool adoption have examined individual factors that influence whether tools are taken up. Meyerovich and Rabkin [1], for example, surveyed developers about programming language adoption and found that social factors such as existing code bases, available libraries, and colleague preferences, strongly shape language choices. Rabkin and Katz [2] studied the adoption of static analysis tools and showed that organizational culture and the way tools fit existing workflows can limit adoption.

Other research has focused on the adoption of APIs and libraries. These studies highlight the importance of clear documentation [5, 10], accessible learning resources [7], and active community support [11]. Work on package ecosystems further shows that factors such as ecosystem maturity, network effects, and maintainer reputation influence which libraries developers choose to use [12].

While this body of work provides valuable insights, most studies examine these factors in isolation rather than considering how they combine during real adoption decisions. In addition, many studies focus only on tools that have already succeeded, which can overlook the experiences of developers who try and abandon tools. Our work builds on prior research by examining multiple

types of friction at the same time and by including both adopted and abandoned tools in the analysis.

2.3 Friction in Technology Use

The idea of "friction" in technology use comes from early research on usability. For example, Nielsen's usability heuristics highlight the importance of efficiency and ease of learning in system design [13]. Similarly, Norman's design principles emphasize reducing mental effort and making systems easy to understand through clear design cues [14]. This work shows that when systems are hard to learn or use, people are less likely to adopt them.

More recent research has applied these ideas to developer tools. Scaffidi et al. [15] examined how much time developers need to learn different programming approaches. Murphy et al. [8] studied how developers discover and learn new APIs. Lerner and Tirosh [16] explored the strategies developers use when picking up new tools. Together, these studies show that learning effort and discovery play an important role in how developers engage with tools.

Our work builds on this research by looking beyond individual usability issues. Instead of focusing on a single source of difficulty, we describe several types of non-technical friction and study how they work together to influence adoption outcomes. We treat friction not just as a usability problem, but as a set of costs that add up over time and affect whether developers continue using a tool.

3 The Invisible Adoption Tax Framework

We propose the Invisible Adoption Tax (IAT) Framework as a way to understand why developers adopt or abandon tools. The framework suggests that adoption outcomes depend not just on technical quality but also on the cumulative effect of non-technical frictions. These frictions act like a "tax" that developers pay beyond evaluating the tool's technical capabilities.

The framework has four main types of friction:

3.1 Cognitive Friction

Cognitive friction is the mental effort required to learn, understand, and use a developer tool. This includes the initial learning curve, the effort to stay proficient, and the ongoing mental load while using the tool. Examples include complicated setup, confusing interfaces, or concepts that clash with what developers already know [17].

Cognitive friction is different from technical problems. A tool can be technically excellent but still hard to use if it demands learning new paradigms or remembering many commands [8].

3.2 Social Friction

Social friction comes from the community and social aspects of using a tool. This includes how responsive the community is, how maintainers communicate, whether online spaces are welcoming or toxic, and how easy it is to get help from peers [6].

Developers rely on communities for learning, troubleshooting, and staying up to date [18]. Negative experiences in these spaces can discourage adoption even if the tool itself works well.

3.3 Institutional Friction

Institutional friction covers trust in the organization or team behind the tool. This includes clarity about governance and decision-making, long-term stability, maintenance commitment, and transparency about funding [19].

Developers face risk if a tool might be abandoned, change in breaking ways, or stop working with other tools [20]. This friction is especially important for tools that take a lot of effort to integrate or become core to a workflow.

3.4 Narrative Friction

Narrative friction relates to how clearly a tool explains its purpose, value, and intended use. High narrative friction happens when developers cannot tell what problem the tool solves, who should use it, or how it compares to alternatives [7].

This is different from documentation quality. Even with excellent technical documentation, a tool can be confusing if its overall purpose or target audience is unclear.

