Received IEEE TCSE Distinguished Paper Award at ICMSE'17

Software Practitioner Perspectives on Merge Conflicts and Resolutions

Shane McKee*, Nicholas Nelson*, Anita Sarma, and Danny Dig
Oregon State University, Corvallis, OR
{mckeesh, nelsonni, anita.sarma, digd}@oregonstate.edu

***Abstract*—Merge conflicts occur when software practitioners need to work in parallel and are inevitable in software development. Tool builders and researchers have focused on the prevention and resolution of merge conflicts, but there is little empirical knowledge about how practitioners actually approach and perform merge conflict resolution. Without such knowledge, tool builders might be building on wrong assumptions and researchers might miss opportunities for improving the state of the art.**

We conducted semi-structured interviews of 10 software practitioners across 7 organizations, including both open-source and commercial projects. We identify the key concepts and perceptions from practitioners, which we then validated via a survey of 162 additional practitioners.

We find that practitioners are directly impacted by their perception of the complexity of the conflicting code, and may alter the timeline in which to resolve these conflicts, as well as the methods employed for conflict resolution based upon that initial perception. Practitioners' perceptions alter the impact of tools and processes that have been designed to preemptively and efficiently resolve merge conflicts. Understanding whether practitioners will react according to standard use cases is important when creating human-oriented tools to support development processes.

I. Introduction

Collaborative development is essential for the success of large projects [1], and is enabled by version control systems. In Git and other version control systems, practitioners work on their changes in seclusion, and periodically synchronize them by merging with the main line of development. Although a large number of commits cleanly merge, parallel changes can overlap, leading to merge conflicts. Kasi et al. [2] and Brun et al. [3], in their studies of several open source projects, found that merge conflicts occur in approximately 19% of all merges.

Resolving merge conflicts is nontrivial, especially when changes diverge significantly, making their synchronization difficult. The resolution process can be tedious and can cause delays as practitioners figure out how to approach and resolve conflicts [2]. Poorly-performed merge conflict resolutions have been known to cause integration errors [4], workflow disruptions, and jeopardize project efficiency and timelines [5].

Practitioners are aware of the merge-resolution "pains" and follow different informal processes to avoid having to resolve conflicts; e.g. sending out emails to the rest of the team, performing partial commits, or racing to finish changes [6][7]. Unfortunately, these practices cause changes to diverge more and makes merge conflict resolution even harder [3].

Past work has examined different mechanisms for proactive conflict detection [3][8][9], proposed tools for resolving merge conflicts [10][11], and discussed advantages of syntax- and semantic-aware merges [12][13]. However, there are several key questions that remain unanswered: How do practitioners actually approach merge conflicts? How do practitioners perceive the difficulty of a merge conflict resolution? Do the current tools support practitioners' conflict resolution needs? Without such knowledge, tool builders might be building on wrong assumptions and researchers might miss opportunities for improving the state of the art.

To answer the above key questions, we talked directly to practitioners, which is crucial to understanding problems in context [14]–[17]. We interviewed 10 software practitioners from 7 organizations about their experiences and perceptions of merge conflicts in the software development process. Our participants had a median of 5 years of software development experience, and work on a mix of both small-scale projects (<10 contributors) and large-scale projects (>1000 contributors). These interviews helped us understand how practitioners approach merge conflicts, and their unmet needs.

To triangulate our findings and provide a broader understanding of practitioners' perceptions of merge conflicts and their difficulty, we deployed a survey to a larger population of software practitioners. The survey sampled 162 participants, spanning both open source and commercial projects. 74.2% of our participants had 6 or more years of software development experience, and reported that they face merge conflicts a few times a week.

To understand the effects and implications of software practitioner perceptions, we answer the following research questions:

- **RQ1:** *How do software practitioners approach merge conflicts?*
- **RQ2:** *What unmet needs impact the difficulty of a merge conflict resolution?*
- **RQ3:** *How well do tools meet practitioner needs for merge conflicts?*

We found that practitioners, when initially assessing a merge conflict, focus on the *code complexity of the conflicting lines* and *their own knowledge in the area of the conflict* as the top two factors when estimating the difficulty of a conflict. These

* First and second author contributed equally to this work.

concerns cause practitioners to alter their resolution strategy, and in some cases delay resolution.

After understanding the merge conflict, practitioners must resolve the conflict in order to return to normal development. We found that the key challenges that practitioners face when resolving conflicts is *understanding the conflicting code*, *their knowledge in the area of code conflict*, and having enough meta information about the conflicting code (who made the change, why, etc).

We found that development tools struggle to address practitioners' perceptions that increases in merge conflict complexity have a greater impact on the conflict difficulty than increases in size. This could partially be alleviated by focusing on the tool improvements most desired by practitioners: *better usability*, *better information filtering*, and *better history exploration*.

This paper makes the following contributions:

- We conduct exploratory semi-structured interviews with 10 software practitioners, then confirm these findings with a survey of 162 practitioners from around the world.
- We provide empirically-derived rankings of merge conflict difficulties based on practitioners' perceptions.
- We expose disparities between practitioners' merge conflict needs and development toolsets.

II. Methodology

We use mixed methods to first explore the topic of practitioners' difficulties when encountering merge conflicts through a set of exploratory interviews, and then validate our findings through a survey of practitioners, as per guidelines by Easterbrook et al. [18]. Mixed methods allow us to analyze both qualitative and quantitative data to get both an individual and a population-wide perspective of software practitioners.

We codified the interview transcripts into a taxonomy of barriers, constraints, and concerns following established guidelines [19]–[21]. We then created a survey to get a broader perspective on how software practitioners approach merge conflicts, using the vocabulary generated from the interviews to create the survey questions.

A. Terminology

For this work, we define merge conflict as a scenario in which two copies of the same codebase diverge and cannot be automatically merged, thus requiring human intervention to resolve. While we recognize that other types of conflicts exist in software projects (i.e. social conflicts or semantic conflicts, such as build or test failure), we focused our interviews and survey on code-related merge conflicts.

B. Interviews

We conducted semi-structured interviews with software practitioners to understand their concerns when facing merge conflicts and the factors that impact merge conflict difficulty.

We selected interview participants from top contributors to open-source projects, and from industry contacts using snowball sampling [22] to reach a larger sample size. Each participant was asked to identify additional practitioners for recruitment to our study. We interviewed ten software practitioners from seven different organizations spanning six different industries: *Semiconductor Manufacturing* (3 participants), *Healthcare Software* (2), *Academia* (2), *Business Software* (1), *IT Services* (1), and *Sports & Wellness Technology* (1). Each participant was asked to describe their role within their organization, resulting in five roles with multiple responses for *Software Engineer / Developer*. Table I provides additional demographics data, including project sizes and whether the participant primarily focuses on open- or closed-source software development.

