A Five-Year Survey of Literature in Software Engineering and Repository Mining Research

Samuel D. Schwartz
sam@cs.uoregon.edu, March 20, 2023

Abstract

This literature review surveys the extant research from 2018 through October 2022 in three key conferences related to software engineering and repository mining: Mining Software Repositories (MSR), the International Conference on Software Engineering (ICSE), and the International Conference on Software Maintenance and Evolution (ICSME). The review taxonifies hundreds of research papers, and provides summary and analysis of noteworthy papers related to code writing and refactoring, code comprehension, smells and code quality, aspects of software development related to an entire software project, and human dynamics within that project. The review also explores cross cutting themes among these topics, such as bots, machine learning, and the influence of different environments – such as free open source projects vs start ups vs established industry – on the development process. Underlying this work is an eye to research applicable to the advanced scientific computing space, which, in the United States, is often performed at national laboratories.

Contents

1 Summary of the Literature Review

This survey of literature is based on my interest in software engineering tools to support the advancement of advanced scientific computing (ASC). Building on my prior experience in the ASC domain and master's thesis work, I also have particular interest in how machine learning can be leveraged to support the development of these tools.

I surveyed over 1,500 papers, primarily from three venues: the MSR, ICSE, and ICSME conferences for the five years from 2018 to October 2022. These papers were filtered and taxonified into 18 different topics in an approach detailed in Section 2.

I further synthesized papers from five topics out of the 18: code writing and refactoring (Section 3), code comprehension (Section 4), smells and quality (Section 5), aspects of development applicable to an entire software project (Section 6), and human dynamics within a project or development team (Section 7).

In addition to the 18 topics identified above, six cross-cutting themes emerged. I focus and elaborate on three of these cross-cutting themes: machine learning (Section 8), bots (Section 9), and specific venues; e.g., open source projects, start ups, national labs, etc. (Section 10).

I summarize key takeaways from this work in the conclusion (Section 11). Namely, there is very little published research on software engineering tools specifically tailored to the work of developers of advanced scientific computing projects, such as those commonly encountered in national laboratory environments. Consequently, this is a fruitful area for future research.

2 Methodological Approach

The survey of the literature was done from a *tabula rasa* angle, and done in several steps. This section describes each of these steps which were taken to collect and taxonify the extant literature.

2.1 First Step: Amass a large collection of works

In pursuit of my aim to survey the literature broadly, I collected the titles and abstracts of all works published in the three conferences and associated journals of Mining Software Repositories (MSR), International Conference on Software Engineering (ICSE), and International Conference on Software Maintenance and Evolution (ICSME) from 2018 to October 2022. Over 1,500 works resulted from this initial query.

2.2 Second Step: Filter the collection of works

I then filtered these 1,500+ works against several porous analytical sieves. One was the necessity of the work being a research paper or well documented tutorial of a tool. Panel discussions or other presentations without an accompanying manuscript were excluded. A second porous sieve used to filter the corpora of works was a manuscript's relevance to my interests. Specifically, the broad criterion for relevance used as I read these hundreds of abstracts was the following:

Related somehow (even loosely) to tools, dynamics, or other analytical framework utilizing graph theory or machine learning techniques to analyze software developer communication, behavior, and git activity juxtaposed with development productivity or code and documentation quality metrics at any point in the software engineering/development process.

This filtering resulted in a selection of just over 400 papers.

2.3 Third Step: Sort works into topics

I manually sorted these 400+ filtered papers into 18 topics, some of which I elaborate on more in this work. These topics are:

- Learning or school/education related papers
- New developer onboarding
- New features and requirements
- Code writing and refactoring (Section 3)
- Code comprehension, and documentation consumption (Section 4)
- Documentation production
- Testing
- Commits, merges, and conflicts
- Pull requests and code reviews
- Smells and quality (Section 5)
- Maintainability, technical debt, production performance
- Bugs, faults, and vulnerabilities
- Traces, links, and context
- Deprecation
- Whole project / entire repository aspects and status (Section 6)
- Human and team dynamics (Section 7)
- Machine learning foundations (Section 8)
- Papers on research in software engineering and repository mining, such as literature reviews

These broad 18 topics are more formally defined in Appendix A.

2.4 Fourth Step: Sort papers in each topic into four distinct categories

Within each of the 18 topics, I further partitioned each paper into the general categories of "Tools," "Psychology or Social Science," "Analysis or Broader Studies," and "Curated dataset." I define these categories as:

- A "*Tools*" paper is any work which introduces a software artifact or other device which their target audience can use as part of their software development toolkit.
- A "*Psychology or Social Science*" paper is one where human beings were heavily used in the research methodology. (Excluding those tool papers where humans were simply used to validate the utility of the tool.)
- A "*Curated dataset*" paper is a work where the authors provide data to facilitate future research by others. In rare instances this may also include a tool, but a tool intended to be primarily used by others doing research in software engineering or repository mining – not typical developers.
- An "*Analysis or Broader Studies*" paper is essentially any other work not already categorized by the above slots. However, these papers are often characterized by phrases like "empirical study," "investigation," "evaluation of ..." "relating X with Y," "does Z implicate W?," and so forth.

Papers were filtered into one or more of these 18 topics $\times$ 4 categories. This formed a matrix of papers, which are all listed in Appendix A.

2.5 Fifth Step: Identify cross-cutting themes

In sorting papers into this initial placement of topics and categories, there were additional themes that emerged which cut across the 18 topics $\times$ 4 categories grid described above. These cross-cutting themes were identified and relevant papers annotated in a second round of assessment. The cross-cutting themes identified are:

- Bots
- Diversity, Equity, and Inclusion (DEI); Specifically identifying the following sub-themes:
  -DEI relating to Race/Ethnicity/National Origin,
  -DEI relating to Sex/Gender,
  -General DEI / Other DEI
- Graph Theory
- Machine Learning
- Privacy or Security
- Venue specific works; Specifically identifying the following sub-themes: Academia, Industry, Open Source, Start Up, National Laboratory / ASC

2.6 Results

The final filtering for all papers can be found in Appendix A.

Five topics were selected for further synthesis: code writing and refactoring (Section 3), code comprehension (Section 4), smells and quality (Section 5), aspects of development applicable to an entire software project (Section 6), and human dynamics within a project or development team (Section 7).

Additionally, three cross-cutting themes receive additional treatment: machine learning (Section 8), bots (Section 9), and specific venues; e.g., open source projects, start ups, national labs, etc. (Section 10).

The next few sections of this review focus on selected papers from these five topics and three cross-cutting themes.

3 Topic: Code writing and refactoring

The topic of "code writing and refactoring," for our purposes, is the act of a developer actually constructing, copying, small-level editing, and wholesale revising of source code in some language(s) – which typically happens in a text editor or integrated development environment (IDE).

3.1 Tools

New tools created in the last five years designed to support developers in the process of writing and rewriting code generally fall into one of three categories:

1. *Automatic code writing.*
   This category includes tools designed to predict what the developer will do next in the (new) code writing process, with a machine learning engine usually running the tool.
2. *Existing code analysis.*
   This category includes tools designed to analyze and help a developer refactor and revise existing code, such as ensuring naming consistency in APIs.
3. *Specialized IDEs or IDE enhancements.*
   This category includes tools – usually GUI dashboards or plug-ins – in an IDE, or entire IDEs themselves, to enhance or enable some aspect of code writing.

These three categories are not mutually exclusive. Rather, the tools available in the code writing and refactoring space often fall into two or more of these categories. The rest of this subsection provides illuminating examples of tools in each of these three categories.

1. Automatic code writing tools in the modern era almost uniformly utilize some sort of machine learning algorithm under the hood. While the exact and idiosyncratic methodologies vary by tool for training and validation, the trend is that machine learning algorithms have gotten more and more accurate at predicting what a developer is going to type next, with [Bulmer et al., 2018] reporting 64% accuracy in their 2018 code prediction tool, [T. Nguyen and Nguyen, 2019a] reporting 70% accuracy in their 2019 tool, and a plateau thereafter, with [F. Wen and Bavota, 2021a] reporting a mere increase of 1% and an overall accuracy of 71% in their tool in 2021.

What has increased in machine learning tools since 2019 is a wider breadth of supported languages, or specializations within those languages. [F. Wen and Bavota, 2021b] supports Android projects, for example. And, as an example of machine learning code prediction tools specializing for niches within a language, [A. M. Mir and Gousios, 2022] leverages a new standard in Python which allows for static typing. The tool predicts what type a variable is (string, int, float, etc.) allowing a developer to quickly add static types to existing code, which makes the performance of this modified code much more efficient when executed with newer Python interpreters. Such work might be relevant to ASC codes, which are often decades old. Tools to "auto upgrade" the code itself to new language standards might find a lot of utility in older and densely technical codes which contain many routine "no brainer" changes to be made.

Other developments in the machine-learning-for-code-writing space focus on improving the machine learning-based tools themselves. In [A. Svyatkovskiy and Allamanis, 2021], the authors note that deploying code completion machine learning tools are often extremely resource intensive – to the point their use can bog down a developer's machine. Svyatkovskiy et al. present a "suggested options tool" which takes up just 6MB of RAM, and the authors claim 90% accuracy in their top-5 suggestions (which is roughly on par with other works claiming about 70% accuracy for their top-1 suggestion). This feat of equivalent accuracy to existing systems but with less resource consumption by a code prediction oracle was done by relying less on machine learning training and more on static analysis, and then cleverly blending the results of the two techniques.

Blending static analysis with machine learning prediction, however, presents trade offs. The larger the code base of a project (or set of reference projects), the more difficult it is to integrate static analysis techniques (e.g., clone identification and consequent suggestion) as part of a code prediction tool. On the other hand, limiting the amount of static analysis done by a tool requires more leaning on a pre-trained neural network, which can be large and bulky to include in an IDE. This is the challenge that [F. Silavong and Otter, 2022] grappled with, and presented a code hashing solution as a possible compromise between the machine learning and static analysis tensions, with computation times for code search suggestions on the order of hundreds of times faster than Facebook's 2019 code recommendation tool, Aroma [Luan et al., 2019].

2. Existing code analysis tools can be analogized to spelling and grammatical checking tools of the word processing world – they help make existing code better, rather than directly suggest its initial creation. In particular, these tools are largely focused on (i) improving code readability by analyzing class and method names and suggesting changes; (ii) analyzing overarching patterns and anti-patterns at a project-wide level and providing suggestions or warnings; and (iii) analyzing how code has changed over time to predict upcoming needed changes.

Examples of improving code readability by analyzing class and method names and suggesting changes include [Liu et al., 2019], which presents a tool with a binary output: does a method named X actually do X? That is, if I have method called `func square(x){return x*x;}`, the tool would return "True." But if my method was `func square(x){return x;}`, the tool would return, "False." Perhaps it's unsurprising, given the difficulty of the problem, that the tool only had a 25% accuracy – which was much better than the 1% accuracy of its comparator. Still, it was able to identify 66 real-world method-name inconsistencies in the wild, indicating the utility of the tool as a possible "let's double check this" warning to a developer. Comparing the results of this tool with nominal documentation may prevent code smells in musty software.

A different flavor of tool, but in the same vein as [Liu et al., 2019], was presented in [S. Nguyen and Nguyen, 2020]. This tool suggested method names based on a method's implementation. For example, if the code was `{return x*x;}` the tool might suggest the name `square`. The authors subsequently analyzed a large number of existing method names in open source projects with their tool and identified methods which the authors thought their tool's suggested name was better. The new name was accepted in pull requests 74% of the time.

Examples of analyzing overarching patterns and anti-patterns at a project-wide level and providing suggestions or warnings include work like FOCUS [P. T. Nguyen and Penta, 2019], which uses context aware mining to identify suggested API calls at the point of code writing, or at refactoring. It also includes tools like [A. Barbez and Guhneuc, 2019], which identified anti-patterns (anti-patterns defined as, "poor solutions to recurring design problems") from a mix of structural and historical data from a project or code file's git history to identify well known anti-patterns, such as the "God Class" anti-pattern – where everything and the kitchen sink is thrown into one massive, convoluted class.

Lastly, examples of analyzing how code has changed over time to predict upcoming needed changes include [N. Tsantalis and Dig, 2018]. This paper utilized techniques that analyzed abstract syntax tree change over time in mining to determine candidate code ready for refactoring without relying on excessive user-defined thresholds or non-generalizable default settings. This work was followed up by [M. Tufano and Poshyvanyk, 2019], which was the first machine-learning based paper which used time-series data to predict refactoring activities, particularly bug fixes in legacy code.

3. Specialized IDEs or IDE enhancement tools which don't have a machine learning or refactoring component are the least common type of tool encountered in the surveyed literature, but include works like [B. Hempel and Chugh, 2018], which presented an IDE with clickable widgets embedded in the GUI that allow developers to graphically drag-and-drop syntax. Essentially, an "adult" version of Scratch. Other exemplary tools include CodeRibbon [Klein and Henley, 2021], which provides an alternative visual layout for workspace management and is available as a plug-in for popular IDEs like Visual Studio Code or Atom. None of these tools are immediately applicable to the ASC paradigm in any differentiating way, but it's important to be aware of them.

3.2 Psychology / Social Science

There is a rich array of work which studies how humans act when writing code.

Two studies are noteworthy for their use of fMRI imaging of the brain. One study, [Huang et al., 2019], compared data structure manipulations of lists, arrays and trees with other spatial rotation exercises on 76 participants. They found that they were all distinct but related neural tasks, and that the more difficult computer science problems required a higher cognitive load (i.e., measurable brain activity) – eventually surpassing the cognitive load of the spatial reasoning problems participants grappled with. Another study, [R. Krueger and Leach, 2020], compared code writing with prose writing, and found that the two activities are extremely dissimilar at the neural level. Code writing primarily activated the right hemisphere of the brain, while prose writing activated the left. Still, neurological studies of the software engineering process are in their infancy, and many questions remain. How does the writing of technical documentation of complex algorithms compare to pure code writing or pure prose writing, for example?

One limitation of fMRI scans is the amount of time one has to measure activity. Work such as [Bellman et al., 2018] recorded clicks and keyboard activity from IDEs long programming sessions, specifically focusing on contextualizing developer activity during build failures and debugger usage. They found that developers spend much of their time debugging code, and using breakpoints do help in that process. This high-level finding is unsurprising, but the details are helpful in creating models to measure developer productivity. As a crude example, one expects less lines of code to be produced when a developer is working on "harder" debugging issues than more straightforward logic. Recording developer behavior in an IDE environment also forms the basis for [K. Damevski and Pollock, 2018], which found certain coding behavior had probabilistic distributions analogous to natural language production and processing. This indicates that, while prose writing and code writing are distinct events at a biological neurological level in humans, machine learning techniques designed for natural language processing and prediction can be applied to code processing and prediction in analogous ways.

Pairing developer behavior with eye-tracking software has also been done. One study, [N. J. Abid and Maletic, 2019], found significant differences in eye-tracking behavior between novice and expert developers, with experts spending less time on a function call and more time on the implementation for complex functions. They also found that both novices and experts revisit control flow terms (e.g., if-statements) in repeated but short bursts. With that said, other work, such as [V. Zyrianov and Maletic, 2020], notes that there are technological shortcomings to most eye tracking + IDE software combinations. Namely, most eye tracking software before 2020 isn't that accurate when focused on typical-font sized lines of code which is being scrolled or moved – especially when the developer switches among different open tabs in a typical editor. They presented a tool called Deja Vu to fix these problems. Consequently, results from earlier work on code development and eye tracking should be taken with a grain of salt.

Of additional interest is the study of developer emotion during the code writing process. Some works in 2018 continue that year's trend of heightened community interest in sentiment analysis, finding that off the shelf tools for sentiment analysis usually didn't work in the code writing and comprehension arena (e.g., developers expressing happiness when a library worked or frustration when encountering a bug) [F. Calefato and Novielli, 2018, B. Lin and Oliveto, 2018]. Other work, such as [D. Girardi and Lanubile, 2020], found that wearing a smartwatch-like wristband that measured electrodermal and heart activity reliably measured developer's emotional activity. Moreover, the authors found that, perhaps unsurprisingly, positive emotions were correlated with being "in flow" or in a productive coding state of mind, while negative emotions were correlated with frustration and roadblocks.

While the above studies have focused on developers as individuals, other studies have focused on code writing from a team or group identity perspective. For example, in [Amlekar et al., 2018] the authors examine whether software engineers, and other code writers like academic researchers, students, hobby programmers, etc. utilize code writing tools like auto-complete in different ways. They found inconclusive results, and suggested more work needs to be done to understand how different practitioners code. This inconclusive finding has particular relevance to the ASC situation, given the unique blends of professional software developer and professional scientists which tend to contribute to ASC code bases, and suggests more research needs to be done in this area.

Another example of work analyzing team-based code writing contributions include [M. D. Stefano and Lucia, 2020], which purports that socio-technical incongruence leads to worse code. That is, poor coordination amongst developers in a team leads to increased technical debt in the project, thus requiring more refactoring down the road. The authors in [M. D. Stefano and Lucia, 2020] provide a possible framework to coordinate refactoring ideas. This work was timely, given the investigation of developer perceptions and decision making processes explored in [Alomar, 2019], which examined when a developer labeled a code change as a "refactor" in a commit message vs when a change wasn't given that label.

Finally, we come full circle back to the utility of tools used to automatically analyze and suggest names discussed in the previous subsection in the study by [R. Alsuhaibani and Maletic, 2021] which surveyed over 1,100 developers on standards for source code method names and analyzed the responses based on things like years of experience and programming language knowledge. The authors found high degrees of conformity on the importance of adhering to the established standards, and provided a foundation for automated method name assessment which could then be used either by automated tools or during human-led code review.

3.3 Analysis / Broader Studies

Relative to the number of papers about tools and social science in the code writing space, there are fewer papers in the broader analysis category.

One theme which does show up are empirical studies on some aspect of machine learning or data mining. For example, [M. Ciniselli and Bavota, 2021] examined the use of BERT models for code completion tools in an empirical study, finding and affirming that BERT models alone, without additional algorithmic cleverness, are good at predicting code at about the 60% accuracy level. This finding is congruent with an empirical study done on GitHub Copilot, branded as an "AI pair programmer," which found great variety in code suggestion accuracy based on language, but no more than 60% accuracy [Nguyen and Nadi, 2022]. The authors in [Y. Huang and Zimmermann, 2021] present work similar to that done by [A. Svyatkovskiy and Allamanis, 2021] in proposing a ranking system idea which merges both machine learning results and other prediction algorithms, such as abstract syntax tree mining based tools, to improve accuracy. And, with respect to abstract syntax tree mining itself, [P. T. Nguyen and Penta, 2019] presents a novel graph-theoretic paradigm to extract program dependencies and change patterns at scale.

Finally, an analysis was done in [Rahman, 2018] which explored how code writing activities correlated with code comprehension activities, finding that the relationship is nuanced and complex, but that when code is difficult to comprehend edits are made at a much slower rate.

3.4 Curated Datasets

There were no papers which released a curated dataset for code writing activities as the primary contribution of their work. That said, MSR has code mining challenges which are typically recorded and released to the community. This data can form the basis for some exploratory studies before authors collect their own custom data.

4 Topic: Code comprehension

Developers don't just spend time writing code and documentation. Rather, undertaking the task of code comprehension – and seeking out the resources to gain understanding – is a major component of a developer's activity. In [X. Xia and Li, 2018], the authors found that approximately 58% of a developer's time was spent on code comprehension activities. Code comprehension includes reading existing documentation, performing web searches, reading Q&A websites (e.g., Stack Overflow), and communicating with other developers.

4.1 Tools

In many ways, tools in the code comprehension space are very similar to the kinds of tools one will find in the code writing space. This is understandable – once a developer understands a code fragment, a developer will often choose to implement it if the fragment has utility. Consequently, many tools in this space are oriented around porting code from places where code comprehension happens (such as Stack Overflow, API/Library documentation sites, similar project's repositories, etc.) and into the code artifact the developer is working on.

Tools in the code comprehension space can be best thought of as lying in a plane with two axes. The first axis is usefulness (from very useful to almost useless), the second is breadth (from niche to all-encompassing). Most tools seem to lie within a zone around a line which starts at (niche, very useful) and terminates at (all encompassing, almost useless).

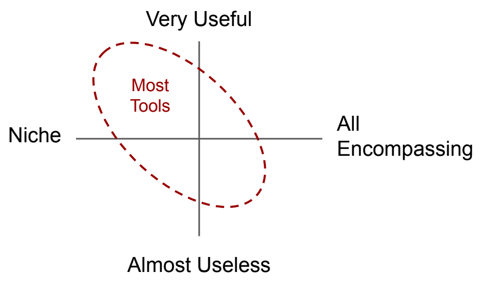


Figure 1: Tools in the code comprehension space

Perhaps the best example of this pattern comes from tools related to APIs and API comprehension. At one end of the spectrum, in the (niche, very useful) zone are tools like those presented in [H. Phan and Nguyen, 2018], which simply finds the fully qualified name of various API calls from snippets posted on sites like StackOverflow. Slightly further down the line is [Li et al., 2018], which is a tool that uses natural language processing to identify various caveats and exceptions to common API usages. Yet further down the line is FOCUS [P. T. Nguyen and Penta, 2019], which provides API recommendations based on mining and analysis of API usage in other open source systems deemed similar to the current project. Next in line is FaCoY [Kim et al., 2018], a code to code search engine. And, at the far end of the spectrum, is [Eberhart and McMillan, 2021], which essentially seeks to eventually replace a developer colleague as a one-stop-shop for code understanding question-and-answer dialog. This work, while quite comprehensive, and trailblazing in the right direction, is not particularly functional yet.

4.2 Psychology / Social Science

Stack Overflow and similar Q&A sites dominate the code comprehension space where humans interact with each other. Consequently, several studies have been conducted to critically evaluate the quality of the developers providing the answers compared to Stack Overflow's metrics, evaluating the speed at which other users provide answers to questions, and analyzing fact vs opinion-based responses.

With respect to developer quality, [S. Wang and Hassan, 2021] succinctly answers that area in their self-explanatory paper, *"Is reputation on Stack Overflow always a good indicator for users' expertise? No!"*. Nevertheless, developers are still asking and answering questions on these platforms. There has been particular attention paid to the speed at which answers are posted, with [S. Wang and Hassan, 2018] first doing a systemic analysis on some 46 factors related to the question itself, the accepted answer, the user posing the question, and the user answering with an accepted answer. Assessed on four Stack Overflow sites (Stack Overflow, Mathematics, Ask Ubuntu, and Superuser), the authors found that factors related to the user answering the question most strongly impacted the statistical likelihood of an answer being accepted. The key takeaway: a quickly provided answer of mediocre quality submitted by a frequently contributing user is much more likely to be accepted than a high quality answer provided by an expert user that infrequently contributes.

This dynamic is reaffirmed and explained in [Y. Lu and Li, 2020], "*Haste Makes Waste: An Empirical Study of Fast Answers in Stack Overflow*" which concluded that quickly provided answers aren't always the best ones – measured in part by the amount of follow up required in the comments – despite being the popular answers. The authors attribute this to the gamification style of Stack Overflow to encourage participation and interaction. Consequently, aspects of the gamification may need to be tweaked to maximize answer quality. On the other hand, Stack Overflow is also a business which has an income stream from advertisement revenue. Therefore, a business decision to maximize human participation on the site at the expense of good-but-not-optimal Q&A-gamification practices may be in play.

Other works in this space analyze the reliability of code fragments provided in Stack Overflow answers. In [T. Zhang and Kim, 2018] the authors find that more than 30% of all Stack Overflow posts contain API usage violations which can compromise the integrity of software which incorporates them. And in [S. Mondal and Roy, 2021], the authors survey "rollback edits" – when a user posts an answer, then edits the answer, then edits or rolls back the edit. They find that these kinds of back-and-forth editing dynamics lead to inconsistencies and more than 80% of professional developers assess that they lead to detrimental post quality. Moreover, a developer's understanding of whether answer content on Stack Overflow is applicable to them often requires additional context than what was initially provided, with [A. Galappaththi and Treude, 2022] finding that almost half of the Stack Overflow threads in their empirical study eventually included clarifying context that was not in the initial question.