3.5 The Adoption Tax Metaphor

We use the idea of a "tax" to explain three important features of these frictions. First, they are unavoidable costs that developers face when trying or adopting a tool, no matter how good it is technically. Second, these costs add up, multiple frictions together make adoption harder. Third, like taxes, these costs are often invisible or underestimated by tool creators, but they are real for the developers who use the tools.

The framework predicts that the more friction a tool has, the less likely developers are to adopt it, even when the tool is technically strong. It also suggests that lowering just one type of friction may not be enough if the other frictions remain high. In other words, the total, combined burden determines whether a developer chooses to adopt or abandon a tool.

4 Method

4.1 Study Design

We ran an online survey to study how non-technical frictions affect whether developers adopt or abandon tools. All participants gave their consent to take part.

4.2 Participants

We recruited software developers who had tried or evaluated at least one new developer tool in the past 12 months. Participants were invited through developer communities on Twitter/X, LinkedIn, specialized forums, and personal networks. The survey was open globally with no geographic restrictions.

Participants reported their professional experience, main programming languages, organization size, and which tools they had recently evaluated. To be included, participants had to complete all key survey questions for at least one tool.

We received 247 responses. After removing 17 incomplete or inattentive responses, the final sample included 230 developers. The participants came from 34 countries and had between less than 1 year and over 20 years of experience (median = 6 years). Their main programming languages were JavaScript (32%), Python (28%), Java (15%), C# (8%), and others (17%).

4.3 Materials and Measures

The survey included questions about participants' backgrounds, the tools they evaluated, and their experiences with different types of friction, technical quality, and adoption outcomes.

4.3.1 Friction Measures

We created several survey questions for each type of friction, drawing on prior research and feedback from a small pilot study with 12 developers.

Cognitive Friction captures how mentally demanding a tool is to learn and use. We measured this using five items adapted from the NASA Task Load Index [21], adjusted for developer tool adoption. Example items include: "Learning this tool required a lot of mental effort," "Using this tool effectively requires remembering many details," and "I felt mentally overloaded when learning this tool." Responses were collected using a 7-point Likert scale (1 = strongly disagree, 7 = strongly agree). The scale showed high internal consistency. Cronbach's $\alpha = 0.86$.

Social Friction reflects how developers experience interactions with a tool's community and maintainers. We measured this using four items focused on responsiveness, tone, and access to support. Example items include: "The community was responsive to questions and issues" (reverse-scored), "Maintainers communicated in a welcoming and helpful manner" (reverse-scored), and "Community interactions felt hostile or dismissive." This scale demonstrated good reliability. Cronbach's $\alpha = 0.82$.

Institutional Friction captures concerns about a tool's governance, ownership, and long-term stability. We measured this construct using five items, including statements such as: "The tool's governance structure was unclear," "I worried about the long-term maintenance of this tool," and "I had concerns about the tool's future stability." The scale showed strong internal consistency. Cronbach's $\alpha = 0.88$.

Narrative Friction describes how clearly a tool explains its purpose, use cases, and position relative to alternatives. We measured this using four items, including: "The tool's purpose and value proposition were clear" (reverse-scored), "I struggled to understand what problems this tool was meant to solve," and "Comparisons to similar tools were confusing or missing." This scale also showed good reliability. Cronbach's $\alpha = 0.81$.

4.3.2 Technical Adequacy

We measured perceived technical adequacy using four survey items: "The tool met my technical requirements," "The tool performed reliably," "The tool's features were sufficient for my needs," and "The tool had clear technical advantages over alternative tools." These items showed good internal consistency (Cronbach's $\alpha = 0.84$). Technical adequacy was included as a control variable so that we could examine the effects of non-technical frictions independently of technical quality.

4.3.3 Adoption Outcome

Participants reported whether they ultimately adopted or abandoned each tool they evaluated. Adoption was defined as using the tool regularly in at least one project or workflow. Abandonment included cases where participants stopped using the tool after an initial trial, did not complete the adoption process, or decided not to use the tool after evaluation.