TABLE I: Interview Participant Demographic

Ptc.[i]	**Exp.**[ii]	**Role**	**Project Source**	**Project Contrib.**[iii]
P1	18 yrs.	Sr. Software Developer	Open	1700
P2	6 yrs.	Software Engineer	Open	1700
P3	3 yrs.	Software Engineer	Open	1700
P4	10 yrs.	Software Developer	Open	<10
P5	3 yrs.	Infrastructure Engineer	Closed	<10
P6	5 yrs.	Software Developer	Closed	<10
P7	5 yrs.	Software Engineer	Open	200
P8	25 yrs.	Director	Open	600
P9	8 yrs.	Software Developer	Open	600
P10	2 yrs.	Software Developer	Open	<5

[i] Ptc. = Participant [ii] Exp. = Years of Software Development Experience
[iii] Project Contrib. = Approximate number of contributors (between March 2016-March 2017)

Each interview lasted between 30 to 60 minutes. Participants were offered US$50 in either cash, gift card, or a donation to a charity of their choice.

At the beginning of the interview we gave participants a short explanation of the research goals, our definition of merge conflicts, and collected demographics data. We then asked participants about the roles that they play in their project, their experience working in team settings, questions about merge conflicts, the process of conflict resolution, and the difficulties that they faced in conflict resolution.

We formulated the interview questions about merge conflicts in order to understand how practitioners perceived and how they approached merge conflicts. The following is an example of some of the questions we asked in the interview. The full set of questions can be found in our companion site [23].

- Can you describe a merge conflict, or a set of conflicts, that you would consider to be the typical case?
- Do you have any particularly memorable merge conflict resolutions that you can recall?
- Have you had some code structures, design patterns, coding styles, etc., that you would consider a "usual suspect" in a conflict?
- What kind of measures would you take to minimize the amount of defects that you introduce?

The semi-structured interview format allowed participants to provide us with unanticipated information [24]. Further,

we allowed open-ended discussion about merge conflicts in general at the end of the interview, which allowed participants to share ideas and topics that they found particularly important. We continued interviewing participants until we reached saturation in the answers [25].

Analysis: Interviews were audio-taped and transcribed. The first two authors unitized [26] the interview transcripts into cards that each contained a single logically consistent statement. To organize these cards we employed card sorting, a collaborative technique of exploring how people think about a certain topic [27][28], which allows key concepts and associations to be identified through an open sorting method that iteratively develops categories during the process.

We performed two iterations of the open card sorting process. In the first iteration, we developed a standardized coding scheme and improved it to an acceptable point by *negotiated agreement*, which was reached when no further thematic categories could be created and agreed upon by both coders [29][30]. The coding scheme dictated that sentences must be consecutive and topically related to be grouped into a single card. Logically connected statements that were separated by other lines were considered to be separate cards, as a conservative measure to preserve context within each card.

In the second iteration, the first two authors sorted cards according to our coding scheme and discussed the resulting taxonomies until consensus was reached. Based upon our research questions, we grouped the resulting categories as follows: the factors that impact how practitioners approach merge conflicts (Section III-A), the difficulties that practitioners face when resolving conflicts (Section III-B), and the impact of development tools on the resolution process (Section III-C).

TABLE II: Survey Participant Roles[i]

	Soft. Developers	Sys. Architects	DevOps	Project Managers	Project Maintainers	Sys. Admins	Other
Software Developers	154						
System Architects	53	54					
DevOps	51	34	53				
Project Managers	44	29	20	44			
Project Maintainers	39	21	24	22	40		
Systems Administrators	22	16	15	14	12	23	
Other	8	4	4	3	1	2	11

[i] Survey respondents were allowed to select multiple roles. Each entry represents the number of respondents that selected both of the roles indicated for the column and row. 62 out of 162 respondents (38%) selected three or more roles.

C. Survey

We conducted a 50-question survey of software development practitioners in order to examine the themes and categories found in the interviews. We sought to understand which factors impact practitioners the most when they encounter and resolve merge conflicts. The survey was conducted online and anonymity was guaranteed in order to elicit honest responses from participants. We developed questions to confirm, extend, and broaden the results from the interviews.

We recruited survey participants from contributor lists on popular open-source repositories on GitHub[1], advertised on social networking sites (Facebook[2] and Reddit[3]), and by directly contacting software practitioners via email. Due to the nature of social media and mailing lists, we cannot compute a response rate. However, our participants spanned different organization structures and geographical locations, giving us generalizability of results. The survey was available for 56 days and we received 162 survey responses, but individual parts of the survey have varying response rates and are reported where appropriate in Section III.

Survey participants were primarily male (91.9% overall). Participants were given six different software roles to select, and in many cases, participants considered themselves to be fulfilling multiple roles. Table II provides a pairwise breakdown of participants' role selections, with a majority of respondents considering themselves to be *Software Engineer/Developer* (95.1% overall). Survey participants indicated a median software development experience of 6-10 years (36.4% overall), and worked on project sizes ranging from 2 to more than 51 developers (the median was 2-5 developers, constituting 48.8% of all responses).

The survey was divided into four categories, with each category containing 5-7 questions (see [23] for questions). First, we elicited background information about demographics, roles, and experience. Second, we asked questions related to difficulties that practitioners experience when encountering merge conflicts. Third, we asked questions related to conflict resolution and the factors that affect practitioners. Finally we asked questions about the tools and tool features that practitioners use when working with merge conflicts. Questions were presented either as 5-point Likert-type scales (with no pre-selected answers) or open-ended text forms to gather additional insights.

Analysis: We evaluated the distribution of survey answers for each Likert-type question by analyzing across demographic categories. Where answers differed across a demographic category, we note the difference and provide further discussion of these results in Section III.

We used Likert-type questions to measure the extent to which participants agreed with a particular statement. This means that lower mean and median values indicate less agreement with the statement in a particular question. We use this design to validate both the degree of agreement to the interview results, as well as the existence of individual factors.

III. Results

A. *RQ1:* How do practitioners approach merge conflicts?

To understand the perspective of practitioners when encountering a merge conflict, we asked interview participants to reflect on situations when they initially face a merge conflict:

[1] github.com [2] facebook.com [3] reddit.com

TABLE III: Factors of Merge Conflict Difficulty from Survey

Factor	Description	1	2	3	4	5	Mean	Median
F1	**Complexity of conflicting lines of code**	**5**	**29**	**38**	**56**	**34**	**3.52**	**4**
F2	**Your knowledge/expertise in area of conflicting code**	**5**	**23**	**50**	**54**	**30**	**3.50**	**4**
F3	**Complexity of the files with conflicts**	**8**	**34**	**49**	**51**	**18**	**3.23**	**3**
F4	**Number of conflicting lines of code**	**2**	**40**	**64**	**45**	**11**	**3.14**	**3**
F5	Time to resolve a conflict	14	56	51	25	15	2.82	3
F6	Atomicity of changesets in the conflict	20	48	51	29	13	2.80	3
F7	Dependencies of conflicting code on other components	20	56	39	33	14	2.78	3
F8	Number of files in the conflict	10	69	50	26	6	2.68	3
F9	Non-functional changes (whitespace, renaming, etc.)	47	63	31	15	4	2.16	2

what kind of information do they seek, how do they approach the resolution of the conflict, and what tools they use.