With the relative unreliability of code snippets posted online in mind, particularly those posted in new threads anchored by novel questions, perhaps it's not surprising that some developers try to find work-arounds from API calls posted on sites like Stack Overflow altogether, which is what was explored in [Lamothe and Shang, 2020]. This work, complemented by similar findings in [M. A. Al Alamin and Iqbal, 2021] which studied developer discussions in software development challenges, found that the most common types of questions which related to low post quality revolved around "customization" and "dynamic event handling." This suggests that APIs which allow developers to do "customization" and "dynamic event handling" easily are more likely to be adopted.

Stack Overflow, while a major component of human interaction with code comprehensibility, isn't the only option for Q&A / code comprehension sites. Other sources of help with comprehension exist and, like with any other aspect of human interaction on the web, search engines loom large. In [Rahman et al., 2018], the authors created a machine learning classifier to automatically identify which real-world queries from several hundred developers were code-based Google searches vs non-code related queries. The authors found that code related searches required more effort (search term modification, multiple result clicks, etc.) than non-code search. Further work in [Hora, 2021] found that most search queries by developers that aren't copy-pasting code or error messages are typically short (three words or less), start with a limited set of key words – often the name of the framework, language, or platform (e.g., Python, Android, etc.), and omit functional words. They found that Stack Overflow results dominate, but YouTube is also a relevant source nowadays. This may indicate that different developers prefer comprehension via different mediums, or that different problem types lend themselves to be addressed via different mediums. More research in this space is needed.

4.3 Analysis / Broader Studies

In an effort to create better tools to support code understanding, several foundational machine learning papers have been presented. These are precursors to more advanced machine learning models which today can predict code. Rather, in an attempt to solve an easier problem of simple classification, there were two noteworthy papers in 2020 which focused on sentiment analysis in the software engineering domain [E. Biswas and Vijay-Shanker, 2020, T. Zhang and Jiang, 2020], particularly using BERT. More modern work has expanded into using these BERT-based models as pre-trained bases for general source code understanding models which can then understand and write code. This work is bleeding edge and not quite ready for day-to-day usage as an engine for any sort of broadly useful tool. These prior works do, however, form the building blocks for tools which, if successful, will likely have broad utility. For example, if one can train a neural network model to do sentiment analysis well on text that includes code snippets or technical jargon, that base model can become a core of a new model via transfer learning to do novel tasks.

Meanwhile, humans still rule the day, and so evaluating techniques to increase readability – like in [J. Johnson and Sharif, 2019], which evaluated different code writing rules to increase comprehensibility with some 275 developers – as well as a system literature review on comprehensibility in [D. Oliveira and Castor, 2020] are key. Particularly in the [D. Oliveira and Castor, 2020] paper, which found that assessing code readability is a highly subjective exercise.

Two other studies in the Stack Overflow arena were focused on specific types of application development. For example, [G. L. Scoccia and Autili, 2021] did topic mining of Stack Overflow posts related to desktop web applications, and found that (1) build and deployment processes were some of the most common issues developers faced; (2) reuse of existing libraries in the desktop app development space is cumbersome; and (3) debugging of native API problems is tough; all of which tracks with [M. A. Al Alamin and Iqbal, 2021]'s finding that API customization and dynamic event handling are common roadblocks in development. Similarly, [Abdellatif et al., 2020] did a topic analysis mining of Stack Overflow posts related to chatbot development, finding that most posts revolve around chatbot model training and integration. Given that the recurring themes of API customization, dynamic event handling, and application deployment + integration categorize the most vexing issues of most development, further work in this area may include an analysis of projects where these aren't the most common issues, and seeing if some categorization of these projects can be made. Moreover, understanding how ASC projects relate to these kinds of common issues in other open source or industry projects remains unstudied.

4.4 Curated Datasets

Three datasets in this area have been published since 2018. One dataset, reflective of the papers published in 2018 relative to ML and sentiment analysis in this space, is a collection of 4,800 Stack Overflow questions, answers, and comments which were hand labeled with emotions [Novielli et al., 2018]. Another two datasets, which provided similar manual labeling of sentiments, was provided in 2018 by [B. Lin and Bavota, 2018].

Additionally, [B. Kou and Zhang, 2022] provides a dataset of over 2,200 popular Stack Overflow posts with manually provided summaries, which can be used as training data in a ML context to summarize discussion.

Lastly, [Baltes et al., 2018] presents SOTorrent which is a tool that facilitates the ease of mining of Stack Overflow data, as well as incorporates various similarity metrics and data which can facilitate time series analysis useful to researchers doing mining on the website.

5 Topic: Smells and quality

Code smells are symptoms of poor implementation choices applied during software evolution [Pecorelli et al., 2020]. While smells were once intuitively identified by experienced developers, various definitions have since been developed and standardized. We also include in this topic the closely related notion of measuring formal smells by also including code quality, and the various metrics to assess code quality, as part of our discussion.

5.1 Tools

There are two flavors of papers in the intersection of "tools" and "smells + code quality metrics": (1) Where code quality (and code quality metrics) are used to further a goal within a tool; or (2) tools used to evaluate and visualize the quality of the code itself, or to identify particular smells.

As an example of a tool used to further a goal, in [Nayrolles and Hamou-Lhadj, 2018] the authors use code quality metrics (with clone detection) in their tool, CLEVER, which does just-in-time fault prevention in large industrial projects. In this same vein, [Trockman et al., 2018] reevaluates a study ([Scalabrino et al., 2017]) which assesses the understandability of written code from a human perspective – in much the same way an algorithm might classify a book as grade school level understandability or college level understandability – using differing kinds of code metrics in their tool. Or, [A. Utture and Palsberg, 2022] which uses static analysis, metric calculation, and graph theory to provide inputs to a machine learning model which is subsequently used to identify and remove null pointer bugs.

An example of a tool which evaluates code is [Sharma and Kessentini, 2021a], which presented a platform called QScored that identifies and visualizes various quality metrics of an overall repository or project. The motivation being to have better access to metrics for project comparison than simply the number of stars or issues that a GitHub based repository might have. Another tool in this vein of code evaluation is the machine learning model developed by [Pecorelli et al., 2020], which used both code metrics and developer perception to rank the smellyness of design issues.

5.2 Psychology / Social Science

"The mind is a powerful place." So claims [M. Wyrich and Wagner, 2021] in the title of their seminal paper on how code comprehensibility metrics intersect with code understanding. In their work, the authors undertook a double-blind study which evaluated the extent of which a displayed code comprehensibility metric impacted a developer's subjective belief that the code was, in fact, comprehensible. The authors found that, regardless of the actual code comprehensibility, developer belief of code comprehensibility was incredibly influenced by a displayed score.

This finding is in congruence with earlier work done in [J. Pantiuchina and Bavota, 2018]. In this work, the authors empirically evaluate code quality metrics' claim to identify smells as valid. They do so by evaluating whether developer submitted commits which specifically claim to improve one of four attributes in the commit message (cohesion, coupling, readability, complexity) are truly improved by various metrics. It turns out that, despite code being improved from the developers perspective, most code quality metrics fail to capture improvement.

5.3 Analysis / Broader Studies

While code quality metrics may not be as reliable as a human software developer's expert assessment in a general context (despite popular belief in the authority of formally defined metrics), they do seem to have utility in the narrow area of testing and test smells. In [D. Spadini and Bacchelli, 2018a] the authors investigate the relationship between test smells – that is, when the testing code itself is smelly – and code quality of the software being tested. Specifically, the authors analyzed 221 releases of 10 software systems and compared six types of smells with various kinds of software quality metrics. They found that smelly tests were correlated with lower quality, more defect-prone software.

Further work in [G. Grano and Gall, 2020] investigated a similar question as in [M. Wyrich and Wagner, 2021] (namely, are code quality metrics congruent with developer perceptions?) and found that, in the narrow area of unit test code quality, metrics are a necessary but not sufficient condition for identifying problematic code. Consequently, the current state of the literature suggests that code quality metrics have only limited utility compared with expert developer assessment.

Further muddying the water is an empirical study by [D. Kavaler and Filkov, 2019] which found that not only do the code quality metrics themselves matter, but so does the idiosyncratic quality assurance tools which are used to automatically assess code as part of CI, as well as the order in the pipeline in which they are introduced.

Still, code smells and quality metrics shouldn't be written off entirely. In [P. Gnoyke and Krger, 2021], the authors identified how architecture smells can evolve over time. In particular, the authors note that if quality assurance is postponed or abandoned for a system then smells and their correlated issue-proneness can dramatically increase.

Consequentially, the state of the literature appears to point to a paradox: smells and code quality metrics being close to meaningless in terms of production quality – that is, will the code break or not – at an individual code fragment level (with some caveats for test code), yet vitally important for the overall health of the project.

5.4 Curated Datasets

More work is needed to tease out the relationship between developer perception, metric utility at the code fragment level (that is, code about the length of a single method implementation), and overall impact on the issue-proneness of the project as a whole. To assist in this future work [Sharma and Kessentini, 2021a] submits, via the QScored platform discussed in their parallel work [Sharma and Kessentini, 2021b], a large dataset of quality metrics and code scores which can subsequently be mined for future insights.

6 Topic: Whole project aspects

While other topics in this survey narrow down on specific aspects of the software development process, this topic covers impacts to an entire software repository or project as a whole, or empirical analysis across multiple repositories.

6.1 Tools

There are relatively few tools in this space which don't have a primary home in another topic. Tools which are applicable to this topic typically involve mining an entire project's repository (or multiple repositories) in order to address a legal, security, or privacy issue. For example, [R. Feng and Zhang, 2022] presents a tool for automated detection of passwords in public repositories, which they deployed on GitHub and found that over 60 thousand public repositories had passwords embedded in publicly available code. Another paper, [X. Xu and Liu, 2021], presented a tool used to find software reuse in public repositories which violates licensing agreements, with a 97% accuracy rate.

These types of tools are dependent on widespread mining ability. As the number of repositories grows, and as the size of each individual repository also contains more artifacts, the ability to mine and crawl becomes a bigger challenge. The authors of [F. Heseding and Dllner, 2022] confront this challenge with a command line tool and library called *pyrepositoryminer* built for multi-threaded repository mining, achieving 15x speedup against other existing off-the-shelf, generic web-based mining tools typically used for repository mining. A different tool, LAGOON [Dey and Woods, 2022], does similar things as pyrepositoryminer, but focuses more on the exploration and visualization of sociotechnical data of open source software projects from a variety of sources (code repositories, mailing lists, project websites, etc.) What seems to be missing in this space is a clear standard for the dissemination and sharing of technosocio repository data, as each tool uses its own idiosyncratic data formats for saving and sharing insights gleaned from mining.

6.2 Psychology / Social Science

Four themes are emergent in this area:

1. Developer mindset or attitude to project-wide characteristics or issues.
2. Analysis on the motivations, behaviors, and characteristics of contributors to projects, particularly open source projects.
3. Analysis on the motivations, behaviors, and characteristics of financial backers of projects, particularly open source projects.
4. Legal policies and licensing issues, and their effects on users, developers, and projects.

1. Developer mindset papers include works like [Hadar et al., 2018], which explored how engineers approached privacy. The authors found that most developers used the vocabulary of data security to inform privacy concerns, and as such limits the perspective of top privacy issues to mostly external threats. Other work analyzing developer mindset includes [T. Sedano and Praire, 2019], which explored how developers and other stakeholders (like project managers) conceptualized, created, and dealt with backlog of technical to-dos – things like feature requests, bug fixes, and known technical debt servicing. The authors found that existing theoretical frameworks for backlog in other business domains didn't really apply to software backlog, but by having team members simply thoughtfully reflect on the items in a backlog as a group, collective sense-making allowed for more effective prioritization of tasks leading to greater productivity. Still other work like [S. Biswas and Rajan, 2022] presented an empirical evaluation of data science pipelines and how data scientists conceptualize the work to be done in a data science project. They present two related conceptual/theoretical models – data science in the large, and data science in the small – for how data science projects are conceptualized in practice by practitioners.

2. Analysis on the motivations, behaviors, and characteristics of contributors to projects includes work like [H. Fang and Vasilescu, 2022], which studied the efficacy of social media – particularly Twitter – in advertising open source projects on GitHub. Among other things, the authors found that tweets did impact repos by increasing the number of people starring a GitHub project, and a modest link of new contributors to these projects. Other work focuses on analyzing the geographic history and diversity of contributors to public code bases, such as [Rossi and Zacchiroli, 2022]. They found that, over the last 50 years, public code contributions have been dominated by North American and European developers, with contributions from other geographic areas like South America, Central Asia, and Africa slowly picking up starting around 1995 and increasing roughly linearly since that time. Today, non-North American and non-European contributors provide roughly 30% of all open source project commits. Notably, China provides very few contributions to the open source project ecosystem relative to its population.

Beyond where contributors are from and how they are incentivized to join projects, work has been done on collaboration and co-commit patterns of developers. One large scale study on some 200 thousand GitHub repositories was [Cohen and Consens, 2018], which found that the most active developers have tighter, more insular, and less collaborative networks than developers as a whole. This work was highly technical, and metrics were based on various graph-theoretic metrics like node connectivity (where a node is a developer, and an edge is placed between two developers if they contributed to the same repository). Results are, frankly, difficult to intuitively interpret.

This led to work like [Lyulina and Jahanshahi, 2021], which presented a tool to visualize projects, developers, and their contributions via interactive graph. A challenge they faced is the sheer size of such networks, which required some sort of pruning for the interactive visualization to be digestible for a human being. Future work in this space includes thoughtful analysis for such filtering and how to motivate the utility of undertaking exploratory analysis via visualized interaction graphs.

3. Analysis on the motivations, behaviors, and characteristics of financial backers of projects, particularly open source projects.

Money makes the world go round, and that is also increasingly the case for open source projects. While software project financing in industry, government, and academia are usually self-explanatory as to where the money is coming from and why (self-explanatory at a high level, exact financing details can be quite complex), the same is not true of open source projects. The authors of [C. Overney and Vasilescu, 2020] explored 25,885 GitHub projects that asked for donations, out of a total of 77,934,441 repositories (0.04%). They found that popular projects (as measured by number of stars), mature projects, and projects with recent activity all were more likely to ask for donations. The authors also concluded that most donated funds are advertised to go to engineering efforts, but there is no systemic evidence that funding makes much of an impact on project activity levels.

Other work by [N. Shimada and Matsumoto, 2022] explores GitHub Sponsors, a program launched in 2019 which allows donors to fund specific developers that contribute to open source software projects. The authors found that developers who are sponsored are more active than non sponsored developers seeking sponsorship, sponsored developers are typically top contributors to projects before getting a sponsorship, that roughly two-thirds of sponsors are developers themselves, and that sponsors and sponsorees are usually part of the same tight-knit networking clusters.

4. Legal policies and licensing issues, and their effects on users, developers, and projects.

Underlying software development are issues related to licensing and intellectual property. Developers are generally not attorneys, yet modifications to software licenses can have significant legal impacts. This phenomena was studied in [C. Vendome and Poshyvanyk, 2018], which empirically examined "licensing bugs," finding that everything from laws and their interpretation, to the legal re-usability of seemingly open source code with conflicting licenses (or even no provided license), to jurisdictional issues all present complex and novel problems which both developers and attorneys have conflicting views on. Understanding just how prevalent these kinds of legal issues are was examined in [Golubev et al., 2020], which empirically studied Java projects on GitHub, searched the repositories for code clones, and analyzed the original licenses of the source of the copied code and the embedding project. They found that up to 30% of projects involved code borrowing and about 9.4% contained copied code which could violate the licensing usage agreement of its source.

Beyond the intellectual property implications of code use and reuse in software projects, other empirical work on the non-discrimination policies of software artifacts is gaining traction. For example, work like [M. Tushev and Mahmoud, 2021] found that most non-discrimination policies are buried deep within "Terms of Service" documents (as opposed to a separate document, like many privacy polices), if there is a written policy at all. The policies that do exist are usually very brief and boilerplate, and almost always have no written enforcement mechanism. Given real-world allegations and court findings of discriminatory behavior of apps or app users related to well known companies like Uber and AirBnB, not to mention concern about algorithmic fairness – particularly in machine learning / artificial intelligence algorithms – closer attention to written non-discrimination policies (or lack thereof) appearers poised to be a dynamic and evolving field in the next few years.

6.3 Analysis / Broader Studies

There are four themes about software projects and their repositories that reoccur in the literature:

1. Taxonification of projects and their artifacts.
2. Qualification of projects and their artifacts.
3. Quantification of projects and their artifacts.
4. Analysis of how projects change over time.

1. "What is software?" This is the question asked by students in introductory programming classes everywhere which [Pfeiffer, 2020] sought to empirically answer by mining 23,715 GitHub repositories. They organized their findings into 19 different categories and assert that, far from software simply consisting of source code and perhaps an executable, software also contains scripts, configuration files, images, databases, documentation, licenses, and so forth. Some of their research questions include "Does a characteristic distribution of frequencies of artifact categories exist?" The author's answer is no, but it is unlikely (less than 1%) that a repository contains more documentation artifacts than data artifacts (like images), and more data artifacts than source code artifacts. This work was assessed on 23,715 repositories covering a wide range of projects, and therefore future work in this area includes a tightened focus on examining a narrow set of related projects. For example, asking the same questions about software artifacts, but restricting the evaluation set to ASC projects.

2. Quality of projects in research

Moving beyond individual artifacts are papers assessing attributes around the quality of repositories as a whole. Of particular note is [Hasabnis, 2022], which was a hackathon that resulted in GitRank, a tool to measure the quality of repositories. The motivating issue was an assertion that poor-quality repositories should be excluded from use in machine learning training data, and as such a tool needed to be developed to rank and compare such repositories when working at a large scale. This assertion that poor-quality repositories are problematic is a well founded one, as the paper "Is 'Better Data' Better than 'Better Data Miners'?" [Agrawal and Menzies, 2018] found the answer to be "Yes." The authors found that ML-based tools which do things like defect prediction performed better with much higher quality training data than simply improving or modifying the underlying ML+data mining model.

3. Quantification of projects in research

The quality of the data isn't the only factor in research. Quantity matters, too, particularly when investigating human productivity in relation to GitHub data. The authors of "*Big Data = Big Insights? Operationalizing Brooks' Law in a Massive GitHub Data Set*" [C. Gote and Scholtes, 2022] concluded that conflicting results in prior empirical work about Brooks' law in open source software projects was primarily driven by poor data collection and aggregation pitfalls that occur when doing massive data analysis. (As a reminder, Brooks' law asserts that adding developers to a project counterintuitively causes overall progress to slow down, not accelerate.)

Gote et al. found that, "Studies of collaborative software projects found evidence for a strong [...] effect for different team sizes, programming languages, and development phases. Other studies, however, found a positive linear or even super-linear relationship between the size of a team and the productivity of its members." and produced a long list of citations of conflicting work. They found that differing methodologies when doing statistical analysis accounted for many of the perceived differences. Methodologies which, in their view, were sloppy. For example, neglecting to do proper stratified sampling.

The takeaway is that big data requires big caution when analyzing human interaction data in a repository mining context, and thus smaller and more curated datasets can often give clearer insights than larger, noisier ones. This echoes earlier work on the threats of aggregating software repository data in [M. P. Robillard and McIntosh, 2018], which identified and described common threats to big data analysis in a software mining context.

4. Time series analysis of projects

High quality, low quality; big data or small; there's an interest in evaluating how projects change over time. Themes in this area include papers which present new metrics, like [Benkoczi et al., 2018], which presented a framework for identifying how changes in commit patterns impact dependencies and metrics like cohesion and coupling in various code modules over time. Other work, like [M. Tushev and Mahmoud, 2019] finds that linguistic features (for example, variable names and patterns) in a multi-developer code base changes quite a bit in the first era of a project's existence before mostly standardizing, usually around the creation of documentation, with slow but notable changes thereafter, with patterns which match human language changes in written speech in long-lived projects. And yet other work like [A. Ait and Cabot, 2022] analyzes the survival rate of GitHub projects, finding that open source projects are prone to death, with less than 50% surviving more than five years, and most projects are characterized by high bursts of activity followed by long periods of low/no active development or maintenance.

6.4 Curated Datasets

While there are several papers discussing data collection in repository mining, there are relatively few works which present curated datasets as the primary contribution. Two are relatively niche: [Zacchiroli, 2022], which presents a large dataset of open source licenses and variants, and [V. Tawosi and Sarro, 2022] which provides a dataset of open source projects that specifically and actively use a particular agile methodology in development.

One dataset of note is provided by [N. Riquet and Vanderose, 2022], which contains git data from an anonymous private company with over 100 repositories and some 120+ developers over the course of more than a decade. Given that software engineering and data mining research on data from private companies is such a rarity, this dataset could help form a useful benchmark against open source assumptions. Unfortunately, the company required the data to be heavily anonymized (including the name of the company itself) as a condition of the data's release. Consequently, there are limitations to the usefulness of this data when used as a comparator benchmark.

7 Topic: Human and team dynamics

This section focuses on the human element of software development. Namely, how do humans relate to the engineering process and each other while engaging in development work.

7.1 Tools

Tools relating to human dynamics fall into one of the following three buckets:

1. Characterizing developer behavior or attributes from human-sourced information pools.
2. Characterizing software from human-sourced information pools.
3. Assisting the developer in the software engineering process.

As is a recurring theme throughout this literature survey, there are no tools developed specifically for ASC, but we do highlight open source-based and other noteworthy tools which may have some utility in the ASC field or spark ideas.

1. Developer behavior tools focus on helping a team lead or manager understand what's going on with individual contributors. For example, in [T. Dey and Mockus, 2021], the authors try to solve the problem of matching open problems / tasks which need finishing in an open source software setting with relevant developer expertise and skill set. The authors accomplish this by using machine learning embeddings of code. Namely, the authors take machine learning embeddings of open source project code which needs an additional maintainer, and compares this vector embedding with code written by various software developers. The idea is that developers who have worked on similar projects as the ones needing maintainers will have code embedding vectors which are likewise similar.

Another tool which may help managers and other team leads is EmoD [K. P. Neupane and Wang, 2019] which captures communication records and identifies the emotions therein. Paired with a visualization, the tool can alert team leads to possibly unhealthy emotions in online communication as developers collaborate with each other.

2. Software behavior tools pull information from human dynamics to inform the development process. For example, [L. Shi and Wang, 2020] is a machine learning tool which mined an extremely large set of developer chats to predict new feature requests from the text, creating an automated repository of ideas to possibly develop and implement. Other tools are more helpful in audit and compliance. For example, the natural language processing machine learning tool developed by [Hu et al., 2021] identifies 26 predefined types of undesirable software behavior from user reviews of mobile applications that violate GooglePlay's marketplace rules. This tool can help human auditors monitor for compliance for things like making sure games or apps which contain nudity are properly rated for adults. While advanced scientific computing tools are unlikely to contain nudity, the broader framework to automatically detect undesirable app behavior may be helpful.

3. Assistants to a developer are also in this space. While there is much more to be said in §9 on bots, tools include assistants like Robin [L. D. Silva and Moisakis, 2020] which is a "voice controlled virtual teammate." Specifically, Robin is an Amazon Alexa or OKGoogle-like assistant for git and GitHub. Robin is able to, via oral command, do trivial git-related tasks, such as closing issues or pull requests. This tool was trained based on extensive data collection from real-world software teams asking each other favors in offices. As for its practical utility, well, it seems to still be in the early stages.