4.4 Data Analysis

For each friction dimension, we computed scale scores by averaging the corresponding survey items. We assessed internal consistency using Cronbach's alpha. We then conducted exploratory factor analysis (EFA) with promax rotation to confirm that survey items loaded on the expected factors and that the four friction dimensions were distinct.

To examine adoption outcomes, we used logistic regression to model the likelihood of tool abandonment. The model included the four friction dimensions and technical adequacy as predictors. All predictors were entered simultaneously. We report odds ratios to describe the strength of associations and use Nagelkerke's pseudo-R^2 to assess model fit.

In addition to individual friction dimensions, we created a composite "total friction" score by standardizing and averaging the four friction scales. This allowed us to examine whether the combined level of non-technical friction predicted abandonment beyond the effects of individual dimensions.

All analyses were conducted using R (version 4.3.0). We report two-tailed significance tests with $\alpha = 0.05$.

4.5 Limitations

This study has several limitations. First, the cross-sectional design limits our ability to draw causal conclusions; the results identify associations rather than causal effects. Second, the use of self-reported data may introduce recall or social desirability bias. Third, participants were recruited through convenience sampling, which may limit the the findings. Fourth, the study focuses on developers' perceptions of friction rather than objective measures, although perceptions play a central role in adoption decisions. Finally, we did not account for all possible external factors, such as organizational constraints or switching costs from existing tools.

5 Results

5.1 Sample Characteristics

The final sample included 230 developers who reported on 312 tool evaluation experiences, as some participants evaluated more than one tool. The tools covered a wide range of categories, including testing frameworks (22%), build tools (18%), IDEs and editors (15%), languages and compilers (13%), deployment and infrastructure tools (12%), version control systems (8%), and other tools (12%).

Across the 312 evaluations, 168 cases (54%) resulted in adoption, while 144 cases (46%) resulted in abandonment. This balance indicates that the sample captures both successful and unsuccessful adoption experiences, rather than focusing on only one outcome.

5.2 Reliability and Factor Structure

All survey scales showed acceptable to good internal consistency. Cronbach's alpha values were 0.86 for cognitive friction, 0.82 for social friction, 0.88 for institutional friction, 0.81 for narrative friction, and 0.84 for technical adequacy.

We conducted an exploratory factor analysis using promax rotation. The analysis identified four factors that aligned with the proposed friction dimensions and together explained 71% of the total variance. Most survey items loaded strongly on their intended factors (loadings above 0.60),

with limited cross-loadings. These results support treating the four friction dimensions as distinct constructs.

5.3 Descriptive Statistics and Correlations

Table 1 summarizes the descriptive statistics for all measures. Cognitive friction had the highest average score ($M = 4.23$, $SD = 1.34$), suggesting that learning effort is a common challenge when developers try new tools. Institutional friction showed the greatest variation across tools ($SD = 1.52$), indicating that developers' views on governance and long-term stability differ widely between tools.

Table 1: Descriptive Statistics and Correlations

Variable	M	SD	1	2	3	4	5
1. Cognitive Friction	4.23	1.34	—				
2. Social Friction	3.87	1.41	.43**	—			
3. Institutional Friction	4.01	1.52	.38**	.51**	—		
4. Narrative Friction	3.65	1.38	.41**	.46**	.49**	—	
5. Technical Adequacy	5.12	1.28	-.31**	-.28**	-.33**	-.29**	—

Note: $N = 312$ tool evaluations. ** $p < 0.01$

The different types of friction were moderately related, with correlations ranging from $r = 0.38$ to $r = 0.51$. This means that while the frictions often appear together, each type still captures something distinct. Technical adequacy was negatively correlated with all friction types ($r = -0.28$ to -0.33), suggesting that tools with higher technical quality tend to have lower friction. However, these correlations are modest, which shows that friction and technical quality vary independently to a meaningful degree.