We identified nine factors (from card sorting) that practitioners consider when approaching a conflict and attempting to determine its difficulty (see Table III). We asked survey participants to rate how each of these nine factors affected their perceptions of difficulty when approaching a merge conflict.

We received 162 responses and present the aggregated results in Table III; ranked according to the mean score for each factor. Here, we discuss in detail the top 4 factors with a mean score greater than 3.00. These factors can be grouped into themes of *technical aspects* and *expertise*, and our results are presented according to these groups.

Technical Aspects: Two of the top four factors refer to perceptions about the complexity of merge conflicts (F1, F3), with the third factor being *number of conflicting lines* (F4), which can be construed as a specific metric of complexity of the conflict. While practitioners mentioned complexity of the lines of code and the file, none mentioned using metrics, such as cyclomatic complexity [31][32] or Function Point Analysis [33][34]. Instead, practitioners made educated guesses on the complexity of the code based on their own experience of either writing the code, or having worked with it. Some of the simple to compute metrics, such as *number of conflicting lines of code* (F4), *number of files involved* (F8), *atomicity of changesets* (F6), and *the time* they thought it would take *to resolve the conflict* (F5) were mentioned. The only factor where static analysis tools can help was in identifying the *dependencies of the conflicting code* (F7). This indicates that understanding the complexity of the conflicting code is important, but practitioners do not use the metrics that have been proposed by research. While some of the simple proxies for complexity are used, practitioners primarily rely on their own "judgement" of the complexity of the conflict.

This perception of the conflict complexity can affect whether a practitioner resolves the conflict immediately (when small), or whether they should wait to examine the conflict when further resources are available; P8 commented:

> *"Small is always easy. A 1-line merge conflict is always easier to resolve than a 400-line merge conflict, and can be done now."*

If a merge conflict is perceived to be large or complex, a practitioner may decide to forgo attempting to resolve it through code manipulation and choose to revert the changes instead [35]. This "nuclear option" requires practitioners to disrupt the development flow, set aside their current development work, and potentially remove "good" code that was not part of the conflict in order to return to a non-conflicting state. In the interview, P1 describes this process as:

> *"If you have many conflicts involved, many commits in the conflict...throw one of the branches away. You cannot combine tens of commits conflicting...it's not sane!"*

Further, when integrators are preparing code for production environments they prioritize merge conflicts for code review based upon the perceived difficulty of resolving the affected code. We find that these decisions rely on human judgement factors as much as they rely on data-driven metrics. Practitioners may not have the time to compute project-wide complexity metrics, such as those proposed in literature. Therefore, we need metrics that can be easily calculated by "lay practitioners" as they face a conflict.

Expertise: Our findings show that expertise in the area of conflicting code is one of the top factors in determining the difficulty of a merge conflict (F2). This reiterates the fact that practitioners rely on their own knowledge about the conflicting code base when approaching a conflict.

Our results indicate that when practitioners feel they don't have the expertise in the conflicting code base, they consider the conflict difficult to merge and seek out more information or assistance from others. P5 illustrated this need for expertise when describing his workflow:

> *"A lot of what I work on is in my own little area...I know what to do... But in [unfamiliar part of code], then I'll get someone else to resolve the merge conflict for me. It's someone else's code, and I don't want to screw it up."*

Our findings confirm the need for tools that identify appropriate experts [36] and encourage further research into selection of knowledgeable practitioners for merge conflict resolution.

B. ***RQ2:*** *What unmet needs impact the difficulty of a conflict resolution?*

There can often be gaps in how practitioners perceive the difficulty of merge conflicts and the actual hurdles that they

TABLE IV: Practitioners' Needs for Merge Conflict Resolutions from Survey

Need	Description	1	2	3	4	5	Mean	Median
N1	**How easy is it to understand the code involved in the merge conflict**	**0**	**14**	**25**	**65**	**37**	**3.89**	**4**
N2	**Your expertise in the area of code with the merge conflict**	**1**	**17**	**38**	**49**	**36**	**3.72**	**4**
N3	**The amount of information you have about the conflicting code**	**2**	**21**	**38**	**48**	**32**	**3.62**	**4**
N4	**How well tools present information in an understandable way**	**4**	**24**	**47**	**32**	**34**	**3.48**	**3**
N5	Changing assumptions within the code	8	27	45	36	25	3.30	3
N6	Complexity of the project structure	6	38	39	41	17	3.18	3
N7	Trustworthiness of tools	17	29	39	32	34	3.12	3
N8	Informativeness of commit messages	18	32	30	44	17	3.07	3
N9	Project culture	13	37	43	27	21	3.04	3
N10	Tool support for examining development history	16	40	31	32	22	3.03	3

face when resolving these conflicts. These gaps can then in turn affect how well practitioners can resolve the conflict.

We, therefore, asked our interview participants open-ended questions about their experiences in resolving the most recent past conflicts, especially their recollection of what made the resolution difficult. Their responses indicated that there are several unmet needs. We identified ten needs (see Table IV), which range from needs about the ability to understand the code, their expertise, and existing tool support.

Using results from the interview, we asked survey participants to rate how much each of the ten needs affected their ability to resolve merge conflicts. We received 141 responses using a 5-point Likert-type scale indicating the degree of effect on resolution difficulty (1 being *Not at all*, 3 being *A moderate amount*, and 5 being *A great deal*). Results of the survey are presented in Table IV.

All the unmet needs have a mean score of at least 3.03 on the 5-point Likert-type scale, implying that all of them mattered at least a moderate amount. We present and discuss in detail the top four unmet needs, plus additional observations regarding the other six unmet needs. As with the factors in the previous section, all these needs also relate to *technical aspects* (e.g., understanding the conflicting code) and their *expertise* in resolving conflicts.

Technical Aspects: Three needs among the top four relate to technical aspects of merge conflict resolution. The *understandability of conflicting code* (N1) is ranked as the most important need, with both *contextual information about the conflict* (N3) and *the way in which tools present relevant information* (N4) ranking in the top four.

Data from version control systems is used by practitioners to identify the evolution of the code [37], however, it is not easily available and requires a context switch from the code editor to the version control system [35]. Moreover, these changes are often processed in isolation, especially when there are many changes (conflicts) to process. Such decomposition of overall conflicting changes into smaller "chunks" is needed to be able to manage the complexity of the resolution process; however, this occludes viewing the changes and their impact in a holistic manner. Often practitioners deal with the decomposed (smaller) changes, hoping that they will all together work out. For example, P1 compared the resolution hurdles between two conflicts, where one was simple, and the other spanned multiple files and complex blocks of code.