7.2 Psychology / Social Science

There is a lot of fun work to read relating to the psychology and social science of software development. There's even work done on cannabis usage among programmers! [M. Endres and Weimer, 2022] found that 35% of their sample of programmers had engaged in software development while high.

More generally, work can be sub-categorized into one of the following bins:

1. Leadership in software development.
2. Motivation for staying on a project.
3. How programmers organize and manage themselves to get work done.
4. How humans perceive and measure productivity, proficiency, effort, and efficiency.
5. How information and results are communicated with actors outside the team.

1. Leadership

Leadership research in software development is somewhat conflicted. While there is a lot of research done in business and management on skills and attributes of leaders, mapping those traits aren't always clear-cut in the software development world and vary depending on the organizational context. For example, in [E. Kalliamvakou and German, 2018] the authors conducted a mixed methods study to find what made a great manager of software engineers, deploying a survey to nearly 3,700 developers at Microsoft. They found that deep technical expertise wasn't important (although a minimum level of competence was needed). Rather, the people skills of providing support, coaching, direction, and motivation is what was most valued.

This contrasts somewhat with the work done by [E. Dias and Pinto, 2021] about the leadership attributes of lead maintainers of major open source projects (e.g., the Linux Kernel). Dias et al. found that deep technical expertise, along with quality communication, were the key attributes of excellent leaders in long-term open source projects.

Yet later work in [Gren and Ralph, 2022] explores leadership in agile software development, which often rejects a hierarchical organization structure with a clear leader in favor of a group of small teams working closely together (e.g., Spotify's internal software development organizational structure). Gren et al. finds leadership is not a property of individual contributors but rather a property of individual teams, within which responsibility and leadership is dynamically shared by developers. The most effective and leadership-like teams demonstrated a strong sense of belonging for their members, an ability to navigate competing interests with other teams in pursuit of the larger organizational mission, and strong communication skills by all team members.

The key theme of software development leadership, it seems, is the ability to motivate and communicate, with technical expertise taking a very important but secondary role.

2. Motivation

Motivation is another key ingredient to successful software development. While we did not find motivation research specifically focused on paid software development roles, there are several studies done on motivation in the open source community, including [H. S. Qiu and Vasilescu, 2019, A. Barcomb and Fitzgerald, 2019, Gerosa et al., 2021, Y. Huang and Zimmermann, 2021]. While the methodologies and foci of these studies vary, all point to an underlying importance of two factors in developer motivation for contributing to these communities: (1) feeling like they belong to the community to which they contribute, which often includes recognition; (2) the belief that their contribution impacts the world for social good in some way. While these motivations are for the open source communities, applying them to a national laboratory context by having lab leaders reimagine their work as contributing to the social good may help with motivation.

3. Team organization

Team organization and human-centric studies on the development process is a well studied area. It covers things like pair-programming work in [Zieris and Prechelt, 2020] which found pairs with experiences with different libraries are more effective than pairs with homogeneous knowledge. Other interesting work looked at whether programmers work at night or on the weekend [M. Claes and Adams, 2018], finding paid programmers tend to work standard office hours but there is wide variability among developers generally, with open source software contributors tending to work at heterogeneous times. Yet more work is area specific, with [Kim et al., 2018, N. Nahar and Kstner, 2022] focusing on the specific issues of data science and machine learning collaboration challenge. They find that communication challenges among stakeholders and mis-matched expectations are the most common roadblocks in machine learning team success.

Other research identified characteristics of work productivity, such as [F. Sarker and Filkov, 2019]. It found that as the frequency of developer commits increased, so did the negative sentimentality of their comments, which the authors suggest is due to increased stress as developer work-rate increases. More generally, [S. Datta and Sarkar, 2021] theorized that projects tend to start out with developers relying on each other more, but as projects mature, developers become more independent. This leads to a situation where developers on a team in successful projects usually don't need much formal structure in the beginning of a project – they are all in frequent contact with each other and the problems are nebulous enough that formal process is useless – but as the project matures, formal contribution guidelines are helpful.

This brings us to [O. Elazhary and Zaidman, 2019], which studied whether nominal contribution guidelines matched the real-world process for over 50 GitHub projects. They found that in nearly 70% of these projects with explicit contribution guidelines the actual contribution processes materially differed.

That was in 2019. In 2020, COVID struck. [de Souza Santos and Ralph, 2022] found that the transition from in-person to remote teams was disruptive, suggesting that managers needed to do more to encourage team engagement and provide greater support for those with child care responsibilities. Other work by [C. Miller and Zimmermann, 2021] suggests that 74% of developers who moved to a work from home environment missed their colleagues, with over 50% saying that communication had become more difficult. How these statistics look now, in 2023, remains to be seen. In any case, leaders sought to motivate their teams. A common team motivator is gamification of a goal (e.g., whoever has the longest daily commit streak "wins"), but work by [L. Moldon and Wachs, 2021] suggests that doing so often leads to lower levels of productivity after the game ends. Understanding the best practices of how to lead and motivate remote or hybrid teams remains open work.

4. Perception of productivity

A variety of studies have examined what makes a developer appear to be productive, usually through surveys. This includes work such as [X. Xia and Lo, 2019,Rodriguez et al., 2018], which found a diverse constellation of factors – time commitment, quality of code produced, etc. – impacted the perception of a developer's productivity. But more work is increasingly focused on what makes software development teams appear to be productive. Work such as [Ruvimova et al., 2022] suggests that highly productive teams in a paid setting are typically made up of highly productive individuals with fluid roles. That is, an ability for the individual members of the team to quickly and interchangeably self select tasks which need to be done. However, more research needs to be done to determine how this finding applies across multiple domains, and whether it replicates in the advanced scientific computing setting.

5. Communication outside the team

Development teams typically communicate with two types of outside actors: other developers and users. In the latter case, work by [S. Hassan and Hassan, 2018] suggests that developers are more likely to respond to longer or more negative user reviews of their software than positive reviews: squeaky wheel gets the grease. But in the former case, of developers on a team talking about their work with other developers, papers like [Fang et al., 2020] have found that the nominal role of a developer on a team impacts how people view the messages. For example, developers perceived to be the team lead or repository owner have different types of engagement on Twitter when broadcasting their work than other members of a team. This broad trend of "the lead" being the key communicator in evangelizing work is reflected in other research like [Aniche et al., 2018, M. Papoutsoglou and Kapitsaki, 2021]. This has relevance in the national laboratory context if one goal is to advertise the open source work done in the labs to other user groups: if you want to share your work, it's important to get it out there on multiple channels, and have the person sharing (1) perceived to be a technical person, not a communications/HR person and (2) be perceived as the leader of the team. There is a tension here, because we just saw in perceptions of team productivity and effective organization that many of the most productive teams are those with members that contribute and lead in fluid ways, as opposed to a strict hierarchy.

7.3 Analysis / Broader Studies

Broader studies in the human dynamics space have focused largely on bias [Zhang and Harman, 2021, Chattopadhyay et al., 2020], typically along gender lines [Hilderbrand et al., 2020, Lee and Carver, 2019, N. Imtiaz and Murphy-Hill, 2019] or age [W. Kopec and Casati, 2018]. The recurring takeaways seem to be that women are not as included in software development teams as their male counterparts, and that special attention should be paid to individuals belonging to a minority demographic among any sufficiently large group setting (e.g., older adults in mostly young groups).

Other work is a bit of a grab bag of topics. One study analyzed toxicity in comments made by developers in places like GitHub and compared it with toxic commentary online generally, finding that toxic communication between developers is often characterized as more demanding and arrogant but with fewer open insults than toxic conversation found elsewhere online. This presents challenges in any natural language processing work done to characterize emotions in developer discourse.

But perhaps most amusing of all is a study done by Google called, "Do Developers Discover New Tools on The Toilet?" in which internal teams advertised new tools to their colleagues by placing physical advertisements/newspapers in workplace restrooms. The technique was effective, and perhaps an option for in-person devops teams seeking to evangelize their tools in large workplaces like national laboratories.

7.4 Curated Datasets

There was only one paper which presented a curated dataset designed for use by other researchers, DISCO [K. M. Subash and Baysal, 2022], which is a dataset of discord chat conversations about software engineering from four programming language communities (e.g., the official python discord server). While the main contribution of their work is the cleaned dataset, the tools they used to create the dataset are interesting and helpful in their own right for mining other discord servers.

8 Topic and cross cutting theme: Machine Learning

....

A brief sidebar on ChatGPT and opinion on its societal implications.

As of this writing, ChatGPT has taken the world by a storm since its release in November 2022. It is the subject of countless news stories and has entered the public conscious. It is also increasingly being used to generate code.

ChatGPT – having not been published as an academic work before our cutoff date of October 2022, or in the three venues of ICSE, ICSME, or MSR – is outside the context of this exam. Nevertheless, a short and high level overview is appropriate for this survey, given the nature of the model.

The architecture of the ChatGPT model isn't particularly complex once one is able to understand the fundamental building blocks of modern deep learning. Namely, transformers, attention, word tokenization, embeddings, transfer learning, and sequence-to-sequence generative models. The underlying generative pre-trained transformer (GPT) model also utilizes reward modeling and reinforcement learning paradigms as part of its training. All of these concepts are well documented and are readily accessible to a machine learning practitioner in formats outside of academic literature, like blog posts and (admittedly advanced) online tutorials.

What separates ChatGPT is not any major conceptual novelty or clever algorithm not already in the literature. Rather, the major driver underlying ChatGPT is the sheer scale of the number of parameters in the model – in the billions – and the terrabytes of training data used. In many ways, the primary limitation of another entity replicating ChatGPT isn't in the software, but rather in the hardware infrastructure needed to train the massive model. OpenAI, the organization behind ChatGPT, converted from a non-profit to a "capped profit" business in 2019 for the purpose of gaining enough investment capital to actually buy the hardware needed to train models.

In some ways, this is a concerning development. Most modern technological innovations related to information, to date, have resulted in long-term benefit for the common person and cheapening ownership costs over time. For example, printers are now ubiquitous, relatively cheap, and there are several companies which manufacture printers in a competitive marketplace environment. Printers are democratized. But large language models, like ChatGPT, will require hardware investment in the billions of dollars and highly specialized software development expertise for meaningful competitors to emerge. In practice, only a handful of players – tech titans and governments – will have the resources to duplicate the feat.

It's also important to note that the key underlying model driving ChatGPT, a model called GPT3, has not been publicly released. While GPT2 has been released as an open source product for anyone to download, GPT3 – built on top of GPT2 and described in various releases by OpenAI – has not been made available for download. OpenAI has made GPT3 available for public consumption by developers only through API calls (which can easily be made motinizable as a subscription service). At a high level, ChatGPT is essentially a light wrapper for GPT3, which OpenAI has made more user friendly to non-software developers by adding a website interface. It is this website interface which the general public thinks of as ChatGPT.

By treating the language model as a software-as-a-service, OpenAI (and other tech players) have set the precedent that consumers of machine learning models are not owners of the model. Depending on your politics, this may be concerning. One can stream music from a service like Spotify or Apple Music, and many find that to be very convenient. But, at the end of the day, one can also purchase a song and own it. Many software products are the same way – there is usually an ownership alternative to software-as-a-service product. At least at the enterprise/business-to-business level.

But, given the massive capital required to create a machine learning model of this scale, it is likely that only tech titans and governments will have the resources to duplicate ChatGPT. This is concerning, as the barrier to entry for potential competitors will grow increasingly higher in the near to mid term. This will result in a situation where only a few select entities will have control over some of the most powerful and generalized machine learning models, or the realistic ability to improve on them, thus exacerbating the problem by increasing the barrier to entry for new actors.

A potential remedy to this problem, assuming there is societal agreement that artificial intelligence products should be democratized, is to treat these artificial intelligence products in much the same way we do prescription drugs. After a period of time of exclusive use for monetization, require companies with monetized machine learning algorithms to release the source code and data for the underlying trained model as a "generic" machine learning algorithm for anyone else to use.

In the meanwhile, I take heart that ChatGPT hasn't fully taken over the world of machine learning as it relates to software development. It means that the remainder of this area exam chapter remains relevant to any entity creating machine learning based tools for software development without a billion dollars of capital to spend.

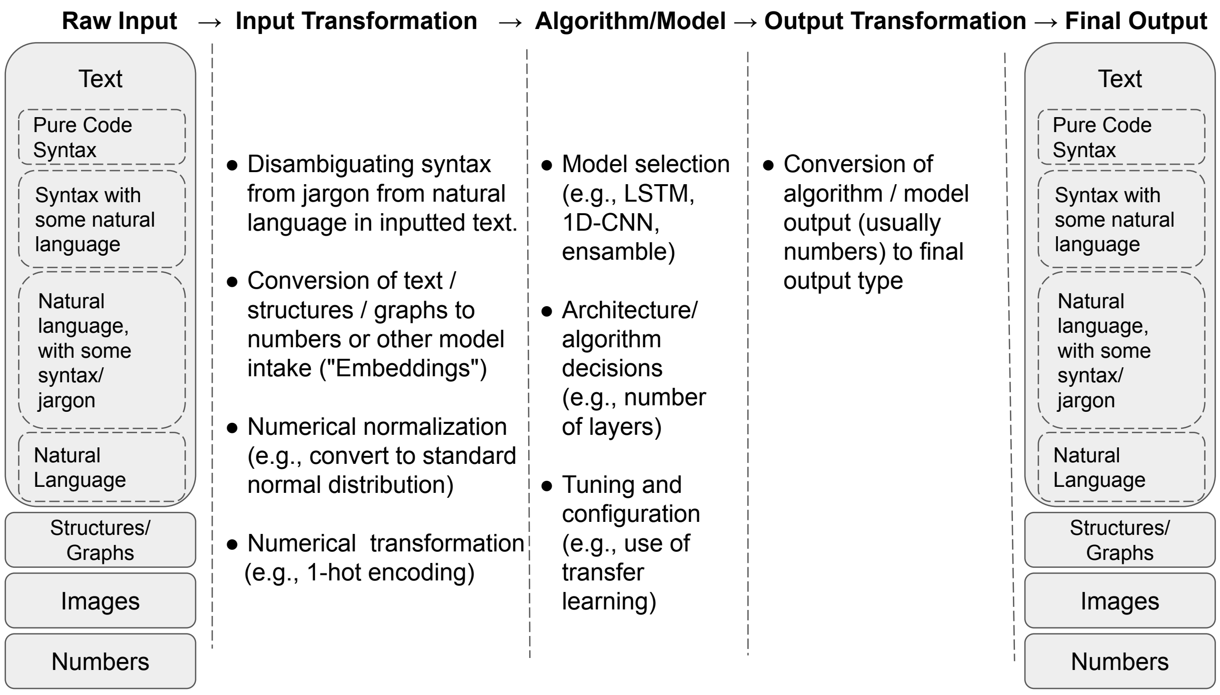
. . . .

Machine learning (ML), in the context of software engineering and repository mining, is used primarily as the underlying engine for tools that aids practitioners in some aspect of software development. Almost all ML work in this space is grounded in a supervised learning context.

Within this larger context there are two flavors of machine learning papers which are published in ICSE, ICSME, and MSR. One flavor of paper reports on machine learning models as a component of a tool. These machine learning-based tools are a cross cutting theme, featuring in topics ranging from bug detection to code suggestions.

The other flavor of paper focuses on some aspect of the machine learning development process itself. That is, the authors of ML development papers focus on an aspect of the training or model design as their core contribution to the community, often in the pursuit of a larger goal to make a useful machine learning-based tool. For example, the task of automatically disambiguating natural language from technical jargon from code before ingestion by a language-based ML model, which tends to improve results.

This section will primarily focus on papers in this input $\rightarrow$ input transformation $\rightarrow$ model / algorithm $\rightarrow$ output transformation $\rightarrow$ final output pipeline, highlighting examples of various tools along the way. This pipeline, and the subcategories of papers in each pipeline, is visualized in Figure 2.



Raw Input → Input Transformation → Algorithm/Model → Output Transformation → Final Output
Text
Pure Code Syntax
Syntax with some natural language
Natural language, with some syntax/ jargon
Natural Language
Structures/ Graphs
Images
Numbers
• Disambiguating syntax from jargon from natural language in inputted text.
• Conversion of text / structures / graphs to numbers or other model intake ("Embeddings")
• Numerical normalization (e.g., convert to standard normal distribution)
• Numerical transformation (e.g., 1-hot encoding)
• Model selection (e.g., LSTM, 1D-CNN, ensamble)
• Architecture/ algorithm decisions (e.g., number of layers)
• Tuning and configuration (e.g., use of transfer learning)
• Conversion of algorithm / model output (usually numbers) to final output type
Text
Pure Code Syntax
Syntax with some natural language
Natural language, with some syntax/ jargon
Natural Language
Structures/ Graphs
Images
Numbers

Figure 2: Components of a typical pipeline in an ML-based tool.

8.1 Raw Inputs and Final Outputs

It is important to note the different types of inputs and outputs typical in software engineering machine learning models. Namely, models can ingest and output raw numbers, images, structured data – such as graphs, and text. Within the category of text is a spectrum of possible input, ranging from pure code, to a mix of natural language and code / technical jargon (e.g., documentation), to non-code natural language.

Examples of software development tools which use images include work like [N. Cooper and Poshyvanyk, 2021], which presents a tool that uses computer vision and image-based machine learning techniques to identify and consolidate duplicate video-based bug reports in a gaming development context from user submissions.

It's important to note that many output types are actually the transformed result of a classifier. For example, there are many tools for bug identification and most provide a text summarization. But often that summarization is linked to a large existing bank of pre-identified bug types. That is, code is transformed and ingested to a machine learning model, the model produces a classification (e.g., "bug type number 20"), which is then linked to a particular output message.

We provide a full table of the raw input type and the model output type before final transformation of nearly all machine learning model driven tools discussed in this report in Table 1, which is is located in Appendix B.

Finally, it's important to note that software development tools which use numbers as input are generally uninteresting to the community, from a machine learning development improvement perspective, as numerical manipulation as input is a well studied topic in machine learning generally. Graph-based machine learning as applied to software engineering is very new and papers are still being produced as of this writing. Yet results are promising, particularly when applied to the repository organizational context. That is, including the mapping of artifact relationships within a repository as input to a machine learning model. Which brings us to the next section of how inputs are actually transformed.

8.2 Input Transformation and Output Transformation

Machine learning models which ingest software engineering artifacts as input have been around for nearly as long as there has been machine learning. The question is how to do so, as deep learning models take in numbers, not source code, and are optimized over a loss function. The conversion from a software engineering artifact to something any given deep learning model can consume is known as an embedding. This embedding is almost always a vector, although it could be a tensor or matrix.

It's important to note that a variety of software artifacts can be embedded. [Theeten et al., 2019] presented an embedding called Import2Vec, which transformed entire software libraries across different programming languages to vectors. Other work, like CC2Vec by [T. Hoang and Lawall, 2020] embedded code change distributions as a vector, finding that their code change embedding had utility in log generation and just-in-time bug patch prediction – that is, improving predicting of bugs and possible fixes. Yet other types of embeddings include code itself, like Code2Vec and improvements on it [Compton et al., 2020], entire repositories like Repo2Vec [M. O. F. Rokon and Faloutsos, 2021], and graphs which map relations among artifacts within repositories in GraphCode2Vec in [Ma et al., 2022].

Almost all of these XtoVec papers rely on text as the initial input to be vectorized. In general, text which is code does not play well in traditional natural language processing paradigms. Yet most software engineering artifacts are a mix of code, technical jargon, and natural language. It is a non-trivial problem to disambiguate among the three, and then to process the code and jargon once they've been separated.

Consequently, there has been several years of work done to answer these questions. [Mantyla et al., 2018] was one of the first packages which automatically pre-processed text to simply disambiguate whether textual information was natural language or not. Other work like [Efstathiou et al., 2018] release a modified Word2Vec embedding trained on StackOverflow data so that ML practitioners had a natural language library which could actually handle technical jargon. Follow up work by [Biswas et al., 2019] analyzed different approaches to improve these types of libraries, finding that improving "off the shelf" natural language embeddings can easily be improved for the software engineering context simply by oversampling of jargon-infused data sources. This approach, however, requires that there is a sufficient diversity of data to oversample from. If the jargon one wants to include is related to a highly popular topic – say, general android application development, there's probably enough on StackOverflow to mine from. But if the topic is even more niche or even confidential, as is the case for highly technical projects at many national laboratories, this approach may suffer.

Subsequent work in [Efstathiou and Spinellis, 2019] focused on how to integrate natural language which is found in code itself, such as inline comments, with the surrounding code. The primary contribution was six language-specific pretrained distributed code representation models for Python, Java, PHP, C, C++, and C#. This work was followed up by [P. P. Prachi and Barr, 2020] which submitted their techniques to automatically and simultaneously tag natural vs programing languages. This was an improvement of over 20%. Similarly, [M. Osama and Ibrahim, 2020] submitted a disambiguation tool utilizing a scoring based technique trained on highly intertwined technical-code-language documents, namely software requirements documents written in natural language made for developers. Use of these two 2020 contributions is the current best approach for disambiguating code vs natural language in machine learning models in which the output is classification. That said, for situations where the output is a sequence, the sequence-to-sequence pretrained source code ML model presented by [C. Niu and Luo, 2022] is a novel approach, but the contribution is "research-ware" and still needs significant programming support for general purpose use.

All of the above papers present software contributions to a machine learning pipeline. But other work in this space examines existing tools to determine the best fit for code processing. For example, [M. Jimenez and Papadakis, 2018a] found that among the most common off-the-shelf word tokenizers, an approach known as Modified Kneser-Ney works best for code analysis, when balancing against reasonable computation time.

Other work wanted to validate what certain ML paradigms asserted was important vs human perception, with [A. Marques and Murphy, 2020] having 20 human software developers manually highlight the important parts of key artifacts. The authors found that an approach called frame semantics most closely matched the consensus human highlights. More recent work, which incorporates attention, includes [Y. Wan and

Jin, 2022], which found pretrained language embedding models could induce certain code syntax trees. This suggests that future embedding models should not only include raw text but a separate preprocessing step that extracts the code's syntax tree to help provide more accurate results.

Just as important as input encoding, however, is output encoding. Examples of output encodings in a classification context include one-hot encoding, scaled label encoding for ordinal values, and so forth. Work by [Gong and Chen, 2022], using software performance learning systems as the underlying models, studied 105 output type encoding configurations. That is, the authors varied output encoding types across a number of different classification models. They found that there are tradeoffs between accuracy and training time based on output encoding, with one hot encoding usually being the most accurate but taking the longest for the model to converge. Scaled label training, on the other hand, caused neural network model training to converge more quickly but had lower overall accuracy. The takeaway is that for classification tasks with code or jargon as input, one should use scaled labels for debugging the model and ballpark parameter tuning before switching to one hot encoding for the final training of the model.

8.3 Papers relating to an Algorithm, Model Architecture, or Tuning

Hyperparameter tuning is key to any neural network architecture, and most papers on it are applicable to models focused on software engineering applications. However, there is one paper, [M. Jimenez and Papadakis, 2018b], which reports on an automated tuning approach for word tokenization based on how natural source code is. That is, if a source code file contains lots of natural language comments, the tokenizer will treat the source code as more of a regular natural language file. And vice versa: if the source code file has strictly code syntax with little natural language, the tool will apply tokenization based on code rules. This 2018 tool, however, has mostly been made obsolete with general advances in embedding and network design (e.g., attention).

With the quick obsoleteness of some deep learning techniques, it's important to verify that a common base model for code ingestion, CodeBERT, is accurate for a wide vareity of tasks. CodeBERT is a transformer based machine learning model which can ingest a mix of code and natural language for a variety of classification and generative tasks. CodeBERT used natural language code search and code documentation generation as example use cases when it was initially introduced. Work by [X. Zhou and Lo, 2021] independently verified that CodeBERT was useful beyond these two narrow use cases, empirically finding that CodeBERT is generalizable far beyond its initial out-of-the-box training data.