5.4 Logistic Regression Results

Table 2 shows the results of our logistic regression predicting whether developers abandoned a tool. The model, which includes all four types of friction and technical quality, was statistically significant ($\chi^2(5) = 156.32$, $p < 0.001$) and explained a substantial part of adoption outcomes (Nagelkerke $R^2 = 0.42$).

Table 2: Logistic Regression Predicting Tool Abandonment

Predictor	B	SE	Wald	p	OR [95% CI]
Cognitive Friction	0.85	0.21	16.43	$< .001$	2.34 [1.55, 3.53]
Social Friction	0.48	0.19	6.40	.011	1.62 [1.12, 2.35]
Institutional Friction	1.05	0.23	20.81	$< .001$	2.87 [1.83, 4.51]
Narrative Friction	0.62	0.20	9.61	.002	1.86 [1.26, 2.75]
Technical Adequacy	-0.71	0.18	15.56	$< .001$	0.49 [0.34, 0.70]

Note: $N = 312$. Model $\chi^2(5) = 156.32$, $p < 0.001$, $R^2_N = 0.42$

All five factors we studied were statistically significant. Institutional friction had the strongest link to tool abandonment (OR = 2.87, 95% CI [1.83, 4.51]), meaning that each increase in institutional friction nearly tripled the odds that a developer would abandon a tool. Cognitive friction

also had a strong effect (OR = 2.34, 95% CI [1.55, 3.53]). Narrative friction (OR = 1.86, 95% CI [1.26, 2.75]) and social friction (OR = 1.62, 95% CI [1.12, 2.35]) were smaller but still meaningful predictors.

Technical adequacy also predicted adoption (OR = 0.49, 95% CI [0.34, 0.70]). This means that higher technical quality lowered the chance of abandonment. However, its effect was similar in size to the individual friction factors, showing that non-technical frictions are just as important as technical performance in explaining adoption outcomes.

5.5 Cumulative Friction Effect

We tested whether a combined measure of total friction, the average of all four friction types, predicted tool abandonment beyond technical quality alone. A model with only technical adequacy explained $R^2 = 0.18$ of the variance. Adding the total friction measure increased the model fit to $R^2 = 0.39$ ($\Delta\chi^2 = 98.45$, $p < 0.001$), supporting the idea that frictions act like a cumulative "tax" on adoption.

The total friction score ranged from -1.52 to +2.18 ($M = 0.00$, $SD = 1.00$). Even for tools with high technical quality (top 25%), abandonment rates varied from 12% for tools with low friction to 61% for tools with high friction. This shows that non-technical frictions have a strong influence, even when technical performance is good.

5.6 Exploratory Analyses

We also explored whether the impact of frictions differed by developer experience or type of tool. Developer experience did not significantly change the effect, suggesting that frictions matter for both new and experienced developers. Tool type showed some differences: institutional friction was especially important for infrastructure and build tools ($OR = 3.45$), while cognitive friction mattered more for programming languages and IDEs ($OR = 2.89$). Because these analyses were exploratory and sample sizes were small within each category, these results should be interpreted cautiously.

6 Discussion

This study looked at how non-technical frictions affect developer tool adoption. Our main finding is that cognitive, social, institutional, and narrative frictions together predict whether a tool is abandoned, even after accounting for technical quality. Among the four, institutional friction concerns about governance, stability, and long-term support was the strongest single predictor. These results suggest that developers' adoption decisions depend not just on technical features but also on learning effort, community support, governance trust, and how clearly a tool communicates its purpose.

6.1 Interpretation of Findings

6.1.1 RQ1: Which non-technical frictions most strongly predict abandonment?

Institutional friction which includes governance clarity, perceived stability, and organizational support had the strongest link to abandonment (OR = 2.87). This agrees with prior research showing that developers are careful when picking tools that may become central to their workflow [19]. Uncertainty about who maintains a tool, how decisions are made, or whether it will be supported long-term can be a major risk, even if the tool works well technically.