> *"You focus on understanding the small change, not the big one. It's easier to understand... get the small change to go with the flow of the bigger change."*

Another challenge when viewing changes in isolation, is the fact that practitioners may miss the impact of the changes made as part of the resolution to the rest of the code base. Identifying the impact of changes on the rest of the code base has been repeatedly found to be a problem in collaborative development, as found by deSouza and Redmiles [38] and more recently by Guzzi et al. [35]. The top unmet needs in our study too revolved around the challenges that practitioners face in how much information they had about the conflicting code (N3), and the difficulty in finding the needed information from current tools and practices (N3, N4, N8, N10). This indicates that despite advances in supporting parallel development practices, the right information needed to resolve conflicts is still not easily available to practitioners.

Conflict resolution can sometime lead to defects in the code base. This can arise because of several reasons. For example the rationale of the two conflicting changes might be unclear and the merge might cause unintentional problems down the line. Or the resolved changes might not follow rigorous code review and testing to which the original changes were subject to. Therefore, even when the practitioner understands the particular conflicting code, they may still need additional meta information about the rationale of changes and idea of future feature implementation. This is especially true in situations where the code base is old, and such information not readily available. During our interview, P7 commented:

> *"It's harder to merge code when you're merging in some legacy code... But if you're a young team, and everybody who wrote the code is still a part of the team, it's easier."*

Expertise: Knowledge is a key component of practitioner's needs when resolving merge conflicts, but along with that general knowledge is a need for expertise in the specific areas of code involved in a conflict. Practitioners recognize this need as having a sizable effect on their ability to resolve a merge

TABLE V: Improvements for Practitioner Toolsets from Survey

Improvement	Description	1	2	3	4	5	Mean	Median
I1	**Better usability**	**6**	**17**	**32**	**48**	**16**	**3.43**	**4**
I2	**Better ways of filtering out less relevant information**	**8**	**15**	**32**	**48**	**16**	**3.41**	**4**
I3	**Better ways of exploring project history**	**7**	**21**	**36**	**39**	**16**	**3.30**	**3**
I4	**Better graphical presentation of information**	**13**	**26**	**26**	**37**	**16**	**3.14**	**3**
I5	Better transparency in tool functionality/operations	16	36	24	40	3	2.82	3
I6	Better terminology that is more consistent with my other tools	23	41	32	15	8	2.53	2

conflict, and selected *expertise in the area of conflicting code* (N2) as the second most important need.

Examining code artifacts, reviewing change history, and reading documentation help with understanding the code when they are present and well-maintained. However, locating and maintaining these supporting documents is not always possible. In fact, Forward et al. [39] conducted a survey of 48 software practitioners and found that 68% either agreed or strongly agreed that documentation is always outdated. When these gaps arise, practitioners compensate by consulting experts in the area of conflicting code instead.

This result aligns with the goals of the TIPMerge tool [36], which seeks to locate experts that are best suited to resolve conflicts in a particular area of code. However, TIPMerge, as well as other recommendation tools are not being used by real-world practitioners, as evidenced by the lack of such tools in the list of top 10 merge tools (see Table VI). The reason for this lack of research tools adoption requires further investigation.

Another surprising fact was that while the informative nature of commit messages (N8) and project culture (N9) were mentioned, they were not as highly ranked. We had expected them to be higher based on prior work [40]–[43]. We found no statistical differences between commercial or open source projects, including when accounting for experience levels. Our results indicate that team practices, including writing commit messages may have matured enough, such that these factors are no longer considered critical in our sample set.

Open-Source vs. Closed-Source Needs: It is interesting to note that for needs N1-N8 there was no statistical difference between practitioners focused on open-source and those focused on closed-source development when it comes to their conflict resolution needs. We found that practitioners who focus on open-source software development consider *tool support for examining development history* (N10) to be the 3rd highest unmet need (mean: 3.60). Whereas, practitioners who focus on closed-source software development consider it to be the least impactful unmet need (mean: 2.86).

This was also true in our interviews, with P8 stating:

> *"I'm often dealing with code other people wrote. Nobody can review every pull request. So now I have to go back and do some archaeology to find out what's going on. Code is much easier to write than read."*

This result suggests that history exploration in open-source projects is a more difficult task due to the lack of upfront planning and large number of volunteering contributors.

C. *RQ3: How well do tools meet practitioner needs in resolving merge conflicts?*

Development tools need to be easy to use and provide contextualized, pertinent information in a manner that is easy to understand. To investigate how well current tools satisfy the needs of practitioners, we asked interview participants open-ended questions about how they resolve merge conflicts. We also ask about improvements that would be most valuable to them.

Our results indicate that practitioners use a wide range of tools, with many directly using the Git command line interface. Our interview participants mentioned six different dimensions along which they would like improvements to tool support (see Table V).

We framed the survey questions to validate the improvement needs expressed in our interviews, and ranked those six needs according to mean score. Table V presents the needs from the survey responses ordered by their mean scores. We received 119 responses using a 5-point Likert-type scale to indicate the usefulness of each type of tool improvement (1 being *Not Useful*, 3 being *Moderately Useful*, and 5 being *Essential*).

In addition, we also asked participants which tools they use during conflict resolution. We identified 105 different tools from the 115 responses. Some mentioned generic responses such as *" text editor"*, for which we create a separate category. Table VI lists the top 10 most common tools used by participants to resolve merge conflicts.

In examining the list of these tools, we note that practitioners most often use basic tools (e.g. Git, Vim/vi, or a Text Editor) to handle merge conflicts instead of employing specialized tools or plugins to modern IDEs. In this list, there is only one IDE (Eclipse), and three diff/merge toolsets (Git Diff, KDiff3, and Meld). This indicates that practitioners are currently not leveraging the functionalities provided by many research prototypes (e.g., Palantir [8], Crystal [3]) that are specifically designed to facilitate proactive conflict detection, since they are built as plug-ins to modern IDEs.

We next discuss the top four improvements rated by survey respondents. These are the responses that have a mean value higher than 3.00.

Better Usability: Usability is an important factor that determines whether a toolset supports or hinders the practitioner's workflow. Our survey results indicate that *better usability* (I1) is the most desired improvement of toolsets used for conflict resolution. While usability of a particular

TABLE VI: Survey Participant Merge Toolsets (Top 10)

Tool	# Participants	Description
Git	37	Version Control System
Vim/vi	17	Text Editor
Text Editor (unspecified)	14	Text Editor
Git Diff	11	Diffing Tool
GitHub	11	Website
Eclipse	10	IDE
KDiff3	9	Diff & Merge
Meld	8	Diff & Merge
SourceTree	8	Git/Hg Desktop Client
Sublime Text	7	Text Editor

tool is important, the usability concerns become even more pertinent when they span multiple tools that are similar and must operate in sync with each other. Survey results indicate that participants use an average of 2.5 tools, and as many as 7 tools, to resolve merge conflicts. For instance, in our interview P1 demonstrated how he typically resolved a merge conflict by using four different tools and said:

> *"I have to jump around between tools and copy and paste version numbers...this is why integration matters."*

Switching across multiple tools while resolving a conflict is disruptive and comes at a cost. Psychology studies [44][45] have shown that task switching reduces performance and causes mental fatigue. Gerald Weinberg highlighted that context switching arising from toolset fragmentation is a big problem in engineering teams [46].