While CodeBERT is certainly a specific machine learning tool targeted to the software engineering domain, the majority of machine learning model architecture for software engineering tools is not specialized for software engineering per-se. Most models, given different training data and input embeddings, would work just as well on other domain problems if properly tuned. That said, work like [A. LeClair and McMillan, 2021] suggests that ensemble models work the best for models with source code as input, with a performance boost of about 15% above singular model approaches. Further work, like [Majumder et al., 2018], suggests that non neural network models, like support vector machines, can preform just as well as deep learning but with significantly quicker training times.

Flipping the script a bit, software repository mining can also help machine learning development. Work like [G. Nguyen and Rajan, 2022] notes that finding the exact neural architecture for a problem is time consuming, requiring lots of tuning and tweaking with tools like Auto-Keras. Their paper introduced a tool, Manas, which takes a shortcut. Namely, the authors first searched and cataloged thousands of machine learning models in GitHub, and Manas uses the catalog to suggest machine learning architectures based on models that are already in use. Then, given a base architecture which is more likely to be closer to a working solution (for example, in image classification), significant time is saved.

Other work attempts to not just select a model, but to automate the entire ML process, from preprocessing steps to final output. [Saha et al., 2022] is a tool which takes in a high level human description of a machine learning pipeline and then predicts a plausible set of machine learning components in code which fit the description. Automatic architecture generation, for both ML and non-ML pipelines, while outside the scope of this review, is likely to gain more traction in the coming years, and is something to consider incorporating as a tool in the advance scientific computing developer's toolkit.

8.3.1 Curated Datasets

All of these pipelines and tools for machine learning are useless without quality training data, however. While there is only one curated dataset for machine learning in the software engineering space – a time series dataset for open source software evolution [B. L. Sousa and Franco, 2022] – there is plenty of work on effective mining practices to accrue training data in the first place. For example, [Z. Sun and Li, 2022] suggests a particular framework which includes a variety of rule based filters to clean Stack Overflow data will generally result in more accurate models which are trained on that data.

Other work like [Ahmed and Devanbu, 2022] suggests that, perhaps counter-intuitively, machine learning models which support multiple programming languages as text input are more robust than monolingual models, so long as the training data from the diverse languages all do roughly the same thing. For example, a model trained with "Hello World" in Java, Python, and C/C++ is more robust than just a single instance of "Hello World" in one language alone.

9 Cross cutting theme: Bots

One of the cross cutting themes of the software engineering experience is the growing reliance on various bots. In §7 we discussed Robin, a voice controlled "virtual teammate" for software development teams [B. da Silva and Sereesathien, 2020]. As of this writing, ChatGPT is making headlines around the world for its human-like text in response to queries.

The development of these bots, whether geared for general purpose language interaction like ChatGPT or specialized bots/automations for software development like Robin or GitHub's Copilot, is non trivial and often clunky. Challenges in chatbot development, in particular, were analyzed in [Abdellatif et al., 2020] which found that machine learning training of chatbots in the software development space was particularly difficult due to the limited data sets available and mix of code, jargon, and non-jargon natural language in many training sets.

To that end, [E. Koshchenko and Kovalenko, 2022] suggests bots designed for software development incorporate a multimodal approach. That is, if the bot's primary feature is to provide a recommendation to the developer to do X, the training should be sourced from a variety of collaboration platforms such as GitHub and Slack, as just focusing on one channel tends to lead to overfitting.

With these challenges in mind, many bots along the full spectrum of software engineering tasks have been developed. Notable are bots that write sample code [F. Wen and Bavota, 2021a], or that provide refactoring suggestions and other edits at pull request / code review time. These include well known bots like GitHub's Copilot, which tends to do better with Java projects than many other languages [Nguyen and Nadi, 2022]. But there's a wider question about how useful bots actually are. By one measure, the answer is, "somewhat," with [M. Wessel and Gerosa, 2020] finding that the existence of a code review bot on open source software projects increases the number of merged pull requests but decreases communication among developers. That work was followed up two years later in [Wessel et al., 2022] where 22 different design strategies and recommendations were presented to make bots better mediators in the human-pull request dynamic. To oversimplify, Wessel et al. suggests that bots should be partitioned into a separate space discrete from human-to-human activity, and also suggest that an overarching bot coordinator – that is, a bot which manages all the other bots – might be helpful.

Of especially great interest to the community is the question of determining if an action has been done by a human or a bot, with tools like BotHunter [A. Abdellatif and Shihab, 2022], BoDeGha [Chidambaram and Mazrae, 2022], and earlier tools like BIMAN [Dey et al., 2020] available for use.

The potential utility of bots for the advanced scientific computing space is large. National laboratories are on budgets, but bots are only used minimally. If tools like bots can add meaningful value to a developer's daily work, there can be impressive gains made. Effectively designing, evangelizing, and incorporating useful bots into the daily habits of developers is an area where ASC can really improve.

10 Cross cutting theme: Venues

As we examined these papers, we identified if there was a specific venue in which the research paper was based in. Namely, papers with data from/geared towards/analysis of open source projects, projects in start ups, projects in established industry players, and projects developed in academia.

To the best of our knowledge, there has not been a single paper published on software engineering specifically focused on how the development process is unique in the advanced scientific computing community of US National Laboratories. As we surveyed papers for this literature review, we found hundreds of papers focused on open source projects – papers which may have incidentally included a few open source ASC repositories in their corpus of evaluated projects, but ASC was not the focus of the study. We only found 27 papers focused on projects or repositories in industry, two related to start ups in some fashion, and only one from academia from the major software engineering research conferences.

The one research paper from academia about software engineering was not immediately applicable to the advanced scientific community: it was about the teaching of software tools to students in an academic context.

In the case of research related to start ups, much is not totally relevant to the ASC space: start ups are usually under enormous financial and time pressure to bring their product to market. The two papers in this space, [T. Besker and Bosch, 2018, C. Gralha and Arajo, 2018], while focusing on different issues like technical debt and requirements gathering, both conclude that typical pressures in a start up environment require corner cutting in one or more areas which eventually must be dealt with as the company stabilizes and the product matures. It's not a question of if corners will be cut, but how. This differs from the national laboratory context markedly, as most projects are very mature and new projects typically have longer time lines and larger budgets.

More interesting were the papers on projects in industry, but these works were typically applicable to any software engineering project or too niche to be of use to ASC. There are some interesting highlights: [A. Ju and Herzig, 2021] investigates how on-boarding processes impact development, finding that onboarding which has a new employee first learn the big picture of the tasks they're assigned to do, finishing their first task quickly, and receiving positive feedback for their first successfully completed tasks all strongly corresponded to positive long term employee outcomes. Thinking about how to do onboarding better in a national lab setting might help with long term attrition issues. Understanding how early interactions among developers can be modeled and subsequently used to predict future success is an open question for future work.

One common theme among industry focused papers was a reliance on using just one large tech company with enough resources to have its own research branch (e.g., Microsoft, Google, etc.) as the data source. That is to say, a common theme of some team at Microsoft sending out a survey to all of its developers, having them complete it, and then writing up the results – perhaps with a collaborator or two from another company. But Microsoft, Google, and other large tech firms are an abnormality in the vast ecosystem of computing. It is important to verify that findings from studies centered on these companies are replicable. Especially to other venues which aren't computing centric. Replicating works done at large tech companies at other large companies that aren't tech firms yet nonetheless heavily rely on programming and employ lots of software developers – like banks or state governments – is an important area of research to pay attention to going forward.

11 Conclusion

This report surveyed hundreds of papers, classified them, and evaluated several topics in depth. Although there are many papers relevant to software development generally, which are helpful to the advanced scientific computing community, there are no papers focused specifically on software engineering in the advanced scientific computing context. Formally and rigorously examining how aspects of the development process are similar or different in the ASC context is a fruitful area for further investigation.

A Taxonomy matrix of topics, categories, and cross cutting themes

This section is a user's reference for all of the papers read and classified for this literature review. It holds the "flattened" 2D matrix of a taxonomy of 18 software engineering topics partitioned by the categories Tools, Psychology/Social Science, Analysis / Broader Studies, and Curated Datasets.

Many of the papers also contain cross cutting themes. We annotate each paper with a cross cutting theme with the following key at the end of the citation:

^{A} = Bots
^{B} = DEI - General
^{C} = DEI - Race/Ethnicity/National Origin
^{D} = DEI - Sex/Gender
^{E} = Graph Theory
^{F} = Machine Learning
^{G} = Privacy or Security
^{H} = Venue - Academia
^{I} = Venue - Industry
^{J} = Venue - Open Source
^{K} = Venue - Start ups

A.1 Learning / School

Several papers revolve around the learning of software engineering along some dimension, with a number of papers focused on classroom learning. This topic incorporates any paper which touches on the learning of software or software engineering, or which has nexus with a school or university environment.

Tools

- Toward Automatic Summarization of Arbitrary Java Statements for Novice Programmers, *M. Hassan and E. Hill,* [Hassan and Hill, 2018];
- Teaching Software Maintenance, *K. Gallagher, M. Fioravanti and S. Kozaitis,* [K. Gallagher and Kozaitis, 2019]^{J};
- Recommending Exception Handling Code, *T. Nguyen, P. Vu and T. Nguyen,* [T. Nguyen and Nguyen, 2019b]^{F};
- A software maintenance-focused process and supporting toolset for academic environments, *R. Hardt,* [Hardt, 2020a];
- A toolset to support a software maintenance process in academic environments, *R. Hardt,* [Hardt, 2020b]^{H};

Psychology / Social Science

- How Modern News Aggregators Help Development Communities Shape and Share Knowledge, *M. Aniche et al.,* [Aniche et al., 2018]^{J};
- How Practitioners Perceive Coding Proficiency, *X. Xia, Z. Wan, P. S. Kochhar and D. Lo,* [X. Xia and Lo, 2019];
- Do Developers Discover New Tools On The Toilet?, *E. Murphy-Hill et al.,* [Murphy-Hill et al., 2019]^{I};
- Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study, *M. Endres, Z. Karas, X. Hu, I. Kovelman and W. Weimer,* [M. Endres and Weimer, 2021];

Analysis / Broader Studies

- Here We Go Again: Why Is It Difficult for Developers to Learn Another Programming Language?, *N. Shrestha, C. Botta, T. Barik and C. Parnin,* [N. Shrestha and Parnin, 2020];
- Towards Bidirectional Live Programming for Incomplete Programs, *X. Zhang and Z. Hu,* [Zhang and Hu, 2022];

A.2 Onboarding

Onboarding research relates to the process of inducing a developer to a project or team, or the steps which occur after evangelizing an existing project to other developers to adopt and these other developers first try it out.

Psychology / Social Science

- How Modern News Aggregators Help Development Communities Shape and Share Knowledge, *M. Aniche et al.,* [Aniche et al., 2018]^{J};
- Almost There: A Study on Quasi-Contributors in Open-Source Software Projects, *I. Steinmacher, G. Pinto, I. S. Wiese and M. A. Gerosa,* [I. Steinmacher and Gerosa, 2018]^{J};
- Do Developers Discover New Tools On The Toilet?, *E. Murphy-Hill et al.,* [Murphy-Hill et al., 2019]^{I};
- A Case Study of Onboarding in Software Teams: Tasks and Strategies, *A. Ju, H. Sajnani, S. Kelly and K. Herzig,* [A. Ju and Herzig, 2021]^{I};
- This Is Damn Slick!" Estimating the Impact of Tweets on Open Source Project Popularity and New Contributors, *H. Fang, H. Lamba, J. Herbsleb and B. Vasilescu,* [H. Fang and Vasilescu, 2022]^{J};

Analysis / Broader Studies

- Which Contributions Predict Whether Developers Are Accepted into Github Teams *Middleton, Justin and Murphy-Hill, Emerson and Green, Demetrius and Meade, Adam and Mayer, Roger and White, David and McDonald, Steve* [Middleton et al., 2018]^{J};
- Open Source Barriers to Entry, Revisited: A Sociotechnical Perspective, *C. Mendez et al.,* [Mendez et al., 2018]^{D};
- Assessing the Characteristics of FOSS Contributions in Network Automation Projects, *J. Anderson, I. Steinmacher and P. Rodeghero,* [J. Anderson and Rodeghero, 2020]^{J};
- Onboarding vs. Diversity, Productivity and Quality - Empirical Study of the OpenStack Ecosystem, *A. Foundjem, E. Eghan and B. Adams,* [A. Foundjem and Adams, 2021]^{BDJ};
- The Shifting Sands of Motivation: Revisiting What Drives Contributors in Open Source, *M. Gerosa et al.,* [Gerosa et al., 2021]^{J};

A.3 New features and requirements

The topic of inducing new features to a project and the requirements engineering which accompanies it consists of an entire subfield on its own, with entire conferences dedicated to the topic. Consequently, most requirement engineering tools end up being presented in conferences like RE. We focus here on just a few papers which lie in the intersection between requirements engineering and the broader software development process.

Psychology / Social Science

- Goal-Conflict Likelihood Assessment Based on Model Counting, *R. Degiovanni, P. Castro, M. Arroyo, M. Ruiz, N. Aguirre and M. Frias,* [R. Degiovanni and Frias, 2018]^{F};

- Do this! Do that!, and Nothing will Happen" Do Specifications Lead to Securely Stored Passwords?, *J. Hallett, N. Patnaik, B. Shreeve and A. Rashid,* [J. Hallett and Rashid, 2021][G];

Analysis / Broader Studies

- The Evolution of Requirements Practices in Software Startups, *C. Gralha, D. Damian, A. Wasserman, M. Goulao and J. Araujo,* [C. Gralha and Arajo, 2018][K];

A.4 Code writing and refactoring

The code writing and refactoring category involves the actual production and manipulation of symbols in the creation or modification of software – usually in some sort of editor. This is the stereotype of programming that non developers usually think as the whole of software development.

Tools

- Predicting Developers' IDE Commands with Machine Learning *Bulmer, Tyson and Montgomery, Lloyd and Damian, Daniela* [Bulmer et al., 2018][F];
- Deuce: A Lightweight User Interface for Structured Editing, *B. Hempel, J. Lubin, G. Lu and R. Chugh,* [B. Hempel and Chugh, 2018];
- Generating Accurate and Compact Edit Scripts Using Tree Differencing, *V. Frick, T. Grassauer, F. Beck and M. Pinzger,* [V. Frick and Pinzger, 2018][E];
- Statistical Learning of API Fully Qualified Names in Code Snippets of Online Forums, *H. Phan, H. A. Nguyen, N. M. Tran, L. H. Truong, A. T. Nguyen and T. N. Nguyen,* [H. Phan and Nguyen, 2018][F];
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018][F G];
- On Learning Meaningful Code Changes Via Neural Machine Translation, *M. Tufano, J. Pantiuchina, C. Watson, G. Bavota and D. Poshyvanyk,* [M. Tufano and Poshyvanyk, 2019][F];
- FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns, *P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule and M. Di Penta,* [P. T. Nguyen and Penta, 2019][F];
- Learning to Spot and Refactor Inconsistent Method Names, *K. Liu et al.,* [Liu et al., 2019][F];
- Personalized Code Recommendation, *T. Nguyen, P. Vu and T. Nguyen,* [T. Nguyen and Nguyen, 2019a][F];
- Deep Learning Anti-Patterns from Code Metrics History, *A. Barbez, F. Khomh and Y. -G. Gueheneuc,* [A. Barbez and Guhneuc, 2019][F];
- Analyzing and Supporting Adaptation of Online Code Examples, *T. Zhang, D. Yang, C. Lopes and M. Kim,* [T. Zhang and Kim, 2019];
- Suggesting Natural Method Names to Check Name Consistencies, *S. Nguyen, H. Phan, T. Le and T. N. Nguyen,* [S. Nguyen and Nguyen, 2020][F];
- Improving Data Scientist Efficiency with Provenance, *J. Hu, J. Joung, M. Jacobs, K. Z. Gajos and M. I. Seltzer,* [J. Hu and Seltzer, 2020];
- Fast and Memory-Efficient Neural Code Completion, *A. Svyatkovskiy, S. Lee, A. Hadjitofi, M. Riechert, J. V. Franco and M. Allamanis,* [A. Svyatkovskiy and Allamanis, 2021][F];
- Siri, Write the Next Method, *F. Wen, E. Aghajani, C. Nagy, M. Lanza and G. Bavota,* [F. Wen and Bavota, 2021a][A F];
- CodeRibbon: More Efficient Workspace Management and Navigation for Mainstream Development Environments, *B. P. Klein and A. Z. Henley,* [Klein and Henley, 2021];
- FeaRS: Recommending Complete Android Method Implementations, *F. Wen, V. Ferrari, E. Aghajani, C. Nagy, M. Lanza and G. Bavota,* [F. Wen and Bavota, 2021b][F];
- IDEAL: An Open-Source Identifier Name Appraisal Tool, *A. Peruma, V. Arnaoudova and C. D. Newman,* [A. Peruma and Newman, 2021];

- Sirius: Static Program Repair with Dependence Graph-Based Systematic Edit Patterns, *K. Noda, H. Yokoyama and S. Kikuchi,* [K. Noda and Kikuchi, 2021]^{EJ};
- Senatus - A Fast and Accurate Code-to-Code Recommendation Engine, *F. Silavong, S. Moran, A. Georgiadis, R. Saphal and R. Otter,* [F. Silavong and Otter, 2022]^{F};
- Type4Py: Practical Deep Similarity Learning-Based Type Inference for Python, *A. M. Mir, E. Latoskinas, S. Proksch and G. Gousios,* [A. M. Mir and Gousios, 2022]^{F};

Psychology / Social Science

- Do Software Engineers Use Autocompletion Features Differently than Other Developers? *Amlekar, Rahul and Gamboa, Andrés Felipe Rincón and Gallaba, Keheliya and McIntosh, Shane* [Amlekar et al., 2018];
- The Road to Live Programming: Insights from the Practice, *J. Kubelka, R. Robbes and A. Bergel,* [J. Kubelka and Bergel, 2018];
- Sentiment Polarity Detection for Software Development, *F. Calefato, F. Lanubile, F. Maiorano and N. Novielli,* [F. Calefato and Novielli, 2018]^{F};
- Towards Better Understanding Developer Perception of Refactoring, *E. A. Alomar,* [Alomar, 2019];
- Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters, *N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed and J. I. Maletic,* [N. J. Abid and Maletic, 2019];
- How Practitioners Perceive Coding Proficiency, *X. Xia, Z. Wan, P. S. Kochhar and D. Lo,* [X. Xia and Lo, 2019];
- Socio-Technical Work-Rate Increase Associates With Changes in Work Patterns in Online Projects, *F. Sarker, B. Vasilescu, K. Blincoe and V. Filkov,* [F. Sarker and Filkov, 2019]^{J};
- Do as I Do, Not as I Say: Do Contribution Guidelines Match the GitHub Contribution Process?, *O. Elazhary, M. -A. Storey, N. Ernst and A. Zaidman,* [O. Elazhary and Zaidman, 2019]^{J};
- Recognizing Developers' Emotions while Programming, *D. Girardi, N. Novielli, D. Fucci and F. Lanubile,* [D. Girardi and Lanubile, 2020];
- Neurological Divide: An fMRI Study of Prose and Code Writing, *R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer and K. Leach,* [R. Krueger and Leach, 2020];
- Refactoring Recommendations Based on the Optimization of Socio-Technical Congruence, *M. D. Stefano, F. Pecorelli, D. A. Tamburri, F. Palomba and A. D. Lucia,* [M. D. Stefano and Lucia, 2020];
- On the Naming of Methods: A Survey of Professional Developers, *R. Alsuhaibani, C. Newman, M. Decker, M. Collard and J. Maletic,* [R. Alsuhaibani and Maletic, 2021]^{I};

Analysis / Broader Studies

- Studying Developer Build Issues and Debugger Usage via Timeline Analysis in Visual Studio IDE *Bellman, Christopher and Seet, Ahmad and Baysal, Olga* [Bellman et al., 2018];
- Predicting Future Developer Behavior in the IDE Using Topic Models, *K. Damevski, H. Chen, D. C. Shepherd, N. A. Kraft and L. Pollock,* [K. Damevski and Pollock, 2018];
- Does the Propagation of Artifact Changes Across Tasks Reflect Work Dependencies?, *C. Mayr-Dorn and A. Egyed,* [Mayr-Dorn and Egyed, 2018];
- Towards a Model to Appraise and Suggest Identifier Names, *A. Peruma,* [Peruma, 2019];
- Self-Admitted Technical Debt Removal and Refactoring Actions: Co-Occurrence or More?, *M. Iammarino, F. Zampetti, L. Aversano and M. Di Penta,* [M. Iammarino and Penta, 2019]^{J};
- Investigating Context Adaptation Bugs in Code Clones, *M. Mondal, B. Roy, C. K. Roy and K. A. Schneider,* [M. Mondal and Schneider, 2019]^{J};
- An Empirical Study on the Usage of BERT Models for Code Completion, *M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. Di Penta and G. Bavota,* [M. Ciniselli and Bavota, 2021]^{F};
- An Empirical Evaluation of GitHub Copilot's Code Suggestions, *N. Nguyen and S. Nadi,* [Nguyen and Nadi, 2022]^{AF};
- To What Extent do Deep Learning-based Code Recommenders Generate Predictions by Cloning Code from the Training Set?, *M. Ciniselli, L. Pascarella and G. Bavota,* [M. Ciniselli and Bavota, 2022]^{FJ};

A.5 Help, Q&A, code comprehension, and documentation consumption

A key part of software development is an interaction between the writing of the code itself and learning about facets of the implementation details. This topic includes papers related to those parts of learning – whether interacting and reading from Stack Overflow, chatting with another developer, or digesting documentation.