Cognitive friction also predicted abandonment (OR = 2.34). This means the effort required to learn a tool is still a big barrier, even for experienced developers. High cognitive friction is especially challenging when developers have limited time or when organizations expect fast results from new tools [8].

Social and narrative friction had smaller, but still significant, effects (ORs = 1.62 and 1.86). These factors matter, but developers may tolerate a weak community or unclear instructions if institutional and cognitive frictions are low.

6.1.2 RQ2: How do non-technical frictions compare to technical adequacy?

Technical adequacy also affected adoption (OR = 0.49), confirming that good technical quality matters. However, its effect was similar to individual friction dimensions and did not outweigh them. The full model including frictions fit much better ($R^2 = 0.42$) than technical adequacy alone ($R^2 = 0.18$).

These findings suggest that technical quality is necessary but not enough. Developers seem to consider both technical features and the overall burden of non-technical frictions when deciding whether to adopt a tool.

6.1.3 RQ3: Can the cumulative effect be interpreted as an adoption "tax"?

The combined friction measure predicted abandonment even after accounting for technical quality. Abandonment rates differed by 49 percentage points (12% vs. 61%) between tools with low and high friction, even when technical quality was good. This supports the idea of an "adoption tax". Non-technical frictions add up and influence developers' cost-benefit decisions.

The "tax" also highlights that these costs are often invisible. Tool creators may underestimate them because they are familiar with their tools and supported by active communities. For new users, however, these frictions represent real time and effort that affect adoption.

6.2 Implications for Research

These findings suggest several directions for future work. First, longitudinal studies could track how non-technical frictions change over a tool's lifecycle and whether some frictions matter more at different stages of adoption. Second, qualitative studies could explore how developers weigh technical benefits against non-technical costs when deciding which tools to adopt. Third, experimental or quasi-experimental designs could test whether interventions targeting specific frictions help increase adoption.

The IAT Framework could also be expanded to include other types of friction. For example, economic friction (licensing costs or vendor lock-in), ecosystem friction (how well a tool integrates with existing workflows), or timing friction (how well adoption fits project deadlines or organizational readiness). Researchers could also explore situations where technical quality is the main driver or where particular frictions are more important.

Finally, our study shows the value of looking at both tools that were adopted and tools that were abandoned. Focusing only on successful tools may miss key factors that cause failure.

6.3 Implications for Practice

For developers building tools, these findings suggest several practical steps. First, reduce institutional friction. Make governance clear, commit to maintenance, and show credible organizational

backing. Tools from small teams or individuals may fail not because they are technically weak but because users perceive them as risky.

Second, reduce cognitive friction. Provide good learning resources, intuitive design, and a gradual way for users to learn complex features. Onboarding should help users get productive quickly while allowing deeper learning over time [15].

Third, reduce social friction. Encourage welcoming and helpful communities. Enforce codes of conduct, train maintainers to communicate clearly, and actively manage community interactions [6].

Finally, focus on narrative clarity. Clearly explain what problems the tool solves, who it is for, and how it compares to alternatives. This helps potential users make informed decisions quickly.

For organizations evaluating tools, recognize that adoption costs include more than technical quality or licensing fees. Consider the time needed to learn the tool, the effort required to engage with the community, and risks related to stability or governance.

6.4 Limitations and Future Directions

This study has some limitations that future work should address. First, the cross-sectional design only shows associations. We cannot say for certain that frictions cause tool abandonment. Tools that were abandoned might simply look worse in hindsight. Following developers over time could clarify how frictions influence adoption decisions.

Second, we relied on self-reported perceptions, which may not match objective behavior. Future research could combine surveys with behavioral data, such as time to productivity, community interactions, or how documentation is used.

Third, our sample was recruited through convenience sampling and may not represent all developers. People from underrepresented groups might experience frictions differently. Studying diversity-related differences in friction would be valuable.