Better Exploration of Project History: Practitioners have been known to use historical data to understand code evolution and development processes [37]. Version control and bug tracking systems contain a huge amount of meta-information about the evolution of code and development processes. However, it is not easy to find the right bit of information in these large systems. Currently, there is insufficient support for performing detailed analysis of how a code snippet evolved over time and why. Better ways of exploring the project history (I3) was one of the top requested improvements in our survey. As P1 mentioned in the interview:

> *"Give me a way to explore the history. To drill down, to go back up, you know? To resurface and understand what happened."*

Currently, when performing any complex analysis it is easier to write stand alone scripts to extract the information. During the interview, P1 mentioned that he has written several scripts to locate particular historical commits that relate to a current merge conflict. Similarly, P9 described a tool, `git-diff`, that was developed by their team to add additional difference analysis functionality across branches:

> *"git-diff will just do the diff based on the SHAs... we're adding metadata... It also hooks into GitHub labels to do some more advanced heuristics."*

While writing these scripts allows extraction of relevant data contextualized to the need, it also leads to a proliferation of multiple scripts that are written by individual practitioners and need to be maintained or integrated. This further adds to the problem of context-switching when practitioners must switch between multiple tools, and execute multiple scripts.

We are not the first to recognize the gap in tool support provided for analyzing development history among practitioners [37], [47]–[49]. It appears that practical applications of history exploration are still beyond the reach of practitioners. One of the reasons for this might be the simple set of text editors, and toolsets, that our study participants seem to prefer.

Better Filtering of Less-Relevant Information: Tools that routinely handle large or complex datasets require filtering in order to efficiently locate desired pieces of information. For example, when there are several commits in a pull request and multiple levels of code review at the line level. It is difficult to extract the key issue in the pull request, which can get lost in the sea of low level details. Similarly, if there are multiple commits in a pull request or branch, it is hard to extract the right information. Therefore, tools that provide filtering can better assist practitioners in working with large amounts of metadata associated with the changes. *Better ways of filtering out less relevant information* (I2) was selected as the second most important need; P1 explained:

> *"You want to extract the relevant commits. The ones that actually clash...you want to zoom in on them and understand just enough and don't waste time."*

While improvements in history exploration (I3) will make project metadata more accessible, improvements in filtering for relevant metadata will allow practitioners to focus on the relevant parts of the code impacted by the merge conflict.

Better Graphical Presentation of Information: The usefulness of information is helped or hindered by the way in which it is presented to users. In our survey results, we found that *better graphical presentation of information* (I4) was ranked the fourth highest improvement needed (mean: 3.14).

In our interviews, several practitioners reported experiencing issues with inconsistent terminology, inconsistent visual metaphors (e.g. colors, notifications, etc.), and the organizational layout of different development tools. The cost of context switching in software development is well-known to researchers [50]–[53], and our results indicate that switching between different terminology and information presentation styles can also be a problem. There is a need for tools that share commonality in both terminology and presentation.

Tool Mistrust/Transparency: Most merge tools attempt to resolve conflicts using a variety of algorithms, but revert to manual resolution when these algorithms fail. Several interview participants indicated that they mistrust merge tools when they obscure the steps and rationale for particular results when resolving merge conflicts. The opaque nature of history exploration tools was also found to be a source of practitioners' overall mistrust of their toolsets. P4 commented:

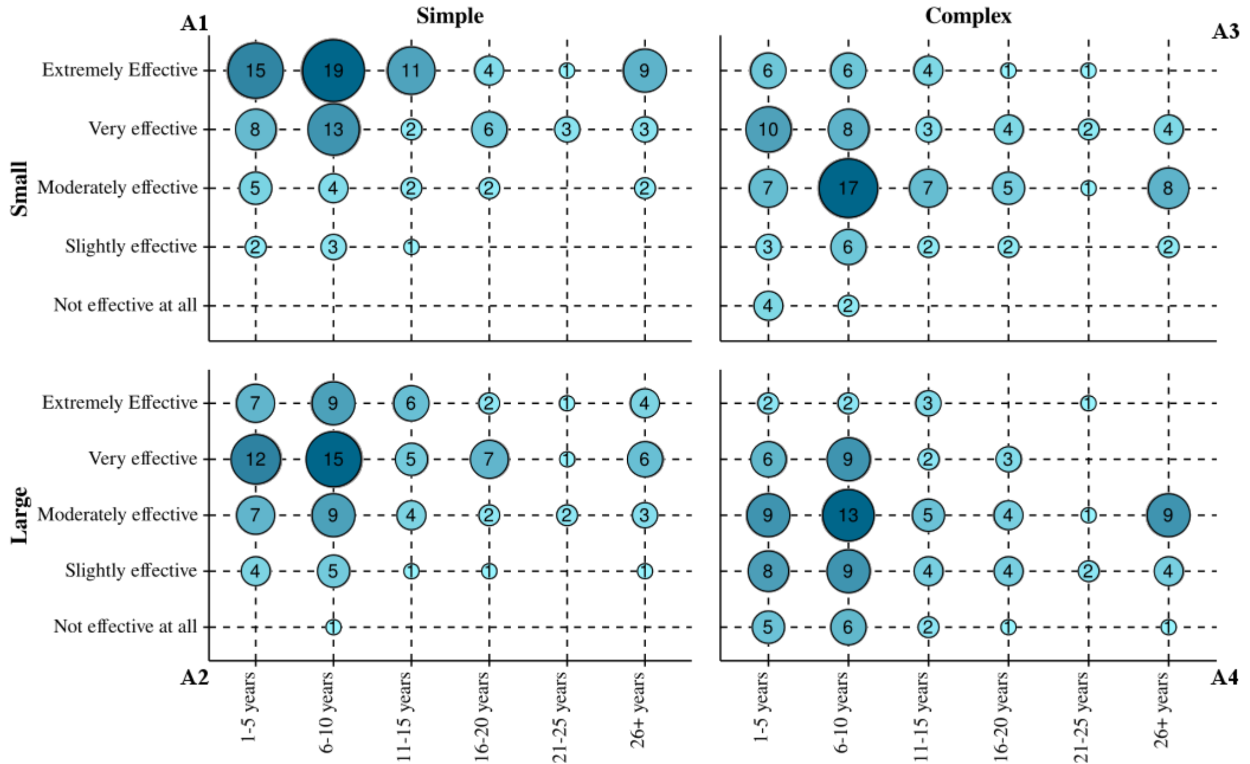

Fig. 1: Effectiveness of practitioners' toolsets in supporting perceived size and complexity of merge conflicts, split on development experience. Bubble values indicate number of survey responses for effectiveness of a particular merge conflict size and complexity, and bubble size indicates the number of responses for comparison purposes.