Tools

- FaCoY – A Code-to-Code Search Engine, *K. Kim et al.,* [Kim et al., 2018];
- Programming Not Only by Example, *H. Peleg, S. Shoham and E. Yahav,* [H. Peleg and Yahav, 2018];
- Statistical Learning of API Fully Qualified Names in Code Snippets of Online Forums, *H. Phan, H. A. Nguyen, N. M. Tran, L. H. Truong, A. T. Nguyen and T. N. Nguyen,* [H. Phan and Nguyen, 2018]^{F};
- Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph, *H. Li et al.,* [Li et al., 2018]^{EF};
- Enlightened Debugging, *X. Li, S. Zhu, M. d'Amorim and A. Orso,* [X. Li and Orso, 2018]^{F};
- Context-Aware Software Documentation, *E. Aghajani,* [Aghajani, 2018];
- FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns, *P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule and M. Di Penta,* [P. T. Nguyen and Penta, 2019]^{F};
- Know-How in Programming Tasks: From Textual Tutorials to Task-Oriented Knowledge Graph, *J. Sun, Z. Xing, R. Chu, H. Bai, J. Wang and X. Peng,* [J. Sun and Peng, 2019]^{E};
- Automatic Identification of Rollback Edit with Reasons in Stack Overflow Q&A Site, *S. Mondal, G. Uddin and C. K. Roy,* [S. Mondal and Roy, 2020]^{F};
- Attention-based model for predicting question relatedness on Stack Overflow, *J. Pei, Y. Wu, Z. Qin, Y. Cong and J. Guan,* [J. Pei and Guan, 2021]^{F};
- Dialogue Management for Interactive API Search, *Z. Eberhart and C. McMillan,* [Eberhart and McMillan, 2021]^{F};
- Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data, *P. R. Mazrae, M. Izadi and A. Heydarnoori,* [P. R. Mazrae and Heydarnoori, 2021]^{F};
- SoCCMiner: A Source Code-Comments and Comment-Context Miner, *M. Sridharan, M. Mantyla, M. Claes and L. Rantala,* [M. Sridharan and Rantala, 2022];

Psychology / Social Science

- Evaluating How Developers Use General-Purpose Web-Search for Code Retrieval *Rahman, Md Masudur and Barson, Jed and Paul, Sydney and Kayani, Joshua and Lois, Federico Andrés and Quezada, Sebastián Fernandez and Parnin, Christopher and Stolee, Kathryn T. and Ray, Baishakhi* [Rahman et al., 2018]^{F};
- Are Code Examples on an Online Q&A Forum Reliable?: A Study of API Misuse on Stack Overflow, *T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan and M. Kim,* [T. Zhang and Kim, 2018];
- Understanding the Factors for Fast Answers in Technical Q&A Websites: An Empirical Study of Four Stack Exchange Websites, *S. Wang, T. -H. Chen and A. E. Hassan,* [S. Wang and Hassan, 2018]^{J};
- Measuring Program Comprehension: A Large-Scale Field Study with Professionals, *X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan and S. Li,* [X. Xia and Li, 2018];
- How Modern News Aggregators Help Development Communities Shape and Share Knowledge, *M. Aniche et al.,* [Aniche et al., 2018]^{J};
- Towards Better Understanding Developer Perception of Refactoring, *E. A. Alomar,* [Alomar, 2019];
- Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters, *N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed and J. I. Maletic,* [N. J. Abid and Maletic, 2019];
- Pattern-Based Mining of Opinions in Q&A Websites, *B. Lin, F. Zampetti, G. Bavota, M. Di Penta and M. Lanza,* [B. Lin and Lanza, 2019]^{FJ};
- An Empirical Study Assessing Source Code Readability in Comprehension, *J. Johnson, S. Lubo, N. Yedla, J. Aponte and B. Sharif,* [J. Johnson and Sharif, 2019]^{B};
- Do as I Do, Not as I Say: Do Contribution Guidelines Match the GitHub Contribution Process?, *O. Elazhary, M. -A. Storey, N. Ernst and A. Zaidman,* [O. Elazhary and Zaidman, 2019]^{J};

- Haste Makes Waste: An Empirical Study of Fast Answers in Stack Overflow, *Y. Lu, X. Mao, M. Zhou, Y. Zhang, T. Wang and Z. Li,* [Y. Lu and Li, 2020];
- Rollback Edit Inconsistencies in Developer Forum, *S. Mondal, G. Uddin and C. K. Roy,* [S. Mondal and Roy, 2021];
- The Mind Is a Powerful Place: How Showing Code Comprehensibility Metrics Influences Code Understanding, *M. Wyrich, A. Preikschat, D. Graziotin and S. Wagner,* [M. Wyrich and Wagner, 2021];

Analysis / Broader Studies

- Studying Developer Build Issues and Debugger Usage via Timeline Analysis in Visual Studio IDE *Bellman, Christopher and Seet, Ahmad and Baysal, Olga* [Bellman et al., 2018];
- Comprehension Effort and Programming Activities: Related? Or Not Related? *Rahman, Akond* [Rahman, 2018];
- Does the Propagation of Artifact Changes Across Tasks Reflect Work Dependencies?, *C. Mayr-Dorn and A. Egyed,* [Mayr-Dorn and Egyed, 2018];
- Analyzing and Supporting Adaptation of Online Code Examples, *T. Zhang, D. Yang, C. Lopes and M. Kim,* [T. Zhang and Kim, 2019];
- 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay, *H. Hata, C. Treude, R. G. Kula and T. Ishio,* [H. Hata and Ishio, 2019][J];
- Challenges in Chatbot Development: A Study of Stack Overflow Posts *Abdellatif, Ahmad and Costa, Diego and Badran, Khaled and Abdalkareem, Rabe and Shihab, Emad* [Abdellatif et al., 2020][A F J];
- Sentiment Analysis for Software Engineering: How Far Can Pre-trained Transformer Models Go?, *T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo and L. Jiang,* [T. Zhang and Jiang, 2020][F];
- Evaluating Code Readability and Legibility: An Examination of Human-centric Studies, *D. Oliveira, R. Bruno, F. Madeiral and F. Castor,* [D. Oliveira and Castor, 2020];
- Characterizing Task-Relevant Information in Natural Language Software Artifacts, *A. Marques, N. C. Bradley and G. C. Murphy,* [A. Marques and Murphy, 2020];
- Studying the Change Histories of Stack Overflow and GitHub Snippets, *S. S. Manes and O. Baysal,* [Manes and Baysal, 2021][J];
- Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, *G. L. Scoccia, P. Migliarini and M. Autili,* [G. L. Scoccia and Autili, 2021][J];
- Googling for Software Development: What Developers Search For and What They Find, *A. Hora,* [Hora, 2021];
- Does This Apply to Me? An Empirical Study of Technical Context in Stack Overflow, *A. Galappaththi, S. Nadi and C. Treude,* [A. Galappaththi and Treude, 2022][J];
- Multimodal Recommendation of Messenger Channels, *E. Koshchenko, E. Klimov and V. Kovalenko,* [E. Koshchenko and Kovalenko, 2022][A];
- Bridging Pre-trained Models and Downstream Tasks for Source Code Understanding, *D. Wang et al.,* [Wang et al., 2022][F];

Curated Datasets

- Two Datasets for Sentiment Analysis in Software Engineering, *B. Lin, F. Zampetti, R. Oliveto, M. Di Penta, M. Lanza and G. Bavota,* [B. Lin and Bavota, 2018][F];
- A Gold Standard for Emotion Annotation in Stack Overflow *Novielli, Nicole and Calefato, Fabio and Lanubile, Filippo* [Novielli et al., 2018][F];
- SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts, *Baltes, Sebastian and Dumani, Lorik and Treude, Christph and Diehl, Stephan* [Baltes et al., 2018]
- SOSum: A Dataset of Stack Overflow Post Summaries, *B. Kou, Y. Di, M. Chen and T. Zhang,* [B. Kou and Zhang, 2022];

A.6 Documentation production

In order for other developers to read documentation and have it be useful, a programmer (or bot) must also create documentation. This topic is all about the creation of code documentation and code summarization in natural language so that humans can readily understand what's going on without having to do manual code tracing. It also can include papers with tangential elements, such as the creation of documentation while refactoring, for example

Tools

- Context-Aware Software Documentation, *E. Aghajani,* [Aghajani, 2018];
- Toward Automatic Summarization of Arbitrary Java Statements for Novice Programmers, *M. Hassan and E. Hill,* [Hassan and Hill, 2018];
- Retrieval-based Neural Source Code Summarization, *J. Zhang, X. Wang, H. Zhang, H. Sun and X. Liu,* [J. Zhang and Liu, 2020]^{F};
- Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data, *P. R. Mazrae, M. Izadi and A. Heydarnoori,* [P. R. Mazrae and Heydarnoori, 2021]^{F};
- Towards Reliable Agile Iterative Planning via Predicting Documentation Changes of Work Items, *J. Pasuksmit, P. Thongtanunam and S. Karunasekera,* [J. Pasuksmit and Karunasekera, 2022]^{F};
- RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation, *Z. Li, G. Q. Chen, C. Chen, Y. Zou and S. Xu,* [Z. Li and Xu, 2022]^{F};

Psychology / Social Science

- Towards Better Understanding Developer Perception of Refactoring, *E. A. Alomar,* [Alomar, 2019];
- Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented?, *J. Pasuksmit, P. Thongtanunam and S. Karunasekera,* [J. Pasuksmit and Karunasekera, 2021];
- Comments on Comments: Where Code Review and Documentation Meet, *N. Rao, J. Tsay, M. Hirzel and V. J. Hellendoorn,* [N. Rao and Hellendoorn, 2022];
- Practitioners' Expectations on Automated Code Comment Generation, *X. Hu, X. Xia, D. Lo, Z. Wan, Q. Chen and T. Zimmermann,* [X. Hu and Zimmermann, 2022]^{F};

Analysis / Broader Studies

- When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects, *A. Head, C. Sadowski, E. Murphy-Hill and A. Knight,* [A. Head and Knight, 2018];
- Inappropriate Usage Examples in Web API Documentations, *M. Hosono et al.,* [Hosono et al., 2019];
- Software Documentation Issues Unveiled, *E. Aghajani et al.,* [Aghajani et al., 2019]^{J};
- Software Documentation: The Practitioners' Perspective, *E. Aghajani et al.,* [Aghajani et al., 2020];
- On the need for automatic knowledge management in modern collaboration tools to improve software maintenance, *V. Balachandran,* [Balachandran, 2020];
- An Empirical Study on Code Comment Completion, *A. Mastropaolo, E. Aghajani, L. Pascarella and G. Bavota,* [A. Mastropaolo and Bavota, 2021]^{F};

A.7 Testing

Well tested code leads to fewer bugs. But what is testing? Do most developers find testing to be a chore? How does it factor into the larger software process? Research papers in this topic address these flavors of questions.

Tools

- Automatic Test Smell Detection Using Information Retrieval Techniques, *F. Palomba, A. Zaidman and A. De Lucia,* [F. Palomba and Lucia, 2018];
- Automatically Generating Precise Oracles from Structured Natural Language Specifications, *M. Motwani and Y. Brun,* [Motwani and Brun, 2019]^{F};
- Automatic Web Testing Using Curiosity-Driven Reinforcement Learning, *Y. Zheng et al.,* [Zheng et al., 2021]^{AFGJ};
- Graph-Based Fuzz Testing for Deep Learning Inference Engines, *W. Luo, D. Chai, X. Ruan, J. Wang, C. Fang and Z. Chen,* [W. Luo and Chen, 2021]^{EF};
- Automated Assertion Generation via Information Retrieval and Its Integration with Deep learning, *H. Yu et al.,* [Yu et al., 2022]^{F};

Psychology / Social Science

- How the Experience of Development Teams Relates to Assertion Density of Test Classes, *G. Catolino, F. Palomba, A. Zaidman and F. Ferrucci,* [G. Catolino and Ferrucci, 2019];

Analysis / Broader Studies

- When Testing Meets Code Review: Why and How Developers Review Tests, *D. Spadini, M. Aniche, M. -A. Storey, M. Bruntink and A. Bacchelli,* [D. Spadini and Bacchelli, 2018b];
- On the Relation of Test Smells to Software Code Quality, *D. Spadini, F. Palomba, A. Zaidman, M. Bruntink and A. Bacchelli,* [D. Spadini and Bacchelli, 2018a];
- How Do Code Changes Evolve in Different Platforms? A Mining-Based Investigation, *M. Viggiato, J. Oliveira, E. Figueiredo, P. Jamshidi and C. Kastner,* [M. Viggiato and Kstner, 2019];
- Studying Test Annotation Maintenance in the Wild, *D. J. Kim, N. Tsantalis, T. -H. Chen and J. Yang,* [D. J. Kim and Yang, 2021];

A.8 Commits, merges, and conflicts

Once code is written, documentation developed, and tests executed, it's time to commit. In modern software projects, that includes committing the code and then (potentially) dealing with merges and merge conflicts. This is a tedious challenge, and several works have focused on what happens when there's complex code overlaps. This topic does blur somewhat with the topic of pull requests, which often happen immediately after (or alongside) merge conflicts are sorted out.

Tools

- Linking Source Code to Untangled Change Intents, *X. Liu, L. Huang, C. Li and V. Ng,* [X. Liu and Ng, 2018]^{F};
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018]^{FG};
- Detecting and Characterizing Bots That Commit Code *Dey, Tapajit and Mousavi, Sara and Ponce, Eduardo and Fry, Tanner and Vasilescu, Bogdan and Filippova, Anna and Mockus, Audris* [Dey et al., 2020]^{AFJ};
- Planning for Untangling: Predicting the Difficulty of Merge Conflicts, *C. Brindescu, I. Ahmed, R. Leano and A. Sarma,* [C. Brindescu and Sarma, 2020]^{F};
- FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation, *J. Dong et al.,* [Dong et al., 2022]^{EF};

Psychology / Social Science

- Communicative Intention in Code Review Questions, *F. Ebert, F. Castor, N. Novielli and A. Serebrenik,* [F. Ebert and Serebrenik, 2018];
- Socio-Technical Work-Rate Increase Associates With Changes in Work Patterns in Online Projects, *F. Sarker, B. Vasilescu, K. Blincoe and V. Filkov,* [F. Sarker and Filkov, 2019]^{J};
- Lifting the Curtain on Merge Conflict Resolution: A Sensemaking Perspective, *C. Brindescu, Y. Ramirez, A. Sarma and C. Jensen,* [C. Brindescu and Jensen, 2020];

Analysis / Broader Studies

- Analyzing Conflict Predictors in Open-Source Java Projects *Accioly, Paola and Borba, Paulo and Silva, Léuson and Cavalcanti, Guilherme* [Accioly et al., 2018]^{J};
- How do Multiple Pull Requests Change the Same Code: A Study of Competing Pull Requests in GitHub, *X. Zhang et al.,* [Zhang et al., 2018]^{J};
- The List is the Process: Reliable Pre-Integration Tracking of Commits on Mailing Lists, *R. Ramsauer, D. Lohmann and W. Mauerer,* [R. Ramsauer and Mauerer, 2019]^{J};
- Can Program Synthesis be Used to Learn Merge Conflict Resolutions? An Empirical Analysis, *R. Pan, V. Le, N. Nagappan, S. Gulwani, S. Lahiri and M. Kaufman,* [R. Pan and Kaufman, 2021]^{J};
- On the Evaluation of Commit Message Generation Models: An Experimental Study, *W. Tao et al.,* [Tao et al., 2021]^{F};
- Which contributions count? Analysis of attribution in open source, *J. -G. Young, A. Casari, K. McLaughlin, M. Z. Trujillo, L. Hebert-Dufresne and J. P. Bagrow,* [J. G. Young and Bagrow, 2021]^{J};
- Why Security Defects Go Unnoticed During Code Reviews? A Case-Control Study of the Chromium OS Project, *R. Paul, A. K. Turzo and A. Bosu,* [R. Paul and Bosu, 2021]^{GJ};
- What Makes a Good Commit Message?, *Y. Tian, Y. Zhang, K. -J. Stol, L. Jiang and H. Liu,* [Y. Tian and Liu, 2022]^{J};

Curated Datasets

- A Graph-Based Dataset of Commit History of Real-World Android Apps *Geiger, Franz-Xaver and Malavolta, Ivano and Pascarella, Luca and Palomba, Fabio and Di Nucci, Dario and Bacchelli, Alberto* [Geiger et al., 2018]^{J};

A.9 Pull requests and code reviews

Code is complete and merged / compatible with an existing project. Now comes the pull request and code reviews: a human element that determines whether your code is up to snuff and good enough to be part of the project. These papers

Tools

- BLIMP Tracer: Integrating Build Impact Analysis with Code Review, *R. Wen, D. Gilbert, M. G. Roche and S. McIntosh,* [R. Wen and McIntosh, 2018];
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018]^{FG};
- Developers' Game: A Preliminary Study Concerning a Tool for Automated Developers Assessment, *W. Fracz and J. Dajda,* [Frcz and Dajda, 2018];
- On Learning Meaningful Code Changes Via Neural Machine Translation, *M. Tufano, J. Pantiuchina, C. Watson, G. Bavota and D. Poshyvanyk,* [M. Tufano and Poshyvanyk, 2019]^{F};
- Expanding the Number of Reviewers in Open-Source Projects by Recommending Appropriate Developers, *A. Chueshev, J. Lawall, R. Bendraou and T. Ziadi,* [A. Chueshev and Ziadi, 2020]^{J};

- Robin: A Voice Controlled Virtual Teammate for Software Developers and Teams, *B. da Silva, C. Hebert, A. Rawka and S. Sereesathien,* [B. da Silva and Sereesathien, 2020]$^{A\,F}$;
- Towards Automating Code Review Activities, *R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk and G. Bavota,* [R. Tufano and Bavota, 2021]^{F};
- Using Pre-Trained Models to Boost Code Review Automation, *R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk and G. Bavota,* [R. Tufano and Bavota, 2022]^{F};
- Modeling Review History for Reviewer Recommendation: A Hypergraph Approach, *G. Rong, Y. Zhang, L. Yang, F. Zhang, H. Kuang and H. Zhang,* [G. Rong and Zhang, 2022]^{E};

Psychology / Social Science

- Was My Contribution Fairly Reviewed?" A Framework to Study the Perception of Fairness in Modern Code Reviews, *D. M. German, G. Robles, G. Poo-Caamano, X. Yang, H. Iida and K. Inoue,* [D. M. German and Inoue, 2018]^{J};
- Almost There: A Study on Quasi-Contributors in Open-Source Software Projects, *I. Steinmacher, G. Pinto, I. S. Wiese and M. A. Gerosa,* [I. Steinmacher and Gerosa, 2018]^{J};
- Socio-Technical Work-Rate Increase Associates With Changes in Work Patterns in Online Projects, *F. Sarker, B. Vasilescu, K. Blincoe and V. Filkov,* [F. Sarker and Filkov, 2019]^{J};
- Less is More: Supporting Developers in Vulnerability Detection during Code Review, *L. Braz, C. Aeberhard, G. Calikli and A. Bacchelli,* [L. Braz and Bacchelli, 2022];

Analysis / Broader Studies

- When Testing Meets Code Review: Why and How Developers Review Tests, *D. Spadini, M. Aniche, M. -A. Storey, M. Bruntink and A. Bacchelli,* [D. Spadini and Bacchelli, 2018b];
- Studying the Impact of Adopting Continuous Integration on the Delivery Time of Pull Requests *Bernardo, João Helis and da Costa, Daniel Alencar and Kulesza, Uirá* [Bernardo et al., 2018]^{J};
- How do Multiple Pull Requests Change the Same Code: A Study of Competing Pull Requests in GitHub, *X. Zhang et al.,* [Zhang et al., 2018]^{J};
- Effects of Adopting Code Review Bots on Pull Requests to OSS Projects, *M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher and M. A. Gerosa,* [M. Wessel and Gerosa, 2020]$^{A\,J}$;
- How Does Modern Code Review Impact Software Design Degradation? An In-depth Empirical Study, *A. Uchoa et al.,* [Uchoa et al., 2020];
- Predicting Design Impactful Changes in Modern Code Review: A Large-Scale Empirical Study, *A. Uchoa et al.,* [Uchoa et al., 2021]$^{F\,J}$;
- Does Code Review Promote Conformance? A Study of OpenStack Patches, *P. Sri-iesaranusorn, R. G. Kula and T. Ishio,* [P. Sri-iesaranusorn and Ishio, 2021]^{J};
- Mining Code Review Data to Understand Waiting Times Between Acceptance and Merging: An Empirical Analysis, *G. Kudrjavets, A. Kumar, N. Nagappan and A. Rastogi,* [G. Kudrjavets and Rastogi, 2022b]^{J};
- The Unexplored Treasure Trove of Phabricator Code Reviews, *G. Kudrjavets, N. Nagappan and A. Rastogi,* [G. Kudrjavets and Rastogi, 2022a]^{J};
- Bots for Pull Requests: The Good, the Bad, and the Promising, *M. Wessel et al.,* [Wessel et al., 2022]$^{A\,J}$;
- Which bugs are missed in code reviews: An empirical study on SmartSHARK dataset, *F. Khoshnoud, A. R. Nasab, Z. Toudeji and A. Sami,* [F. Khoshnoud and Sami, 2022]^{J};

Curated Datasets

- A Graph-Based Dataset of Commit History of Real-World Android Apps *Geiger, Franz-Xaver and Malavolta, Ivano and Pascarella, Luca and Palomba, Fabio and Di Nucci, Dario and Bacchelli, Alberto* [Geiger et al., 2018]^{J};

A.10 Smells and quality

This topic has to do with the metrics of code – namely, papers which revolve around code quality, smells, or other similar types of measurements.

Tools

- Automatic Test Smell Detection Using Information Retrieval Techniques, *F. Palomba, A. Zaidman and A. De Lucia,* [F. Palomba and Lucia, 2018];
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018]$^{\mathrm{F\,G}}$;
- Beyond Metadata: Code-Centric and Usage-Based Analysis of Known Vulnerabilities in Open-Source Software, *S. E. Ponta, H. Plate and A. Sabetta,* [S. E. Ponta and Sabetta, 2018]$^{\mathrm{J}}$;
- Developers' Game: A Preliminary Study Concerning a Tool for Automated Developers Assessment, *W. Fracz and J. Dajda,* [Frcz and Dajda, 2018];
- Deep Learning Anti-Patterns from Code Metrics History, *A. Barbez, F. Khomh and Y. -G. Gueheneuc,* [A. Barbez and Guhneuc, 2019]$^{\mathrm{F}}$;
- Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction, *A. Trautsch, S. Herbold and J. Grabowski,* [A. Trautsch and Grabowski, 2020]$^{\mathrm{F}}$;
- Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data, *P. R. Mazrae, M. Izadi and A. Heydarnoori,* [P. R. Mazrae and Heydarnoori, 2021]$^{\mathrm{F}}$;
- RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation, *Z. Li, G. Q. Chen, C. Chen, Y. Zou and S. Xu,* [Z. Li and Xu, 2022]$^{\mathrm{F}}$;

Psychology / Social Science

- The Mind Is a Powerful Place: How Showing Code Comprehensibility Metrics Influences Code Understanding, *M. Wyrich, A. Preikschat, D. Graziotin and S. Wagner,* [M. Wyrich and Wagner, 2021];

Analysis / Broader Studies

- Improving Code: The (Mis) Perception of Quality Metrics, *J. Pantiuchina, M. Lanza and G. Bavota,* [J. Pantiuchina and Bavota, 2018];
- On the Relation of Test Smells to Software Code Quality, *D. Spadini, F. Palomba, A. Zaidman, M. Bruntink and A. Bacchelli,* [D. Spadini and Bacchelli, 2018a];
- Design Smell Detection and Analysis for Open Source Java Software, *A. Imran,* [Imran, 2019]$^{\mathrm{J}}$;
- Which contributions count? Analysis of attribution in open source, *J. -G. Young, A. Casari, K. McLaughlin, M. Z. Trujillo, L. Hebert-Dufresne and J. P. Bagrow,* [J. G. Young and Bagrow, 2021]$^{\mathrm{J}}$;
- An Evolutionary Analysis of Software-Architecture Smells, *P. Gnoyke, S. Schulze and J. Kruger,* [P. Gnoyke and Krger, 2021];

Curated Datasets

- QScored: A Large Dataset of Code Smells and Quality Metrics, *T. Sharma and M. Kessentini,* [Sharma and Kessentini, 2021a]$^{\mathrm{J}}$;

A.11 Maintainability, technical debt, production performance

Once code is written and its quality analyzed, there are other metrics recorded about its behavior under real-world conditions. This topic is all about the behavior of code once its been released and how changes are measured and occur.