We also did not account for organizational factors, switching costs, or network effects, which can affect adoption. Future work could examine how teams or organizations influence friction or how frictions interact with coordination costs.

Finally, we focused on individual decisions. Team and organizational adoption is more complex. Understanding how frictions affect collective decisions or propagate through teams remains an open question.

7 Conclusion

This paper introduced the Invisible Adoption Tax (IAT) Framework and showed that non-technical frictions matter for developer tool adoption. Cognitive, social, institutional, and narrative frictions together help explain adoption beyond technical quality. Institutional friction such as governance clarity and perceived stability, was the strongest predictor of abandonment.

Our findings show that technical quality alone does not guarantee adoption. Developers weigh technical benefits against non-technical costs, and even good tools may fail if frictions are high. The “adoption tax” metaphor captures how these costs accumulate across different dimensions.

For researchers, the IAT Framework offers a simple way to study adoption through the lens of friction. For practitioners, the results suggest that reducing frictions through clear governance, supportive communities, easy learning curves, and coherent messaging, can be as important as technical improvements for success.

As developer tools continue to grow and technical differences become smaller, understanding and reducing the invisible adoption tax may determine which tools thrive and which do not.

References

[1] L. A. Meyerovich and A. S. Rabkin, "Empirical analysis of programming language adoption," in *Proc. 2013 ACM SIGPLAN Int. Conf. Object Oriented Programming Systems Languages & Applications*, 2013, pp. 1–18.

[2] A. Rabkin and R. Katz, "Static extraction of program configuration options," in *Proc. 2010 Int. Conf. Software Engineering*, 2020, pp. 131–140.

[3] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quarterly*, vol. 27, no. 3, pp. 425–478, 2003.

[4] E. M. Rogers, *Diffusion of Innovations*, 5th ed. New York: Free Press, 2003.

[5] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *Proc. 2020 IEEE/ACM 42nd Int. Conf. Software Engineering*, 2020, pp. 1199–1210.

[6] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "The (r)evolution of social media in software engineering," in *Proc. Future of Software Engineering*, 2014, pp. 100–116.

[7] S. Nakshatri, M. Hegde, and S. Thandra, "Analysis of exception handling patterns in Java projects: An empirical study," in *Proc. 2016 IEEE/ACM 13th Working Conf. Mining Software Repositories*, 2020, pp. 500–503.

[8] G. C. Murphy, M. Kersten, and L. Findlater, "How are Java software developers using the Eclipse IDE?" *IEEE Software*, vol. 23, no. 4, pp. 76–83, 2006.

[9] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, vol. 13, no. 3, pp. 319–340, 1989.

[10] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.

[11] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," *Georgia Institute of Technology, Tech. Rep.*, 2020.

[12] A. Decan, T. Mens, and M. Claes, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, 2020.

[13] J. Nielsen, *Usability Engineering*. San Francisco: Morgan Kaufmann, 1994.

[14] D. A. Norman, *The Design of Everyday Things: Revised and Expanded Edition*. New York: Basic Books, 2013.

[15] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in *Proc. 2005 IEEE Symp. Visual Languages and Human-Centric Computing*, 2020, pp. 207–214.

[16] B. S. Lerner, H. Krishnamurthi, and S. Krishnamurthi, "What do beginning programmers need to know about recursion?" in *Proc. 2020 ACM SIGCSE Technical Symp. Computer Science Education*, 2020, pp. 1091–1097.

[17] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[18] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2020.

[19] J. Lerner and J. Tirole, "Some simple economics of open source," *The Journal of Industrial Economics*, vol. 50, no. 2, pp. 197–234, 2002.

[20] S. Raemaekers, A. van Deursen, and J. Visser, "Semantic versioning versus breaking changes: A study of the Maven repository," in *Proc. 2014 IEEE 14th Int. Working Conf. Source Code Analysis and Manipulation*, 2020, pp. 215–224.

[21] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," in *Advances in Psychology*, vol. 52, 1988, pp. 139–183.