TABLE VII: Practitioners' Trust in their Merging, History Exploration, and Conflict Resolution Tools[i]

Trust Level	Response Count	Response %
Completely	20	16.52
A lot	50	41.32
A moderate amount	41	33.88
A little	10	8.26
Not at all	0	0.00

[i] Survey respondents answered on a 5-point Likert-type scale indicating trust in their toolset (1 being *Not at all* and 5 being *Completely*).

"I've never trusted the merge tools or diff tools... Sometimes I'll even manually go and do the merge myself rather than use a tool. Just because I've had several times where it's a bad merge, and I broke some code."

Based upon this theme of mistrust, we asked survey participants to rate the degree to which they trust their merging, history exploration, and conflict resolution tools. We received 121 responses to this question, with a mean score of 3.66 placing the most common responses between *a moderate amount* and *a lot* of trust (Table VII). Assuming that responses of *a moderate amount*, *a little*, or *not at all* indicate some degree of mistrust, we find that 42.15% of practitioners experience some gap in toolset trust.

However, the severity of toolset mistrust is not as significant as our interview results suggested. Only 8.26% of practitioners indicated that they trust their toolset *a little* or *not at all* (10 out of 121 responses). As the results of the survey were counter to our interview results, we looked further. We found that: (1) participants reported on the trust levels of the tools that they regularly use, and (2) a large number of participants reported that they had discontinued toolsets when they ran into errors. This indicates that if participants had reported their trust level of these discontinued tools the results would have been lower.

Perceptions of Tool Effectiveness: The perceived size and complexity of merge conflicts affect the way in which practitioners plan, allocate, and enact resolutions. To understand the degree to which these two factors impact practitioners' perceptions about the effectiveness of their toolsets, we asked survey participants to rate their toolset across four different merge conflict archetypes: (A1) *simple, small merge conflicts*, (A2) *simple, large merge conflicts*, (A3) *complex, small merge conflicts*, and (A4) *complex, large merge conflicts*.

Since individual participants have different toolsets, and consider different factors when determining the perceived size and complexity of a merge conflict, we instructed participants to rate their own toolset against these archetypes using their notion of what constitutes a simple vs. complex and small vs. large merge conflict.

Fig. 1 provides a visual illustration of the results of this

survey question. The four plots display the results for each of the archetypes, with archetype (A1) in the top-left plot, (A2) in bottom-left plot, (A3) in the top-right plot, and (A4) in the bottom-right plot. Individual plots are composed of a horizontal axis containing participants' software development experience, which we collect since experience can determine the range of conflicts that they have faced and their perceptions. The vertical axis shows the range of possible responses for the effectiveness of merge toolsets. The size and number within each bubble represent the number of respondents with a particular amount of software development experience that rated their toolset at that specific effectiveness level.

For example, a practitioner with 6-10 years of experience who indicates that her merge toolset is *Extremely Effective* for *small, simple merge conflicts* would be represented in the largest bubble (containing 19) in A1. She would also be represented in the largest bubble (containing 13) in the bottom-right plot (A4) if she indicated that her merge toolset was *Moderately effective* for *large, complex merge conflicts*.

Observing the overall trends when moving between plots, we find that practitioners perceive complexity of the conflict to have a greater impact on the effectiveness of their merge toolsets than the size of merge conflicts. Numerical analysis confirms this when finding that the mean response for archetype (A1) is 4.278 (where 5 is *Extremely Effective* and 1 is *Not effective at all*), (A2) is 3.782, (A3) is 3.347, and (A4) is 2.783. The shift from *small* to *large* merge conflict size (A1 to A2) results in a difference in mean responses of 0.496, whereas the shift from *simple* to *complex* merge conflict complexity results in a difference in mean responses of 0.930.

These results suggest that merge tools are currently equipped to handle increases in the size of merge conflicts, but not as well equipped for increases in complexity. The increasing amount of code being developed in distributed environments means that scaling support in both dimensions is necessary to accommodate practitioners' needs.

IV. Implications

A. For Researchers

Our results inform future research by providing insights into software practitioners' perspectives during merge conflicts.

The top factors that impact the assessment of merge conflict difficulty are primarily focused on program comprehension (F1, F3, F4 from Table III). Program comprehension has been an important research focus, with entire conferences dedicated to it. Previous research has explored tool support and visualizations to help comprehend programs, both small and large. Our results indicate that practitioners still have unmet needs along the following dimensions: (1) comprehending code snippets in isolation, (2) understanding the code context underlying multiple code snippets that are split across multiple files, and commits, and (3) the ability to quickly comprehend the complexity of these code snippets.

Practitioners indicate that their needs during merge conflict resolutions center around the retrieval, organization, and presentation of relevant information (N1, N3, N4 from Table IV). With the variety of meta-information available across different toolsets, and the inconsistent use of terminology, there is a need for standardization and best practices to be developed. Standardization efforts would likely help to alleviate some of the mistrust of merging tools that practitioners have expressed. However, researchers should investigate the margin of errors that are tolerated by practitioners to determine the context in which practitioners discontinue use of tools.

Expertise is seen as both a significant factor that affects the assessment of merge conflict difficulty (F2), and an important need for practitioners to effectively resolve the conflict (N2).

Previous work has focused on recommending developers best suited to perform a collaborative merge based on the previous edits to conflicting files [54] or developers' experience across branches and project history [36]. However, these efforts have resulted in tools that require standalone installation and execution. Our results indicate that practitioners are concerned about toolset fragmentation, and therefore adding an additional tool might be counterproductive to the workflow of most practitioners.

Finally, we find that practitioners need to quickly estimate whether they can fix the conflict, and whether to resolve it now or delay the resolution. This indicates that practitioners need mechanisms to identify the skillsets required to complete the conflict resolution task, by viewing the code fragments. Research should investigate mechanisms to identify required skillsets by using information retrieval or machine learning techniques on the code fragment and past edits.

B. For Practitioners

Practitioners indicate that understanding code, having appropriate information, and dealing with complex codesets are key themes of difficulty when working with merge conflicts (Sections III-A, III-B). Existing tool support can help with some of these issues, but practitioners also need to educate themselves on development processes that prevent and alleviate the severity of merge conflicts. For example, the number of conflicting files and the size of changes are considered important factors. Researchers [55] have previously found that when developers use distributed version control systems that they commit small changes often. Therefore, practitioners should strive to make smaller commits, and commit often. Other agile development processes such as continuous integration, iterative development, and branch merging policies are known to facilitate development in large, distributed teams. However, not all practitioners are actively using such techniques [56], and further work is needed to determine how to enable and ease adoption of these processes and practices.