Tools

- Enlightened Debugging, *X. Li, S. Zhu, M. d'Amorim and A. Orso,* [X. Li and Orso, 2018]^{F};
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018]^{FG};
- Supporting Software Architecture Maintenance by Providing Task-Specific Recommendations, *M. Galster, C. Treude and K. Blincoe,* [M. Galster and Blincoe, 2019]^{F};
- How Android Developers Handle Evolution-induced API Compatibility Issues: A Large-scale Study, *H. Xia et al.,* [Xia et al., 2020]^{F};
- FeaRS: Recommending Complete Android Method Implementations, *F. Wen, V. Ferrari, E. Aghajani, C. Nagy, M. Lanza and G. Bavota,* [F. Wen and Bavota, 2021b]^{F};
- Sirius: Static Program Repair with Dependence Graph-Based Systematic Edit Patterns, *K. Noda, H. Yokoyama and S. Kikuchi,* [K. Noda and Kikuchi, 2021]^{EJ};
- You Look so Different: Finding Structural Clones and Subclones in Java Source Code, *W. Amme, T. S. Heinze and A. Schafer,* [W. Amme and Schfer, 2021]^{F};
- DEAR: A Novel Deep Learning-based Approach for Automated Program Repair, *Y. Li, S. Wang and T. N. Nguyen,* [Y. Li and Nguyen, 2022]^{F};
- Tooling for Time- and Space-efficient git Repository Mining, *F. Heseding, W. Scheibel and J. Dollner,* [F. Heseding and Dllner, 2022];

Psychology / Social Science

- Embracing Technical Debt, from a Startup Company Perspective, *T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe and J. Bosch,* [T. Besker and Bosch, 2018]^{IK};
- Do You Remember This Source Code?, *J. Kruger, J. Wiemann, W. Fenske, G. Saake and T. Leich,* [J. Krger and Leich, 2018];
- How the Experience of Development Teams Relates to Assertion Density of Test Classes, *G. Catolino, F. Palomba, A. Zaidman and F. Ferrucci,* [G. Catolino and Ferrucci, 2019];
- Haste Makes Waste: An Empirical Study of Fast Answers in Stack Overflow, *Y. Lu, X. Mao, M. Zhou, Y. Zhang, T. Wang and Z. Li,* [Y. Lu and Li, 2020];
- Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented?, *J. Pasuksmit, P. Thongtanunam and S. Karunasekera,* [J. Pasuksmit and Karunasekera, 2021];

Analysis / Broader Studies

- Was Self-Admitted Technical Debt Removal a Real Removal? An in-Depth Perspective *Zampetti, Fiorella and Serebrenik, Alexander and Di Penta, Massimiliano* [Zampetti et al., 2018]^{J};
- How Maintainability Issues of Android Apps Evolve, *I. Malavolta, R. Verdecchia, B. Filipovic, M. Bruntink and P. Lago,* [I. Malavolta and Lago, 2018]^{J};
- How Do Code Changes Evolve in Different Platforms? A Mining-Based Investigation, *M. Viggiato, J. Oliveira, E. Figueiredo, P. Jamshidi and C. Kastner,* [M. Viggiato and Kstner, 2019];
- 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay, *H. Hata, C. Treude, R. G. Kula and T. Ishio,* [H. Hata and Ishio, 2019]^{J};
- Self-Admitted Technical Debt Removal and Refactoring Actions: Co-Occurrence or More?, *M. Iammarino, F. Zampetti, L. Aversano and M. Di Penta,* [M. Iammarino and Penta, 2019]^{J};
- Investigating Context Adaptation Bugs in Code Clones, *M. Mondal, B. Roy, C. K. Roy and K. A. Schneider,* [M. Mondal and Schneider, 2019]^{J};
- How Does Modern Code Review Impact Software Design Degradation? An In-depth Empirical Study, *A. Uchoa et al.,* [Uchoa et al., 2020];
- On the Recall of Static Call Graph Construction in Practice, *L. Sui, J. Dietrich, A. Tahir and G. Fourtounis,* [L. Sui and Fourtounis, 2020]^{E};

- Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, *G. L. Scoccia, P. Migliarini and M. Autili,* [G. L. Scoccia and Autili, 2021]^{J};
- An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems, *Y. Tang, R. Khatchadourian, M. Bagherzadeh, R. Singh, A. Stewart and A. Raja,* [Y. Tang and Raja, 2021]^{F};
- Cross-language Code Coupling Detection: A Preliminary Study on Android Applications, *B. Shen et al.,* [Shen et al., 2021]^{J};
- To What Extent do Deep Learning-based Code Recommenders Generate Predictions by Cloning Code from the Training Set?, *M. Ciniselli, L. Pascarella and G. Bavota,* [M. Ciniselli and Bavota, 2022]^{FJ};
- Log-based Anomaly Detection with Deep Learning: How Far Are We?, *V. -H. Le and H. Zhang,* [Le and Zhang, 2022]^{FJ};

Curated Datasets

- QScored: A Large Dataset of Code Smells and Quality Metrics, *T. Sharma and M. Kessentini,* [Sharma and Kessentini, 2021a]^{J};

A.12 Bugs, faults, and vulnerabilities

This topic deals with problems in the code. This topic is very large, and not all papers related to bugs and other problems which were published in the three conferences surveyed are included here. But a significant and representative number are listed, particularly ones which may have some utility in the advanced scientific computing space.

Tools

- Cloned Buggy Code Detection in Practice Using Normalized Compression Distance, *T. Ishio, N. Maeda, K. Shibuya and K. Inoue,* [T. Ishio and Inoue, 2018];
- Enlightened Debugging, *X. Li, S. Zhu, M. d'Amorim and A. Orso,* [X. Li and Orso, 2018]^{F};
- CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects *Nayrolles, Mathieu and Hamou-Lhadj, Abdelwahab* [Nayrolles and Hamou-Lhadj, 2018]^{FG};
- Developers' Game: A Preliminary Study Concerning a Tool for Automated Developers Assessment, *W. Fracz and J. Dajda,* [Frcz and Dajda, 2018];
- Recommending Exception Handling Code, *T. Nguyen, P. Vu and T. Nguyen,* [T. Nguyen and Nguyen, 2019b]^{F};
- Learning to Spot and Refactor Inconsistent Method Names, *K. Liu et al.,* [Liu et al., 2019]^{F};
- Ticket Tagger: Machine Learning Driven Issue Classification, *R. Kallis, A. Di Sorbo, G. Canfora and S. Panichella,* [R. Kallis and Panichella, 2019]^{FJ};
- Learning to Identify Security-Related Issues Using Convolutional Neural Networks, *D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cardenas, D. Poshyvanyk and C. Shenefiel,* [D. N. Palacio and Shenefiel, 2019]^{F};
- Improving Data Scientist Efficiency with Provenance, *J. Hu, J. Joung, M. Jacobs, K. Z. Gajos and M. I. Seltzer,* [J. Hu and Seltzer, 2020];
- Graph Neural Network-based Vulnerability Predication, *Q. Feng, C. Feng and W. Hong,* [Q. Feng and Hong, 2020]^{F};
- Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction, *A. Trautsch, S. Herbold and J. Grabowski,* [A. Trautsch and Grabowski, 2020]^{F};
- Can I Solve It? Identifying APIs Required to Complete OSS Tasks, *F. Santos, I. Wiese, B. Trinkenreich, I. Steinmacher, A. Sarma and M. A. Gerosa,* [F. Santos and Gerosa, 2021]^{F};
- Applying CodeBERT for Automated Program Repair of Java Simple Bugs, *E. Mashhadi and H. Hemmati,* [Mashhadi and Hemmati, 2021]^{F};

- It Takes Two to Tango: Combining Visual and Textual Information for Detecting Duplicate Video-Based Bug Reports, *N. Cooper, C. Bernal-Cardenas, O. Chaparro, K. Moran and D. Poshyvanyk,* [N. Cooper and Poshyvanyk, 2021];
- Sirius: Static Program Repair with Dependence Graph-Based Systematic Edit Patterns, *K. Noda, H. Yokoyama and S. Kikuchi,* [K. Noda and Kikuchi, 2021]^{EJ};
- Towards Bidirectional Live Programming for Incomplete Programs, *X. Zhang and Z. Hu,* [Zhang and Hu, 2022];
- LineVD: Statement-level Vulnerability Detection using Graph Neural Networks, *D. Hin, A. Kan, H. Chen and M. A. Babar,* [D. Hin and Babar, 2022]^{EF};
- Fast Changeset-based Bug Localization with BERT, *A. Ciborowska and K. Damevski,* [Ciborowska and Damevski, 2022]^{F};
- Online Summarizing Alerts through Semantic and Behavior Information, *J. Chen, P. Wang and W. Wang,* [J. Chen and Wang, 2022]^{F};
- DEAR: A Novel Deep Learning-based Approach for Automated Program Repair, *Y. Li, S. Wang and T. N. Nguyen,* [Y. Li and Nguyen, 2022]^{F};

Psychology / Social Science

- Do Programmers Work at Night or During the Weekend?, *M. Claes, M. V. Mantyla, M. Kuutila and B. Adams,* [M. Claes and Adams, 2018]^{J};
- An Empirical Study Assessing Source Code Readability in Comprehension, *J. Johnson, S. Lubo, N. Yedla, J. Aponte and B. Sharif,* [J. Johnson and Sharif, 2019]^{B};
- Studying the Impact of Policy Changes on Bug Handling Performance, *Z. Abou Khalil,* [Khalil, 2019];
- The Relationship Between Cognitive Complexity and the Probability of Defects, *B. S. Alqadi,* [Alqadi, 2019];
- On the Relationship between User Churn and Software Issues *El Zarif, Omar and Da Costa, Daniel Alencar and Hassan, Safwat and Zou, Ying* [El Zarif et al., 2020]^{FIJ};
- Failures and Fixes: A Study of Software System Incident Response, *J. Sillito and E. Kutomi,* [Sillito and Kutomi, 2020];
- Mea culpa: How developers fix their own simple bugs differently from other developers, *W. Zhu and M. W. Godfrey,* [Zhu and Godfrey, 2021];
- How heated is it? Understanding GitHub locked issues, *I. Ferreira, B. Adams and J. Cheng,* [I. Ferreira and Cheng, 2022]^{J};

Analysis / Broader Studies

- Studying Developer Build Issues and Debugger Usage via Timeline Analysis in Visual Studio IDE *Bellman, Christopher and Seet, Ahmad and Baysal, Olga* [Bellman et al., 2018];
- When Testing Meets Code Review: Why and How Developers Review Tests, *D. Spadini, M. Aniche, M. -A. Storey, M. Bruntink and A. Bacchelli,* [D. Spadini and Bacchelli, 2018b];
- How Maintainability Issues of Android Apps Evolve, *I. Malavolta, R. Verdecchia, B. Filipovic, M. Bruntink and P. Lago,* [I. Malavolta and Lago, 2018]^{J};
- Do Automated Program Repair Techniques Repair Hard and Important Bugs?, *M. Motwani, S. Sankaranarayanan, R. Just and Y. Brun,* [M. Motwani and Brun, 2018];
- A Conceptual Replication Study on Bugs that Get Fixed in Open Source Software, *H. Wang and H. Kagdi,* [Wang and Kagdi, 2018]^{FJ};
- How Do Code Changes Evolve in Different Platforms? A Mining-Based Investigation, *M. Viggiato, J. Oliveira, E. Figueiredo, P. Jamshidi and C. Kastner,* [M. Viggiato and Kstner, 2019];
- Mining Software Defects: Should We Consider Affected Releases?, *S. Yatish, J. Jiarpakdee, P. Thongtanunam and C. Tantithamthavorn,* [S. Yatish and Tantithamthavorn, 2019]^{F};
- Investigating Context Adaptation Bugs in Code Clones, *M. Mondal, B. Roy, C. K. Roy and K. A. Schneider,* [M. Mondal and Schneider, 2019]^{J};
- An Industrial Study on the Differences between Pre-Release and Post-Release Bugs, *R. Rwemalika, M. Kintis, M. Papadakis, Y. Le Traon and P. Lorrach,* [R. Rwemalika and Lorrach, 2019]^{I};

- Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, *G. L. Scoccia, P. Migliarini and M. Autili,* [G. L. Scoccia and Autili, 2021]^{J};
- Early Life Cycle Software Defect Prediction. Why? How?, *S. N.C., S. Majumder and T. Menzies,* [S. N.C. and Menzies, 2021]$^{F\,J}$;
- Incorporating Multiple Features to Predict Bug Fixing Time with Neural Networks, *W. Yuan, Y. Xiong, H. Sun and X. Liu,* [W. Yuan and Liu, 2021]^{F};
- Duplicate Bug Report Detection by Using Sentence Embedding and Fine-tuning, *H. Isotani, H. Washizaki, Y. Fukazawa, T. Nomoto, S. Ouji and S. Saito,* [H. Isotani and Saito, 2021]^{F};
- Sine-Cosine Algorithm for Software Fault Prediction, *T. Sharma and O. P. Sangwan,* [Sharma and Sangwan, 2021]^{F};
- Which bugs are missed in code reviews: An empirical study on SmartSHARK dataset, *F. Khoshnoud, A. R. Nasab, Z. Toudeji and A. Sami,* [F. Khoshnoud and Sami, 2022]^{J};
- Beyond Duplicates: Towards Understanding and Predicting Link Types in Issue Tracking Systems, *C. M. Luders, A. Bouraffa and W. Maalej,* [C. M. Lders and Maalej, 2022]^{J};

A.13 Traces, links, and context

Traces, links, and context all refer to the connections between code and its broader environment. For example, error and log messages which are printed out, inline popups in an IDE which provide the documentation to a third party API call, and so on. Essentially, this category has to do with the aspects involved with bridging between the code and everything else.

Tools

- Context-Aware Software Documentation, *E. Aghajani,* [Aghajani, 2018];
- Automatic Traceability Maintenance via Machine Learning Classification, *C. Mills, J. Escobar-Avila and S. Haiduc,* [C. Mills and Haiduc, 2018]^{F};
- Ticket Tagger: Machine Learning Driven Issue Classification, *R. Kallis, A. Di Sorbo, G. Canfora and S. Panichella,* [R. Kallis and Panichella, 2019]$^{F\,J}$;
- Improving the Effectiveness of Traceability Link Recovery using Hierarchical Bayesian Networks, *K. Moran et al.,* [Moran et al., 2020]^{F};
- Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models, *J. Lin, Y. Liu, Q. Zeng, M. Jiang and J. Cleland-Huang,* [J. Lin and Cleland-Huang, 2021]^{F};
- Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data, *P. R. Mazrae, M. Izadi and A. Heydarnoori,* [P. R. Mazrae and Heydarnoori, 2021]^{F};
- Online Summarizing Alerts through Semantic and Behavior Information, *J. Chen, P. Wang and W. Wang,* [J. Chen and Wang, 2022]^{F};

Psychology / Social Science

- Latent Patterns in Activities: A Field Study of How Developers Manage Context, *S. Chattopadhyay, N. Nelson, Y. Ramirez Gonzalez, A. Amelia Leon, R. Pandita and A. Sarma,* [S. Chattopadhyay and Sarma, 2019];
- Need for Tweet: How Open Source Developers Talk About Their GitHub Work on Twitter *Fang, Hongbo and Klug, Daniel and Lamba, Hemank and Herbsleb, James and Vasilescu, Bogdan* [Fang et al., 2020]^{J};

Analysis / Broader Studies

- Predicting Future Developer Behavior in the IDE Using Topic Models, *K. Damevski, H. Chen, D. C. Shepherd, N. A. Kraft and L. Pollock,* [K. Damevski and Pollock, 2018];

- Revisiting "Programmers' Build Errors" in the Visual Studio Context: A Replication Study Using IDE Interaction Traces *Rabbani, Noam and Harvey, Michael S. and Saquif, Sadnan and Gallaba, Keheliya and McIntosh, Shane* [Rabbani et al., 2018]^{IJ};
- Does the Propagation of Artifact Changes Across Tasks Reflect Work Dependencies?, *C. Mayr-Dorn and A. Egyed,* [Mayr-Dorn and Egyed, 2018];
- Why are Features Deprecated? An Investigation Into the Motivation Behind Deprecation, *A. A. Sawant, G. Huang, G. Vilen, S. Stojkovski and A. Bacchelli,* [A. A. Sawant and Bacchelli, 2018b]^{F};
- Studying the Dialogue Between Users and Developers of Free Apps in the Google Play Store, *S. Hassan, C. Tantithamthavorn, C. -P. Bezemer and A. E. Hassan,* [S. Hassan and Hassan, 2018]^{J};
- 9.6 Million Links in Source Code Comments: Purpose, Evolution, and Decay, *H. Hata, C. Treude, R. G. Kula and T. Ishio,* [H. Hata and Ishio, 2019]^{J};
- Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery, *C. Mills, J. Escobar-Avila, A. Bhattacharya, G. Kondyukov, S. Chakraborty and S. Haiduc,* [C. Mills and Haiduc, 2019]^{F};
- Leveraging Intermediate Artifacts to Improve Automated Trace Link Retrieval, *A. D. Rodriguez, J. Cleland-Huang and D. Falessi,* [A. D. Rodriguez and Falessi, 2021]^{IJ};
- Log-based Anomaly Detection with Deep Learning: How Far Are We?, *V. -H. Le and H. Zhang,* [Le and Zhang, 2022]^{FJ};
- Beyond Duplicates: Towards Understanding and Predicting Link Types in Issue Tracking Systems, *C. M. Luders, A. Bouraffa and W. Maalej,* [C. M. Lders and Maalej, 2022]^{J};

Curated Datasets

- A Graph-Based Dataset of Commit History of Real-World Android Apps *Geiger, Franz-Xaver and Malavolta, Ivano and Pascarella, Luca and Palomba, Fabio and Di Nucci, Dario and Bacchelli, Alberto* [Geiger et al., 2018]^{J};
- A Traceability Dataset for Open Source Systems, *M. Hammoudi, C. Mayr-Dorn, A. Mashkoor and A. Egyed,* [M. Hammoudi and Egyed, 2021]^{J};
- Apache Software Foundation Incubator Project Sustainability Dataset, *L. Yin, Z. Zhang, Q. Xuan and V. Filkov,* [L. Yin and Filkov, 2021]^{J};

A.14 Deprecation

Over time code becomes obsolete. This section is all about the (usually deliberate) sunsetting of code or projects which aren't actively used anymore.

Tools

- Can Automated Impact Analysis Techniques Help Predict Decaying Modules?, *N. Sae-Lim, S. Hayashi and M. Saeki,* [N. Sae-Lim and Saeki, 2019];

Psychology / Social Science

- Understanding Developers' Needs on Deprecation as a Language Feature, *A. A. Sawant, M. Aniche, A. van Deursen and A. Bacchelli,* [A. A. Sawant and Bacchelli, 2018a];

Analysis / Broader Studies

- Why are Features Deprecated? An Investigation Into the Motivation Behind Deprecation, *A. A. Sawant, G. Huang, G. Vilen, S. Stojkovski and A. Bacchelli,* [A. A. Sawant and Bacchelli, 2018b]^{F};

A.15 Whole project / repository aspects and status

This topic incorporates aspects of the software development process from a 30,000 foot view. It often incorporates metrics or aspects which project managers or other “bosses” are interested in and responsible for.

Tools

- Towards Automatically Identifying Paid Open Source Developers *Claes, Maëlick and Mäntylä, Mika and Kuutila, Miikka and Farooq, Umar* [Claes et al., 2018]$^{\mathrm{F\,J}}$;
- Interpretation-Enabled Software Reuse Detection Based on a Multi-level Birthmark Model, *X. Xu, Q. Zheng, Z. Yan, M. Fan, A. Jia and T. Liu,* [X. Xu and Liu, 2021]$^{\mathrm{G}}$;
- CHAMP: Characterizing Undesired App Behaviors from User Comments Based on Market Policies, *Y. Hu et al.,* [Hu et al., 2021]$^{\mathrm{F\,J}}$;
- Online Summarizing Alerts through Semantic and Behavior Information, *J. Chen, P. Wang and W. Wang,* [J. Chen and Wang, 2022]$^{\mathrm{F}}$;
- BotHunter: An Approach to Detect Software Bots in GitHub, *A. Abdellatif, M. Wessel, I. Steinmacher, M. A. Gerosa and E. Shihab,* [A. Abdellatif and Shihab, 2022]$^{\mathrm{A\,F\,J}}$;
- Bot Detection in GitHub Repositories, *N. Chidambaram and P. R. Mazrae,* [Chidambaram and Mazrae, 2022]$^{\mathrm{A\,F\,J}}$;
- Tooling for Time- and Space-efficient git Repository Mining, *F. Heseding, W. Scheibel and J. Dollner,* [F. Heseding and Dllner, 2022];
- Automated Detection of Password Leakage from Public GitHub Repositories, *R. Feng, Z. Yan, S. Peng and Y. Zhang,* [R. Feng and Zhang, 2022]$^{\mathrm{F\,G\,J}}$;
- Starting the InnerSource Journey: Key Goals and Metrics to Measure Collaboration, *D. Izquierdo-Cortazar, J. Alonso-Gutierrez, A. Perez Garcia-Plaza, G. Robles and J. M. Gonzalez-Barahona,* [D. Izquierdo-Cortzar and Gonzlez-Barahona, 2022]$^{\mathrm{I}}$;
- SoCCMiner: A Source Code-Comments and Comment-Context Miner, *M. Sridharan, M. Mantyla, M. Claes and L. Rantala,* [M. Sridharan and Rantala, 2022];

Psychology / Social Science

- Embracing Technical Debt, from a Startup Company Perspective, *T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe and J. Bosch,* [T. Besker and Bosch, 2018]$^{\mathrm{I\,K}}$;
- Privacy by Designers: Software Developers’ Privacy Mindset, *I. Hadar et al.,* [Hadar et al., 2018]$^{\mathrm{G}}$;
- Latent Patterns in Activities: A Field Study of How Developers Manage Context, *S. Chattopadhyay, N. Nelson, Y. Ramirez Gonzalez, A. Amelia Leon, R. Pandita and A. Sarma,* [S. Chattopadhyay and Sarma, 2019];
- The Product Backlog, *T. Sedano, P. Ralph and C. Peraire,* [T. Sedano and Praire, 2019]$^{\mathrm{I}}$;
- EmoD: An End-to-End Approach for Investigating Emotion Dynamics in Software Development, *K. P. Neupane, K. Cheung and Y. Wang,* [K. P. Neupane and Wang, 2019]$^{\mathrm{F}}$;
- How Practitioners Perceive Coding Proficiency, *X. Xia, Z. Wan, P. S. Kochhar and D. Lo,* [X. Xia and Lo, 2019];
- Do as I Do, Not as I Say: Do Contribution Guidelines Match the GitHub Contribution Process?, *O. Elazhary, M. -A. Storey, N. Ernst and A. Zaidman,* [O. Elazhary and Zaidman, 2019]$^{\mathrm{J}}$;
- A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub *Golubev, Yaroslav and Eliseeva, Maria and Povarov, Nikita and Bryksin, Timofey* [Golubev et al., 2020]$^{\mathrm{G\,I\,J}}$;
- How Software Practitioners Use Informal Local Meetups to Share Software Engineering Knowledge, *C. Ingram and A. Drachen,* [Ingram and Drachen, 2020];
- Analysis of Non-Discrimination Policies in the Sharing Economy, *M. Tushev, F. Ebrahimi and A. Mahmoud,* [M. Tushev and Mahmoud, 2021]$^{\mathrm{B}}$;
- Playing Planning Poker in Crowds: Human Computation of Software Effort Estimates, *M. Alhamed and T. Storer,* [Alhamed and Storer, 2021]$^{\mathrm{I\,J}}$;

- What Makes a Great Maintainer of Open Source Projects?, *E. Dias, P. Meirelles, F. Castor, I. Steinmacher, I. Wiese and G. Pinto,* [E. Dias and Pinto, 2021]^{J};
- Geographic Diversity in Public Code Contributions: An Exploratory Large-Scale Study Over 50 Years, *D. Rossi and S. Zacchiroli,* [Rossi and Zacchiroli, 2022]^{BCJ};
- This Is Damn Slick!" Estimating the Impact of Tweets on Open Source Project Popularity and New Contributors, *H. Fang, H. Lamba, J. Herbsleb and B. Vasilescu,* [H. Fang and Vasilescu, 2022]^{J};

Analysis / Broader Studies

- The Evolution of Requirements Practices in Software Startups, *C. Gralha, D. Damian, A. Wasserman, M. Goulao and J. Araujo,* [C. Gralha and Arajo, 2018]^{K};
- A Qualitative Study of Variability Management of Control Software for Industrial Automation Systems, *J. Fischer, S. Bougouffa, A. Schlie, I. Schaefer and B. Vogel-Heuser,* [J. Fischer and Vogel-Heuser, 2018];
- A Design Structure Matrix Approach for Measuring Co-Change-Modularity of Software Products *Benkoczi, Robert and Gaur, Daya and Hossain, Shahadat and Khan, Muhammad A.* [Benkoczi et al., 2018];
- Adapting Neural Text Classification for Improved Software Categorization, *A. LeClair, Z. Eberhart and C. McMillan,* [A. LeClair and McMillan, 2018]^{F};
- How to Not Get Rich: An Empirical Study of Donations in Op***n $our***e, *C. Overney, J. Meinicke, C. Kastner and B. Vasilescu,* [C. Overney and Vasilescu, 2020]^{J};
- What Constitutes Software? An Empirical, Descriptive Study of Artifacts *Pfeiffer, Rolf-Helge* [Pfeiffer, 2020];
- Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, *G. L. Scoccia, P. Migliarini and M. Autili,* [G. L. Scoccia and Autili, 2021]^{J};
- Which contributions count? Analysis of attribution in open source, *J. -G. Young, A. Casari, K. McLaughlin, M. Z. Trujillo, L. Hebert-Dufresne and J. P. Bagrow,* [J. G. Young and Bagrow, 2021]^{J};
- An Empirical Study on the Survival Rate of GitHub Projects, *A. Ait, J. L. C. Izquierdo and J. Cabot,* [A. Ait and Cabot, 2022]^{J};
- GitRank: A Framework to Rank GitHub Repositories, *N. Hasabnis,* [Hasabnis, 2022]^{F};
- Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set, *C. Gote, P. Mavrodiev, F. Schweitzer and I. Scholtes,* [C. Gote and Scholtes, 2022]^{J};
- The Art and Practice of Data Science Pipelines: A Comprehensive Study of Data Science Pipelines In Theory, In-The-Small, and In-The-Large, *S. Biswas, M. Wardat and H. Rajan,* [S. Biswas and Rajan, 2022]^{F};

Curated Datasets

- Threats of Aggregating Software Repository Data *M. P. Robillard, M. Nassif and S. McIntosh,* [M. P. Robillard and McIntosh, 2018]^{J};
- A Graph-Based Dataset of Commit History of Real-World Android Apps *Geiger, Franz-Xaver and Malavolta, Ivano and Pascarella, Luca and Palomba, Fabio and Di Nucci, Dario and Bacchelli, Alberto* [Geiger et al., 2018]^{J};
- Apache Software Foundation Incubator Project Sustainability Dataset, *L. Yin, Z. Zhang, Q. Xuan and V. Filkov,* [L. Yin and Filkov, 2021]^{J};
- A Large-scale Dataset of (Open Source) License Text Variants, *S. Zacchiroli,* [Zacchiroli, 2022]^{J};
- A Versatile Dataset of Agile Open Source Software Projects, *V. Tawosi, A. Al-Subaihin, R. Moussa and F. Sarro,* [V. Tawosi and Sarro, 2022]^{J};

A.16 Human and team dynamics

Software engineering is technical, but also very human. This topic explores the human element of the engineering process.

Tools

- Who's This? Developer Identification Using IDE Event Data *Wilkie, John and Halabi, Ziad Al and Karaoglu, Alperen and Liao, Jiafeng and Ndungu, George and Ragkhitwetsagul, Chaiyong and Paixao, Matheus and Krinke, Jens* [Wilkie et al., 2018]^{F};
- Developers' Game: A Preliminary Study Concerning a Tool for Automated Developers Assessment, *W. Fracz and J. Dajda,* [Frcz and Dajda, 2018];
- Towards Automatically Identifying Paid Open Source Developers *Claes, Maëlick and Mäntylä, Mika and Kuutila, Miikka and Farooq, Umar* [Claes et al., 2018]^{FJ};
- Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network, *L. Shi, M. Xing, M. Li, Y. Wang, S. Li and Q. Wang,* [L. Shi and Wang, 2020]^{F};
- Robin: A Voice Controlled Virtual Teammate for Software Developers and Teams, *B. da Silva, C. Hebert, A. Rawka and S. Sereesathien,* [B. da Silva and Sereesathien, 2020]^{AF};
- CHAMP: Characterizing Undesired App Behaviors from User Comments Based on Market Policies, *Y. Hu et al.,* [Hu et al., 2021]^{FJ};
- Representation of Developer Expertise in Open Source Software, *T. Dey, A. Karnauch and A. Mockus,* [T. Dey and Mockus, 2021]^{FJ};

Psychology / Social Science

- Embracing Technical Debt, from a Startup Company Perspective, *T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe and J. Bosch,* [T. Besker and Bosch, 2018]^{IK};
- Communicative Intention in Code Review Questions, *F. Ebert, F. Castor, N. Novielli and A. Serebrenik,* [F. Ebert and Serebrenik, 2018];
- Empirical Study on the Relationship between Developer's Working Habits and Efficiency *Rodriguez, Ariel and Tanaka, Fumiya and Kamei, Yasutaka* [Rodriguez et al., 2018];
- Do Programmers Work at Night or During the Weekend?, *M. Claes, M. V. Mantyla, M. Kuutila and B. Adams,* [M. Claes and Adams, 2018]^{J};
- How Modern News Aggregators Help Development Communities Shape and Share Knowledge, *M. Aniche et al.,* [Aniche et al., 2018]^{J};
- What Makes a Great Manager of Software Engineers?, *E. Kalliamvakou, C. Bird, T. Zimmermann, A. Begel, R. DeLine and D. M. German,* [E. Kalliamvakou and German, 2018]^{I};
- Older Adults and Hackathons: A Qualitative Study, *W. Kopec, B. Balcerzak, R. Nielek, G. Kowalik, A. Wierzbicki and F. Casati,* [W. Kopec and Casati, 2018]^{J};
- Data Scientists in Software Teams: State of the Art and Challenges, *M. Kim, T. Zimmermann, R. DeLine and A. Begel,* [M. Kim and Begel, 2018]^{I};
- Team Maturity in Agile Software Development: The Impact on Productivity, *S. L. Ramirez-Mora and H. Oktaba,* [Ramirez-Mora and Oktaba, 2018];
- How the Experience of Development Teams Relates to Assertion Density of Test Classes, *G. Catolino, F. Palomba, A. Zaidman and F. Ferrucci,* [G. Catolino and Ferrucci, 2019];
- Latent Patterns in Activities: A Field Study of How Developers Manage Context, *S. Chattopadhyay, N. Nelson, Y. Ramirez Gonzalez, A. Amelia Leon, R. Pandita and A. Sarma,* [S. Chattopadhyay and Sarma, 2019];
- The Product Backlog, *T. Sedano, P. Ralph and C. Peraire,* [T. Sedano and Praire, 2019]^{I};
- How Practitioners Perceive Coding Proficiency, *X. Xia, Z. Wan, P. S. Kochhar and D. Lo,* [X. Xia and Lo, 2019];
- Investigating the Effects of Gender Bias on GitHub, *N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai and E. Murphy-Hill,* [N. Imtiaz and Murphy-Hill, 2019]^{DJ};
- Socio-Technical Work-Rate Increase Associates With Changes in Work Patterns in Online Projects, *F. Sarker, B. Vasilescu, K. Blincoe and V. Filkov,* [F. Sarker and Filkov, 2019]^{J};
- Do as I Do, Not as I Say: Do Contribution Guidelines Match the GitHub Contribution Process?, *O. Elazhary, M. -A. Storey, N. Ernst and A. Zaidman,* [O. Elazhary and Zaidman, 2019]^{J};
- FLOSS Participants' Perceptions About Gender and Inclusiveness: A Survey, *A. Lee and J. C. Carver,* [Lee and Carver, 2019]^{DJ};

- Going Farther Together: The Impact of Social Capital on Sustained Participation in Open Source, *H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik and B. Vasilescu,* [H. S. Qiu and Vasilescu, 2019]^{DJ};
- Why Do Episodic Volunteers Stay in FLOSS Communities?, *A. Barcomb, K. -J. Stol, D. Riehle and B. Fitzgerald,* [A. Barcomb and Fitzgerald, 2019]^{J};
- A Study of Potential Code Borrowing and License Violations in Java Projects on GitHub *Golubev, Yaroslav and Eliseeva, Maria and Povarov, Nikita and Bryksin, Timofey* [Golubev et al., 2020]^{GIJ};
- Need for Tweet: How Open Source Developers Talk About Their GitHub Work on Twitter *Fang, Hongbo and Klug, Daniel and Lamba, Hemank and Herbsleb, James and Vasilescu, Bogdan* [Fang et al., 2020]^{J};
- How Software Practitioners Use Informal Local Meetups to Share Software Engineering Knowledge, *C. Ingram and A. Drachen,* [Ingram and Drachen, 2020];
- A Tale from the Trenches: Cognitive Biases and Software Development, *S. Chattopadhyay et al.,* [Chattopadhyay et al., 2020]^{B};
- Engineering Gender-Inclusivity into Software: Ten Teams' Tales from the Trenches, *C. Hilderbrand et al.,* [Hilderbrand et al., 2020]^{D};
- Remote Pair Programming in Virtual Reality, *J. Dominic, B. Tubre, C. Ritter, J. Houser, C. Smith and P. Rodeghero,* [J. Dominic and Rodeghero, 2020];
- Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented?, *J. Pasuksmit, P. Thongtanunam and S. Karunasekera,* [J. Pasuksmit and Karunasekera, 2021];
- Mining DEV for social and technical insights about software development, *M. Papoutsoglou, J. Wachs and G. M. Kapitsaki,* [M. Papoutsoglou and Kapitsaki, 2021]^{J};
- Ignorance and Prejudice" in Software Fairness, *J. M. Zhang and M. Harman,* [Zhang and Harman, 2021]^{F};
- Playing Planning Poker in Crowds: Human Computation of Software Effort Estimates, *M. Alhamed and T. Storer,* [Alhamed and Storer, 2021]^{IJ};
- How Was Your Weekend?" Software Development Teams Working From Home During COVID-19, *C. Miller, P. Rodeghero, M. -A. Storey, D. Ford and T. Zimmermann,* [C. Miller and Zimmermann, 2021]^{I};
- How Gamification Affects Software Developers: Cautionary Evidence from a Natural Experiment on GitHub, *L. Moldon, M. Strohmaier and J. Wachs,* [L. Moldon and Wachs, 2021]^{J};
- Clustering, Separation, and Connection: A Tale of Three Characteristics, *S. Datta, A. Mysore, H. Wira and S. Sarkar,* [S. Datta and Sarkar, 2021]^{E};
- What Makes a Great Maintainer of Open Source Projects?, *E. Dias, P. Meirelles, F. Castor, I. Steinmacher, I. Wiese and G. Pinto,* [E. Dias and Pinto, 2021]^{J};
- This Is Damn Slick!" Estimating the Impact of Tweets on Open Source Project Popularity and New Contributors, *H. Fang, H. Lamba, J. Herbsleb and B. Vasilescu,* [H. Fang and Vasilescu, 2022]^{J};
- Hashing It Out: A Survey of Programmers' Cannabis Usage, Perception, and Motivation, *M. Endres, K. Boehnke and W. Weimer,* [M. Endres and Weimer, 2022];
- What Makes Effective Leadership in Agile Software Development Teams?, *L. Gren and P. Ralph,* [Gren and Ralph, 2022];
- Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process, *N. Nahar, S. Zhou, G. Lewis and C. Kastner,* [N. Nahar and Kstner, 2022]^{F};
- An Exploratory Study of Productivity Perceptions in Software Teams, *A. Ruvimova et al.,* [Ruvimova et al., 2022];
- A Grounded Theory of Coordination in Remote-First and Hybrid Software Teams, *R. E. de Souza Santos and P. Ralph,* [de Souza Santos and Ralph, 2022];

Analysis / Broader Studies

- Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub, *G. L. Scoccia, P. Migliarini and M. Autili,* [G. L. Scoccia and Autili, 2021]^{J};
- Which contributions count? Analysis of attribution in open source, *J. -G. Young, A. Casari, K. McLaughlin, M. Z. Trujillo, L. Hebert-Dufresne and J. P. Bagrow,* [J. G. Young and Bagrow, 2021]^{J};

- The Shifting Sands of Motivation: Revisiting What Drives Contributors in Open Source, *M. Gerosa et al.,* [Gerosa et al., 2021]^{J};
- Report From The Trenches A Case Study In Modernizing Software Development Practices, *M. H. Houekpetodji, N. Anquetil, S. Ducasse, F. Djareddir and J. Sudich,* [M. H. Houekpetodji and Sudich, 2021]^{I};
- DISCO: A Dataset of Discord Chat Conversations for Software Engineering Research, *K. M. Subash, L. P. Kumar, S. L. Vadlamani, P. Chatterjee and O. Baysal,* [K. M. Subash and Baysal, 2022]^{A};

A.17 Machine learning foundations

In addition to the cross cutting theme of applied machine learning, which interweaves itself with many tools, there are several works related to research in the domain-specific foundational algorithms needed for machine learning to later be successful in the software engineering tool space. This topic investigates these works.

Tools

- TUNA: TUning Naturalness-Based Analysis, *M. Jimenez, C. Maxime, Y. Le Traon and M. Papadakis,* [M. Jimenez and Papadakis, 2018b]^{F};
- Natural Language or Not (NLON): A Package for Software Engineering Text Analysis Pipeline *Mäntylä, Mika V. and Calefato, Fabio and Claes, Maelick* [Mantyla et al., 2018]^{F};
- Word Embeddings for the Software Engineering Domain *Efstathiou, Vasiliki and Chatzilenas, Christos and Spinellis, Diomidis* [Efstathiou et al., 2018]^{F};
- Learning to Identify Security-Related Issues Using Convolutional Neural Networks, *D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cardenas, D. Poshyvanyk and C. Shenefiel,* [D. N. Palacio and Shenefiel, 2019]^{F};
- Import2vec Learning Embeddings for Software Libraries *Theeten, Bart and Vandeputte, Frederik and Van Cutsem, Tom* [Theeten et al., 2019]^{F};
- Retrieval-based Neural Source Code Summarization, *J. Zhang, X. Wang, H. Zhang, H. Sun and X. Liu,* [J. Zhang and Liu, 2020]^{F};
- Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements, *M. Osama, A. Zaki-Ismail, M. Abdelrazek, J. Grundy and A. Ibrahim,* [M. Osama and Ibrahim, 2020]^{F};
- Embedding Java Classes with Code2vec: Improvements from Variable Obfuscation *Compton, Rhys and Frank, Eibe and Patros, Panos and Koay, Abigail* [Compton et al., 2020]^{F};
- CC2Vec: Distributed Representations of Code Changes, *T. Hoang, H. J. Kang, D. Lo and J. Lawall,* [T. Hoang and Lawall, 2020]^{F};
- POSIT: Simultaneously Tagging Natural and Programming Languages, *P. -P. Partachi, S. K. Dash, C. Treude and E. T. Barr,* [P. P. Prachi and Barr, 2020]^{F};
- Repo2Vec: A Comprehensive Embedding Approach for Determining Repository Similarity, *M. O. F. Rokon, P. Yan, R. Islam and M. Faloutsos,* [M. O. F. Rokon and Faloutsos, 2021]$^{F\,J}$;
- Fast Changeset-based Bug Localization with BERT, *A. Ciborowska and K. Damevski,* [Ciborowska and Damevski, 2022]^{F};
- GraphCode2Vec: Generic Code Embedding via Lexical and Program Dependence Analyses, *W. Ma et al.,* [Ma et al., 2022]^{F};
- Manas: Mining Software Repositories to Assist AutoML, *G. Nguyen, M. J. Islam, R. Pan and H. Rajan,* [G. Nguyen and Rajan, 2022]^{F};

Psychology / Social Science

- Ignorance and Prejudice" in Software Fairness, *J. M. Zhang and M. Harman,* [Zhang and Harman, 2021]^{F};

Analysis / Broader Studies

- On the Impact of Tokenizer and Parameters on N-Gram Based Code Analysis, *M. Jimenez, C. Maxime, Y. Le Traon and M. Papadakis,* [M. Jimenez and Papadakis, 2018a]^{F};
- Semantic Source Code Models Using Identifier Embeddings *Efstathiou, Vasiliki and Spinellis, Diomidis* [Efstathiou and Spinellis, 2019]^{FJ};
- Assessing Generalizability of CodeBERT, *X. Zhou, D. Han and D. Lo,* [X. Zhou and Lo, 2021]^{F};
- Log-based Anomaly Detection with Deep Learning: How Far Are We?, *V. -H. Le and H. Zhang,* [Le and Zhang, 2022]^{FJ};
- Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes, *J. Gong and T. Chen,* [Gong and Chen, 2022]^{F};
- What Do They Capture? - A Structural Analysis of Pre-Trained Language Models for Source Code, *Y. Wan, W. Zhao, H. Zhang, Y. Sui, G. Xu and H. Jin,* [Y. Wan and Jin, 2022]^{EF};
- Multilingual training for Software Engineering, *T. Ahmed and P. Devanbu,* [Ahmed and Devanbu, 2022]^{F};
- The Art and Practice of Data Science Pipelines: A Comprehensive Study of Data Science Pipelines In Theory, In-The-Small, and In-The-Large, *S. Biswas, M. Wardat and H. Rajan,* [S. Biswas and Rajan, 2022]^{F};

Curated Datasets

- A Time Series-Based Dataset of Open-Source Software Evolution, *B. L. Sousa, M. A. S. Bigonha, K. A. M. Ferreira and G. C. Franco,* [B. L. Sousa and Franco, 2022]^{J};

A.18 Software Engineering / Repository Mining Research Meta Analysis

This topic covers research on research, which includes things like literature reviews, papers on methodology, and so forth.

Tools

- Sampling Projects in GitHub for MSR Studies, *O. Dabic, E. Aghajani and G. Bavota,* [O. Dabic and Bavota, 2021]^{J};
- SoCCMiner: A Source Code-Comments and Comment-Context Miner, *M. Sridharan, M. Mantyla, M. Claes and L. Rantala,* [M. Sridharan and Rantala, 2022];

Psychology / Social Science

- A Study on the Prevalence of Human Values in Software Engineering Publications, 2015 - 2018, *H. Perera et al.,* [Perera et al., 2020]^{B};
- Social Science Theories in Software Engineering Research, *T. Lorey, P. Ralph and M. Felderer,* [T. Lorey and Felderer, 2022];

Analysis / Broader Studies

- Synthesizing Qualitative Research in Software Engineering: A Critical Review, *X. Huang, H. Zhang, X. Zhou, M. Ali Babar and S. Yang,* [X. Huang and Yang, 2018];
- What Constitutes Software? An Empirical, Descriptive Study of Artifacts *Pfeiffer, Rolf-Helge* [Pfeiffer, 2020];
- Ethical Mining: A Case Study on MSR Mining Challenges *Gold, Nicolas E. and Krinke, Jens* [Gold and Krinke, 2020]^{B};
- Empirical Standards for Repository Mining, *P. Chatterjee, T. Sharma and P. Ralph,* [P. Chatterjee and Ralph, 2022];

Curated Datasets

- The Unsolvable Problem or the Unheard Answer? A Dataset of 24,669 Open-Source Software Conference Talks, *K. Truong, C. Miller, B. Vasilescu and C. Kastner*, [K. Truong and Kstner, 2022]^{J};

B Input-Output Types of Machine Learning Tools

Table 1: Primary input and output types of all machine learning model driven tools examined in this report.

Title	**Citation**	**Input**	**Output**
500+ Times Faster than Deep Learning: A Case Study Exploring Faster Methods for Text Mining Stackoverflow	[Majumder et al., 2018]	Text (Stack Overflow)	Classification
A Gold Standard for Emotion Annotation in Stack Overflow	[Novielli et al., 2018]	Text (Stack Overflow)	Classification
A Practical Approach to the Automatic Classification of Security-Relevant Commits	[Sabetta and Bezzi, 2018]	Text (Commits)	Classification
A Simple NLP-Based Approach to Support Onboarding and Retention in Open Source Communities	[C. Stanik and Maalej, 2018]	Text (Issues)	Classification
Achieving Reliable Sentiment Analysis in the Software Engineering Domain using BERT	[E. Biswas and Vijay-Shanker, 2020]	Text (Stack Overflow)	Classification
Applying CodeBERT for Automated Program Repair of Java Simple Bugs	[Mashhadi and Hemmati, 2021]	Text (Bug-ridden code)	Classification
Attention-based model for predicting question relatedness on Stack Overflow	[J. Pei and Guan, 2021]	Text (Stack Overflow)	Classification
Automated Assertion Generation via Information Retrieval and Its Integration with Deep learning	[Yu et al., 2022]	Text (Code)	Text (Code)
Automated Characterization of Software Vulnerabilities	[D. Gonzalez and Mirakhorli, 2019]	Text (Jargon)	Classification
Automated Detection of Password Leakage from Public GitHub Repositories	[R. Feng and Zhang, 2022]	Text (GitHub/Repo artifacts)	Classification
Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data	[P. R. Mazrae and Heydarnoori, 2021]	Text (Issues and Commits); Numbers	Classification
Automatic Extraction of Opinion-Based Q&A from Online Developer Chats	[P. Chatterjee and Pollock, 2021]	Text (Jargon)	Classification
Automatic Identification of Rollback Edit with Reasons in Stack Overflow Q&A Site	[S. Mondal and Roy, 2020]	Text (Jargon)	Classification
Automatic Traceability Maintenance via Machine Learning Classification	[C. Mills and Haiduc, 2018]	Text (GitHub/Repo artifacts)	Classification
Automatic Web Testing Using Curiosity-Driven Reinforcement Learning	[Zheng et al., 2021]	Text (Code)	Complex – See Paper; Classification

Table 1: Primary input and output types of all machine learning model driven tools examined in this report.

Title	**Citation**	**Input**	**Output**
Automatically Generating Precise Oracles from Structured Natural Language Specifications	[Motwani and Brun, 2019]	Text (Jargon)	Text (Code)
Bot Detection in GitHub Repositories	[Chidambaram and Mazrae, 2022]	Text (GitHub/Repo artifacts)	Classification
BotHunter: An Approach to Detect Software Bots in GitHub	[A. Abdellatif and Shihab, 2022]	Text (GitHub/Repo artifacts)	Classification
BugListener: Identifying and Synthesizing Bug Reports from Collaborative Live Chats	[Shi et al., 2022]	Text (Jargon)	Text (Jargon); Classification
Can I Solve It? Identifying APIs Required to Complete OSS Tasks	[F. Santos and Gerosa, 2021]	Text (GitHub/Repo artifacts)	Text (Code); Classification
Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews	[Guo and Singh, 2020]	Text (Natural Language)	Classification
CHAMP: Characterizing Undesired App Behaviors from User Comments Based on Market Policies	[Hu et al., 2021]	Text (Natural Language)	Classification
CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects	[Nayrolles and Hamou-Lhadj, 2018]	Text (Code)	Classification
DEAR: A Novel Deep Learning-based Approach for Automated Program Repair	[Y. Li and Nguyen, 2022]	Text (Code)	Text (Code)
Deep Learning Anti-Patterns from Code Metrics History	[A. Barbez and Guhneuc, 2019]	Text (Code)	Classification
Detecting and Characterizing Bots That Commit Code	[Dey et al., 2020]	Text (GitHub/Repo artifacts)	Classification
Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network	[L. Shi and Wang, 2020]	Text (Natural Language)	Text (Jargon)
Developer-Driven Code Smell Prioritization	[Pecorelli et al., 2020]	Text (Code)	Classification
Dialogue Management for Interactive API Search	[Eberhart and McMillan, 2021]	Text (Natural Language)	Text (Code)
Efficient Bug Triage For Industrial Environments	[Zhang, 2020]	Text (Jargon)	Classification
EmoD: An End-to-End Approach for Investigating Emotion Dynamics in Software Development	[K. P. Neupane and Wang, 2019]	Text (Jargon)	Classification
Ensemble Models for Neural Source Code Summarization of Subroutines	[A. LeClair and McMillan, 2021]	Text (Code)	Text (Natural Language)
Fast and Memory-Efficient Neural Code Completion	[A. Svyatkovskiy and Allamanis, 2021]	Text (Code)	Text (Code)

Table 1: Primary input and output types of all machine learning model driven tools examined in this report.