C. For Tool Builders

Version control systems provide an easy method for storing and retrieving recent development history, but examining older development history at scale and in a usable manner has not completely met practitioners' expectations. Tool builders should work to address this unmet need by leveraging research in search systems for developer-assistance [57] and machine

learning-based code assistance [58] to provide intuitive and expressive tools for history exploration.

Practitioners indicate that current merge toolsets do not scale to handle large, complex merge conflicts (see Section III-C6). To address this concern, tool builders should look at consolidating feature sets that currently span multiple tools in order to provide better usability (I1 from Table V). Tool builders should also add more expressive search and filtering features for both project history and meta-information related to merge conflicts (I2, I3) to ease the frustration of practitioners that must understand the context and evolution of code involved in the conflict.

Finally, we found practitioners having to "guess-timate" the difficulty of the conflict resolution to decide whether to work on it now or delay it, or whether to integrate the changes or simply start over. Prediction tools that identify the complexity of conflicts and difficulty of resolution can help alleviate this.

V. Threats to Validity

As in any empirical study, there are threats to validity with our work. We attempt to remove these threats where possible, and mitigate the effect when removal is not possible.

Construct: Interview questions were open-ended and designed to elicit practitioner opinions about the experiences, difficulties, and perceptions of merge conflicts. We determined particular factors and needs after concluding all interviews, and thus did not bias interview participants to only factors previously mentioned. We created survey questions using factors found through card-based unitization. This methodology allowed us to capture the common themes that practitioners experience when working with merge conflicts, but might have allowed themes specific to particular sub-groups to be unrepresented in our results.

Internal Validity: Central tendency bias [59] occurs when using 5-point Likert-type scales, since participants tend to choose less opinionated answers. We lessen this effect by examining the answers in comparison to each other, as opposed to analysis of absolute mean values. Because we use this method to highlight stronger answers by degree, this also means that we may have missed subtle trends across our data that could have been visible otherwise.

External Validity: Interview results may not generalize to all practitioners due to a small sample size, but we reduce this effect by selecting interview participants from open- and closed-source projects, varying industries, and varying project sizes (see Table I). To expand and confirm our interview results, we survey 162 practitioners to ensure our results match with trends in the larger software development community. We do not report a response rate for our survey, since social media and mailing lists do not allow accurate measurement of the number of individuals that read our recruitment message and but did not choose to participate.

VI. Related Work

Gousios et al. [60] conduct a study in which they ask integrators to describe difficulties in maintaining their projects and code contributions. They showed that integrators have problems with their tools, have trouble with non-atomic changesets, and rank *git knowledge* in the top 30% of their list of biggest challenges. Gousios et al. [61] additionally conducted a study into the challenges of the pull-based model from the perspective of contributors. They found that most challenges relate to code contribution, the tools and model used to contribute, and the social aspects of contributing (specifically highlighting merge conflicts). These works focus on the collaborative processes that go into contributing to open-source projects and operating as integrators within them, whereas we examine the issues inherent to merge conflicts and the tools built to support their resolution.

Guzzi et al. [35] conducted an exploratory investigation and tool evaluation for supporting collaboration in teamwork within the IDE. They found that developers working within a variety of companies were able to quickly and easily resolve merge conflicts, and did this using merge tools. However, they also note that although automatic merging was used, their participants also manually checked each conflict and suggest that this reveals some mistrust of tools. Guzzi et al. further explain that their interviewees avoid merge conflicts by using strict policies and software modularity. Their results complement our findings that toolset mistrust is a major concern, and that standards need to be implemented in order to avoid complex merge conflicts.

Codoban et al. [37] seek to evaluate developer understanding and usage of code history. Our results show that tool support during history exploration factors into the difficulty of a merge conflict a moderate amount (N10). This result independently verifies their findings that practitioners experience tool limitations in usability (I1) and history visualization (I4).

VII. Conclusion

Practitioner perceptions of merge conflicts have an impact on their development process. First, they use perceptions to determine which tactics they will use to resolve the conflict. After choosing how to resolve the conflict, practitioners encounter a new set of needs, both technical and social. Understanding these perceptions and needs is critical to understanding how to design tools which conform to the issues that these practitioners face in collaborative development. We provide actionable implications for researchers, tool builders, and practitioners to harness the results of our study. In future work, we hope to explore whether these factors, needs, and desired toolset improvements can be seamlessly merged into tools or techniques that assist developers' workflows.

Acknowledgments

We thank Iftekhar Ahmed, Amin Alipour, Alex Hoffer, Michael Hilton, Sruti Ragavan, and the anonymous reviewers for their valuable comments on earlier versions of this paper. We also thank all of our interview and survey participants, especially those who helped distribute survey links. This research was partially supported by NSF grants CCF-1439957, CCF-1553741, CCF-1560526, and IIS-1559657.

References

[1] L. Hattori and M. Lanza, "Syde: A tool for collaborative software development," in *International Conference on Software Engineering (ICSE)*, 2010, pp. 235–238.

[2] B. K. Kasi and A. Sarma, "Cassandra: Proactive conflict minimization through optimized task scheduling," in *International Conference on Software Engineering (ICSE)*, 2013, pp. 732–741.

[3] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in *International Symposium and European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 168–178.

[4] C. Bird and T. Zimmermann, "Assessing the value of branches with what-if analysis," in *International Symposium on the Foundations of Software Engineering (FSE)*, 2012, p. 45.

[5] H. C. Estler, M. Nordio, C. A. Furia, and B. Meyer, "Awareness and merge conflicts in distributed software development," in *International Conference on Global Software Engineering (ICGSE)*, 2014, pp. 26–35.

[6] C. R. de Souza, D. Redmiles, and P. Dourish, "Breaking the code, moving between private and public work in collaborative software development," in *International Conference on Supporting Group Work (GROUP)*, 2003, pp. 105–114.

[7] M. Cataldo and J. D. Herbsleb, "Communication networks in geographically distributed software development," in *ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW)*, 2008, pp. 579–588.

[8] A. Sarma, "Palantir: Enhancing configuration management systems with workspace awareness to detect and resolve emerging conflicts," Ph.D. dissertation, University of California, Irvine, 2008.

[9] M. L. Guimarães and A. R. Silva, "Improving early detection of software merge conflicts," in *International Conference on Software Engineering (ICSE)*, 2012, pp. 342–352.

[10] Y. Nishimura and K. Maruyama, "Supporting merge conflict resolution by using fine-grained code change history," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 661–664.

[11] T. Mens, "A state-of-the-art survey on software merging," *IEEE Transactions on Software Engineering (TSE)*, vol. 28, no. 5, pp. 449–462, 2002.

[12] D. Dig, K. Manzoor, R. Johnson, and T. N. Nguyen, "Refactoring-aware configuration management for object-oriented programs," in *International Conference on Software Engineering (ICSE)*, 2007, pp. 427–436.

[13] J. J. Hunt and W. F. Tichy, "Extensible language-aware merging," in *International Conference on Software Maintenance (ICSM)*, 2002, pp. 511–520.

[14] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *International Conference on Software Engineering (ICSE)*, 2010, pp. 175–184.

[15] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *International Symposium on Foundations of Software Engineering (FSE)*, 2006, pp. 23–34.

[16] B. De Alwis and G. Murphy, "Answering conceptual queries with ferret," in *International Conference on Software Engineering (ICSE)*, 2008, pp. 21–30.

[17] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *International Conference on Software Engineering (ICSE)*, 2007, pp. 344–353.

[18] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 285–311.

[19] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *International Conference on Software Engineering (ICSE)*, 2006, pp. 492–501.

[20] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Springer, 2008.

[21] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes?: An exploratory study in industry," in *International Symposium on the Foundations of Software Engineering (FSE)*, 2012, p. 51.

[22] L. A. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, pp. 148–170, 1961.

[23] "Companion site," http://engr.oregonstate.edu/~nelsonni/icsme17.html.

[24] C. B. Seaman, "Qualitative methods," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 35–62.

[25] P. I. Fusch and L. R. Ness, "Are we there yet? data saturation in qualitative research," *The Qualitative Report*, vol. 20, no. 9, p. 1408, 2015.

[26] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement," *Sociological Methods & Research*, vol. 42, no. 3, pp. 294–320, 2013.

[27] D. Spencer, *Card Sorting: Designing Usable Categories*. Rosenfeld Media, 2009.

[28] W. Hudson, "Card sorting," in *The Encyclopedia of Human-Computer Interaction*. Interaction Design Foundation, 2013.

[29] D. R. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman, "Revisiting methodological issues in transcript analysis: Negotiated coding and reliability," *The Internet and Higher Education*, vol. 9, no. 1, pp. 1–8, 2006.

[30] J. Ritchie, J. Lewis, C. M. Nicholls, R. Ormston *et al.*, *Qualitative Research Practice: A Guide for Social Science Students and Researchers*. Sage, 2013.

[31] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering (TSE)*, vol. 26, no. 8, pp. 797–814, 2000.

[32] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering (TSE)*, no. 4, pp. 308–320, 1976.

[33] D. Garmus and D. Herron, *Function Point Analysis: Measurement Practices for Successful Software Projects*. Addison-Wesley Longman Publishing Co., 2001.

[34] C. R. Symons, "Function point analysis: Difficulties and improvements," *IEEE Transactions on Software Engineering (TSE)*, vol. 14, no. 1, pp. 2–11, 1988.

[35] A. Guzzi, A. Bacchelli, Y. Riche, and A. van Deursen, "Supporting developers' coordination in the IDE," in *Computer Supported Cooperative Work & Social Computing (CSCW)*, 2015, pp. 518–532.

[36] C. Costa, J. Figueiredo, L. Murta, and A. Sarma, "TIPMerge: Recommending experts for integrating changes across branches," in *International Symposium on Foundations of Software Engineering (FSE)*, 2016, pp. 523–534.

[37] M. Codoban, S. S. Ragavan, D. Dig, and B. Bailey, "Software history under the lens: A study on why and how developers examine it," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 1–10.

[38] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *International Conference on Software Engineering (ICSE)*, 2008, pp. 241–250.

[39] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *ACM Symposium on Document Engineering (DocEng)*, 2002, pp. 26–33.

[40] K. Yamauchi, J. Yang, K. Hotta, Y. Higo, and S. Kusumoto, "Clustering commits for understanding the intents of implementation," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 406–410.

[41] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classification of large changes into maintenance categories," in *International Conference on Program Comprehension (ICPC)*, 2009, pp. 30–39.

[42] L. F. Cortés-Coy, M. L. Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2014, pp. 275–284.

[43] L. P. Hattori and M. Lanza, "On the nature of commits," in *International Workshop on Automated engineeRing of Autonomous and run-tiMe evolvIng Systems (ARAMIS), ASE Workshops*, 2008, pp. 63–71.

[44] N. Meiran, "Modeling cognitive control in task-switching," *Psychological Research*, vol. 63, no. 3, pp. 234–249, 2000.

[45] D. Gopher, L. Armony, and Y. Greenshpan, "Switching tasks and attention policies," *Journal of Experimental Psychology: General*, vol. 129, no. 3, p. 308, 2000.

[46] G. M. Weinberg, *Quality Software Management, Vol. 1: Systems Thinking*. Dorset House Publishing Co., 1992.

[47] X. Sun, B. Li, Y. Li, and Y. Chen, *What Information in Software Historical Repositories Do We Need to Support Software Maintenance Tasks? An Approach Based on Topic Model*. Springer, 2015, pp. 27–37.

[48] J. Guo, M. Rahimi, J. Cleland-Huang, A. Rasin, J. H. Hayes, and M. Vierhauser, "Cold-start software analytics," in *International Conference on Mining Software Repositories (MSR)*, 2016, pp. 142–153.

[49] Y. Yan, M. Menarini, and W. Griswold, "Mining software contracts for software evolution," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 471–475.

[50] M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2004, pp. 175–182.

[51] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Workshop on Experimental Computer Science (ExpCS), FCRC Workshop*, 2007, p. 2.

[52] A. Blackwell and M. Burnett, "Applying attention investment to end-user programming," in *Symposia on Human-Centric Computing Languages and Environments (HCC)*, 2002, pp. 28–30.

[53] G. Convertino, J. Chen, B. Yost, Y. S. Ryu, and C. North, "Exploring context switching and cognition in dual-view coordinated visualizations," in *International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV)*, 2003, pp. 55–62.

[54] J. R. da Silva, E. Clua, L. Murta, and A. Sarma, "Niche vs. breadth: Calculating expertise over time through a fine-grained analysis," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2015, pp. 409–418.

[55] C. Brindescu, M. Codoban, S. Shmarkatiuk, and D. Dig, "How do centralized and distributed version control systems impact software changes?" in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 322–333.

[56] S. Phillips, J. Sillito, and R. Walker, "Branching and merging: An investigation into current version control practices," in *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2011, pp. 9–15.

[57] T. Nabi, K. M. Sweeney, S. Lichlyter, D. Piorkowski, C. Scaffidi, M. Burnett, and S. D. Fleming, "Putting Information Foraging Theory to work: Community-based design patterns for programming tools," in *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 129–133.

[58] A. W. Bradley and G. C. Murphy, "Supporting software history exploration," in *Working Conference on Mining Software Repositories (MSR)*, 2011, pp. 193–202.

[59] J. P. Guilford, *Psychometric Methods*. McGraw-Hill, 1954.

[60] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 358–368.

[61] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, 2016, pp. 285–296.