Title	**Citation**	**Input**	**Output**
Fast Changeset-based Bug Localization with BERT	[Ciborowska and Damevski, 2022]	Text (Code)	Complex – See Paper; Classification
FeaRS: Recommending Complete Android Method Implementations	[F. Wen and Bavota, 2021b]	Text (Code)	Text (Code)
FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation	[Dong et al., 2022]	Text (Code)	Text (Commits)
Graph Neural Network-based Vulnerability Predication	[Q. Feng and Hong, 2020]	Text (Code)	Classification
Graph-Based Fuzz Testing for Deep Learning Inference Engines	[W. Luo and Chen, 2021]	Text (Code); Complex – see paper	Complex – See Paper; Classification
HARP: Holistic Analysis for Refactoring Python-Based Analytics Programs	[W. Zhou and Shen, 2020]	Text (Code)	Complex – See Paper; Classification
How (Not) to Find Bugs: The Interplay Between Merge Conflicts, Co-Changes, and Bugs	[Amaral et al., 2020]	Text (Commits with Bugs)	Classification
How Android Developers Handle Evolution-induced API Compatibility Issues: A Large-scale Study	[Xia et al., 2020]	Text (Code)	Classification
Human, bot or both? A study on the capabilities of classification models on mixed accounts	[N. Cassee and Serebrenik, 2021]	Text (GitHub/Repo artifacts)	Classification
Improving Bug Triaging with High Confidence Predictions at Ericsson	[A. Sarkar and Bartalos, 2019]	Text (Jargon)	Classification
Improving the Effectiveness of Traceability Link Recovery using Hierarchical Bayesian Networks	[Moran et al., 2020]	Text (GitHub/Repo artifacts)	Classification
Incorporating Multiple Features to Predict Bug Fixing Time with Neural Networks	[W. Yuan and Liu, 2021]	Complex – see paper	Classification
Learning Off-By-One Mistakes: An Empirical Study	[H. Sellik and Aniche, 2021]	Text (Code)	Classification
Learning to Identify Security-Related Issues Using Convolutional Neural Networks	[D. N. Palacio and Shenefiel, 2019]	Text (Jargon)	Classification
Learning to Spot and Refactor Inconsistent Method Names	[Liu et al., 2019]	Text (Code)	Text (Code)
LineVD: Statement-level Vulnerability Detection using Graph Neural Networks	[D. Hin and Babar, 2022]	Text (Code)	Classification
Linking Source Code to Untangled Change Intents	[X. Liu and Ng, 2018]	Text (Code)	Classification
Manas: Mining Software Repositories to Assist AutoML	[G. Nguyen and Rajan, 2022]	Text (GitHub/Repo artifacts)	Complex – See Paper; Text (Code)

Table 1: Primary input and output types of all machine learning model driven tools examined in this report.

Title	Citation	Input	Output
Natural Language or Not (NLON): A Package for Software Engineering Text Analysis Pipeline	[Mantyla et al., 2018]	Text	Classification
On Learning Meaningful Code Changes Via Neural Machine Translation	[M. Tufano and Poshyvanyk, 2019]	Text (Pull Requests)	Text (Code)
On the Importance of Building High-quality Training Datasets for Neural Code Search	[Z. Sun and Li, 2022]	Text (GitHub/Repo artifacts)	Text (Code)
Online Summarizing Alerts through Semantic and Behavior Information	[J. Chen and Wang, 2022]	Text (Jargon)	Text (Natural Language)
Personalized Code Recommendation	[T. Nguyen and Nguyen, 2019a]	Text (Code)	Text (Code)
Planning for Untangling: Predicting the Difficulty of Merge Conflicts	[C. Brindescu and Sarma, 2020]	Text (GitHub/Repo artifacts)	Classification
POSIT: Simultaneously Tagging Natural and Programming Languages	[P. P. Prachi and Barr, 2020]	Text	Classification
Predicting Developers' IDE Commands with Machine Learning	[Bulmer et al., 2018]	Complex – see paper	Classification
Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques	[H. Alsolai and Nassar, 2018]	Numbers (Metrics from Code)	Regression
Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning	[S. Reddy and Sen, 2020]	Text (Code)	Complex – See Paper
Recommending Exception Handling Code	[T. Nguyen and Nguyen, 2019b]	Text (Code)	Classification
Recommending Good First Issues in GitHub OSS Projects	[W. Xiao and Zhou, 2022]	Text (GitHub/Repo artifacts)	Classification
Representation of Developer Expertise in Open Source Software	[T. Dey and Mockus, 2021]	Text (GitHub/Repo artifacts)	Complex – See Paper
Retrieval-based Neural Source Code Summarization	[J. Zhang and Liu, 2020]	Text (Code)	Text (Natural Language)
Robin: A Voice Controlled Virtual Teammate for Software Developers and Teams	[B. da Silva and Sereesathien, 2020]	Text (Natural Language)	Complex – See Paper
RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation	[Z. Li and Xu, 2022]	Text (Code)	Classification
SAPIENTML: Synthesizing Machine Learning Pipelines by Learning from Human-Written Solutions	[Saha et al., 2022]	Text (Natural Language)	Complex – See Paper
Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements	[M. Osama and Ibrahim, 2020]	Text (Natural Language)	Complex – See Paper
Senatus - A Fast and Accurate Code-to-Code Recommendation Engine	[F. Silavong and Otter, 2022]	Text (Code)	Text (Code)

Table 1: Primary input and output types of all machine learning model driven tools examined in this report.

Title	**Citation**	**Input**	**Output**
Siri, Write the Next Method	[F. Wen and Bavota, 2021a]	Text (Natural Language)	Text (Code)
SPT-Code: Sequence-to-Sequence Pre-Training for Learning Source Code Representations	[C. Niu and Luo, 2022]	Text (Code)	Text (Code)
Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction	[A. Trautsch and Grabowski, 2020]	Metrics from Code	Classification
Statistical Learning of API Fully Qualified Names in Code Snippets of Online Forums	[H. Phan and Nguyen, 2018]	Text (Jargon)	Text (Code)
Striking a Balance: Pruning False-Positives from Static Call Graphs	[A. Utture and Palsberg, 2022]	Text (Code)	Classification
Suggesting Natural Method Names to Check Name Consistencies	[S. Nguyen and Nguyen, 2020]	Text (Code)	Text (Natural Language), Text (Code)
Supporting Software Architecture Maintenance by Providing Task-Specific Recommendations	[M. Galster and Blincoe, 2019]	Text (GitHub/Repo artifacts)	Classification
Ticket Tagger: Machine Learning Driven Issue Classification	[R. Kallis and Panichella, 2019]	Text (Issues)	Classification
Toward Less Hidden Cost of Code Completion with Acceptance and Ranking Models	[J. Li and Tan, 2021]	Text (Code)	Text (Code)
Towards Automatically Identifying Paid Open Source Developers	[Claes et al., 2018]	Text (GitHub/Repo artifacts)	Classification
Towards Automating Code Review Activities	[R. Tufano and Bavota, 2021]	Text (GitHub/Repo artifacts)	Complex – See paper
Towards Reliable Agile Iterative Planning via Predicting Documentation Changes of Work Items	[J. Pasuksmit and Karunasekera, 2022]	Text (GitHub/Repo artifacts)	Complex – See paper
Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models	[J. Lin and Cleland-Huang, 2021]	Text (GitHub/Repo artifacts)	Classification
Type4Py: Practical Deep Similarity Learning-Based Type Inference for Python	[A. M. Mir and Gousios, 2022]	Text (Code)	Text (Code)
Using Pre-Trained Models to Boost Code Review Automation	[R. Tufano and Bavota, 2022]	Text (GitHub/Repo artifacts)	Classification
Where is Your App Frustrating Users?	[Y. Wang and Wang, 2022]	Text (Natural Language)	Complex – See Paper
Who's This? Developer Identification Using IDE Event Data	[Wilkie et al., 2018]	Complex – see paper	Classification
You Look so Different: Finding Structural Clones and Subclones in Java Source Code	[W. Amme and Schfer, 2021]	Text (Code)	Classification

References

[A. A. Sawant and Bacchelli, 2018a] A. A. Sawant, M. Aniche, A. v. D. and Bacchelli, A. (2018a). Understanding developers' needs on deprecation as a language feature. In *ICSE*.

[A. A. Sawant and Bacchelli, 2018b] A. A. Sawant, G. Huang, G. V. S. S. and Bacchelli, A. (2018b). Why are features deprecated? an investigation into the motivation behind deprecation. In *ICSME*.

[A. Abdellatif and Shihab, 2022] A. Abdellatif, M. Wessel, I. S. M. A. G. and Shihab, E. (2022). Bothunter: An approach to detect software bots in github. In *MSR*.

[A. Ait and Cabot, 2022] A. Ait, J. L. C. I. and Cabot, J. (2022). An empirical study on the survival rate of github projects. In *MSR*.

[A. Barbez and Guhneuc, 2019] A. Barbez, F. K. and Guhneuc, Y. G. (2019). Deep learning anti-patterns from code metrics history. In *ICSME*.

[A. Barcomb and Fitzgerald, 2019] A. Barcomb, K. J. Stol, D. R. and Fitzgerald, B. (2019). Why do episodic volunteers stay in floss communities? In *ICSE*.

[A. Chueshev and Ziadi, 2020] A. Chueshev, J. Lawall, R. B. and Ziadi, T. (2020). Expanding the number of reviewers in open-source projects by recommending appropriate developers. In *ICSME*.

[A. D. Rodriguez and Falessi, 2021] A. D. Rodriguez, J. C.-H. and Falessi, D. (2021). Leveraging intermediate artifacts to improve automated trace link retrieval. In *ICSME*.

[A. Foundjem and Adams, 2021] A. Foundjem, E. E. and Adams, B. (2021). Onboarding vs. diversity, productivity and quality empirical study of the openstack ecosystem. In *ICSE*.

[A. Galappaththi and Treude, 2022] A. Galappaththi, S. N. and Treude, C. (2022). Does this apply to me? an empirical study of technical context in stack overflow. In *MSR*.

[A. Head and Knight, 2018] A. Head, C. Sadowski, E. M.-H. and Knight, A. (2018). When not to comment: Questions and tradeoffs with api documentation for c++ projects. In *ICSE*.

[A. Ju and Herzig, 2021] A. Ju, H. Sajnani, S. K. and Herzig, K. (2021). A case study of onboarding in software teams: Tasks and strategies. In *ICSE*.

[A. LeClair and McMillan, 2021] A. LeClair, A. B. and McMillan, C. (2021). Ensemble models for neural source code summarization of subroutines. In *ICSME*.

[A. LeClair and McMillan, 2018] A. LeClair, Z. E. and McMillan, C. (2018). Adapting neural text classification for improved software categorization. In *ICSME*.

[A. M. Mir and Gousios, 2022] A. M. Mir, E. Latokinas, S. P. and Gousios, G. (2022). Type4py: Practical deep similarity learning-based type inference for python. In *ICSE*.

[A. Marques and Murphy, 2020] A. Marques, N. C. B. and Murphy, G. C. (2020). Characterizing task-relevant information in natural language software artifacts. In *ICSME*.

[A. Mastropaolo and Bavota, 2021] A. Mastropaolo, E. Aghajani, L. P. and Bavota, G. (2021). An empirical study on code comment completion. In *ICSME*.

[A. Peruma and Newman, 2021] A. Peruma, V. A. and Newman, C. D. (2021). Ideal: An open-source identifier name appraisal tool. In *ICSME*.

[A. Sarkar and Bartalos, 2019] A. Sarkar, P. C. R. and Bartalos, B. (2019). Improving bug triaging with high confidence predictions at ericsson. In *ICSME*.

[A. Svyatkovskiy and Allamanis, 2021] A. Svyatkovskiy, S. Lee, A. H. M. R. J. V. F. and Allamanis, M. (2021). Fast and memory-efficient neural code completion. In *MSR*.

[A. Trautsch and Grabowski, 2020] A. Trautsch, S. H. and Grabowski, J. (2020). Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction. In *ICSME*.

[A. Utture and Palsberg, 2022] A. Utture, S. Liu, C. G. K. and Palsberg, J. (2022). Striking a balance: Pruning false-positives from static call graphs. In *ICSE*.

[Abdellatif et al., 2020] Abdellatif, A., Costa, D., Badran, K., Abdalkareem, R., and Shihab, E. (2020). Challenges in chatbot development: A study of stack overflow posts. In *MSR*.

[Accioly et al., 2018] Accioly, P., Borba, P., Silva, L., and Cavalcanti, G. (2018). Analyzing conflict predictors in open-source java projects. In *MSR*.

[Aghajani, 2018] Aghajani, E. (2018). Context-aware software documentation. In *ICSME*.

[Aghajani et al., 2020] Aghajani, E., Nagy, C., Linares-Vasquez, M., Moreno, L., Bavota, G., Lanza, M., and Shepherd, D. C. (2020). Software documentation: The practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 590601, New York, NY, USA. Association for Computing Machinery.

[Aghajani et al., 2019] Aghajani, E., Nagy, C., Vega-Marquez, O. L., Linares-Vasquez, M., Moreno, L., Bavota, G., and Lanza, M. (2019). Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1199–1210.

[Agrawal and Menzies, 2018] Agrawal, A. and Menzies, T. (2018). Is "better data" better than "better data miners"? In *ICSE*.

[Ahmed and Devanbu, 2022] Ahmed, T. and Devanbu, P. (2022). Multilingual training for software engineering. In *ICSE*.

[Alhamed and Storer, 2021] Alhamed, M. and Storer, T. (2021). Playing planning poker in crowds: Human computation of software effort estimates. In *ICSE*.

[Alomar, 2019] Alomar, E. A. (2019). Towards better understanding developer perception of refactoring. In *ICSME*.

[Alqadi, 2019] Alqadi, B. S. (2019). The relationship between cognitive complexity and the probability of defects. In *ICSME*.

[Amaral et al., 2020] Amaral, L., Oliveira, M. C., Luz, W., Fortes, J., Bonifacio, R., Alencar, D., Monteiro, E., Pinto, G., and Lo, D. (2020). How (not) to find bugs: The interplay between merge conflicts, co-changes, and bugs. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 441–452.

[Amlekar et al., 2018] Amlekar, R., Gamboa, A. F. R., Gallaba, K., and McIntosh, S. (2018). Do software engineers use autocompletion features differently than other developers? In *MSR*.

[Aniche et al., 2018] Aniche, M., Treude, C., Steinmacher, I., Wiese, I., Pinto, G., Storey, M.-A., and Gerosa, M. A. (2018). How modern news aggregators help development communities shape and share knowledge. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 499–510.

[B. da Silva and Sereesathien, 2020] B. da Silva, C. Hebert, A. R. and Sereesathien, S. (2020). Robin: A voice controlled virtual teammate for software developers and teams. In *ICSME*.

[B. Hempel and Chugh, 2018] B. Hempel, J. Lubin, G. L. and Chugh, R. (2018). Deuce: A lightweight user interface for structured editing. In *ICSE*.

[B. Kou and Zhang, 2022] B. Kou, Y. Di, M. C. and Zhang, T. (2022). Sosum: A dataset of stack overflow post summaries. In *MSR*.

[B. L. Sousa and Franco, 2022] B. L. Sousa, M. A. S. Bigonha, K. A. M. F. and Franco, G. C. (2022). A time series-based dataset of open-source software evolution. In *MSR*.

[B. Lin and Lanza, 2019] B. Lin, F. Zampetti, G. B. M. D. P. and Lanza, M. (2019). Pattern-based mining of opinions in q&a websites. In *ICSE*.

[B. Lin and Oliveto, 2018] B. Lin, F. Zampetti, G. B. M. D. P. M. L. and Oliveto, R. (2018). Sentiment analysis for software engineering: How far can we go? In *ICSE*.

[B. Lin and Bavota, 2018] B. Lin, F. Zampetti, R. O. M. D. P. M. L. and Bavota, G. (2018). Two datasets for sentiment analysis in software engineering. In *ICSME*.

[Balachandran, 2020] Balachandran, V. (2020). On the need for automatic knowledge management in modern collaboration tools to improve software maintenance. In *ICSME*.

[Baltes et al., 2018] Baltes, S., Dumani, L., Treude, C., and Diehl, S. (2018). Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In *MSR*.

[Bellman et al., 2018] Bellman, C., Seet, A., and Baysal, O. (2018). Studying developer build issues and debugger usage via timeline analysis in visual studio ide. In *MSR*.

[Benkoczi et al., 2018] Benkoczi, R., Gaur, D., Hossain, S., and Khan, M. A. (2018). A design structure matrix approach for measuring co-change-modularity of software products. In *MSR*.

[Bernardo et al., 2018] Bernardo, J. a. H., da Costa, D. A., and Kulesza, U. (2018). Studying the impact of adopting continuous integration on the delivery time of pull requests. In *MSR*.

[Biswas et al., 2019] Biswas, E., Vijay-Shanker, K., and Pollock, L. (2019). Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In *MSR*.

[Bulmer et al., 2018] Bulmer, T., Montgomery, L., and Damian, D. (2018). Predicting developers' ide commands with machine learning. In *MSR*.

[C. Brindescu and Jensen, 2020] C. Brindescu, Y. Ramirez, A. S. and Jensen, C. (2020). Lifting the curtain on merge conflict resolution: A sensemaking perspective. In *ICSME*.

[C. Brindescu and Sarma, 2020] C. Brindescu, I. Ahmed, R. L. and Sarma, A. (2020). Planning for untangling: Predicting the difficulty of merge conflicts. In *ICSE*.

[C. Gote and Scholtes, 2022] C. Gote, P. Mavrodiev, F. S. and Scholtes, I. (2022). Big data = big insights? operationalising brooks' law in a massive github data set. In *ICSE*.

[C. Gralha and Arajo, 2018] C. Gralha, D. Damian, A. W. M. G. and Arajo, J. (2018). The evolution of requirements practices in software startups. In *ICSE*.

[C. M. Lders and Maalej, 2022] C. M. Lders, A. B. and Maalej, W. (2022). Beyond duplicates: Towards understanding and predicting link types in issue tracking systems. In *MSR*.

[C. Miller and Zimmermann, 2021] C. Miller, P. Rodeghero, M. A. S. D. F. and Zimmermann, T. (2021). How was your weekend?" software development teams working from home during covid-19. In *ICSE*.

[C. Mills and Haiduc, 2019] C. Mills, J. Escobar-Avila, A. B. G. K. S. C. and Haiduc, S. (2019). Tracing with less data: Active learning for classification-based traceability link recovery. In *ICSME*.

[C. Mills and Haiduc, 2018] C. Mills, J. E.-A. and Haiduc, S. (2018). Automatic traceability maintenance via machine learning classification. In *ICSME*.

[C. Niu and Luo, 2022] C. Niu, C. Li, V. N. J. G. L. H. and Luo, B. (2022). Spt-code: Sequence-to-sequence pre-training for learning source code representations. In *ICSE*.

[C. Overney and Vasilescu, 2020] C. Overney, J. Meinicke, C. K. and Vasilescu, B. (2020). How to not get rich: An empirical study of donations in opn $oure. In *ICSE*.

[C. Stanik and Maalej, 2018] C. Stanik, L. Montgomery, D. M. D. F. and Maalej, W. (2018). A simple nlp-based approach to support onboarding and retention in open source communities. In *ICSME*.

[C. Vendome and Poshyvanyk, 2018] C. Vendome, D. German, M. D. P. G. B. M. L.-V. and Poshyvanyk, D. (2018). To distribute or not to distribute? why licensing bugs matter. In *ICSE*.

[Chattopadhyay et al., 2020] Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., and Sarma, A. (2020). A tale from the trenches: Cognitive biases and software development. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 654–665.

[Chidambaram and Mazrae, 2022] Chidambaram, N. and Mazrae, P. R. (2022). Bot detection in github repositories. In *MSR*.

[Ciborowska and Damevski, 2022] Ciborowska, A. and Damevski, K. (2022). Fast changeset-based bug localization with bert. In *ICSE*.

[Claes et al., 2018] Claes, M., Mantyla, M., Kuutila, M., and Farooq, U. (2018). Towards automatically identifying paid open source developers. In *MSR*.

[Cohen and Consens, 2018] Cohen, E. and Consens, M. P. (2018). Large-scale analysis of the co-commit patterns of the active developers in github's top repositories. In *MSR*.

[Compton et al., 2020] Compton, R., Frank, E., Patros, P., and Koay, A. (2020). Embedding java classes with code2vec: Improvements from variable obfuscation. In *MSR*.

[D. Girardi and Lanubile, 2020] D. Girardi, N. Novielli, D. F. and Lanubile, F. (2020). Recognizing developers' emotions while programming. In *ICSE*.

[D. Gonzalez and Mirakhorli, 2019] D. Gonzalez, H. H. and Mirakhorli, M. (2019). Automated characterization of software vulnerabilities. In *ICSME*.

[D. Hin and Babar, 2022] D. Hin, A. Kan, H. C. and Babar, M. A. (2022). Linevd: Statement-level vulnerability detection using graph neural networks. In *MSR*.

[D. Izquierdo-Cortzar and Gonzlez-Barahona, 2022] D. Izquierdo-Cortzar, J. Alonso-Gutirrez, A. P. G.-P. G. R. and Gonzlez-Barahona, J. M. (2022). Starting the innersource journey: Key goals and metrics to measure collaboration. In *MSR*.

[D. J. Kim and Yang, 2021] D. J. Kim, N. Tsantalis, T. H. C. and Yang, J. (2021). Studying test annotation maintenance in the wild. In *ICSE*.

[D. Kavaler and Filkov, 2019] D. Kavaler, A. Trockman, B. V. and Filkov, V. (2019). Tool choice matters: Javascript quality assurance tools and usage outcomes in github projects. In *ICSE*.

[D. M. German and Inoue, 2018] D. M. German, G. Robles, G. P.-C. X. Y. H. I. and Inoue, K. (2018). Was my contribution fairly reviewed?" a framework to study the perception of fairness in modern code reviews. In *ICSE*.

[D. N. Palacio and Shenefiel, 2019] D. N. Palacio, D. McCrystal, K. M.-C. B.-C. D. P. and Shenefiel, C. (2019). Learning to identify security-related issues using convolutional neural networks. In *ICSME*.

[D. Oliveira and Castor, 2020] D. Oliveira, R. Bruno, F. M. and Castor, F. (2020). Evaluating code readability and legibility: An examination of human-centric studies. In *ICSME*.

[D. Spadini and Bacchelli, 2018a] D. Spadini, F. Palomba, A. Z. M. B. and Bacchelli, A. (2018a). On the relation of test smells to software code quality. In *ICSME*.

[D. Spadini and Bacchelli, 2018b] D. Spadini, M. Aniche, M. A. S. M. B. and Bacchelli, A. (2018b). When testing meets code review: Why and how developers review tests. In *ICSE*.

[de Souza Santos and Ralph, 2022] de Souza Santos, R. E. and Ralph, P. (2022). A grounded theory of coordination in remote-first and hybrid software teams. In *ICSE*.

[Dey and Woods, 2022] Dey, S. and Woods, W. (2022). Lagoon: An analysis tool for open source communities. In *MSR*.

[Dey et al., 2020] Dey, T., Mousavi, S., Ponce, E., Fry, T., Vasilescu, B., Filippova, A., and Mockus, A. (2020). Detecting and characterizing bots that commit code. In *MSR*.

[Dong et al., 2022] Dong, J., Lou, Y., Zhu, Q., Sun, Z., Li, Z., Zhang, W., and Hao, D. (2022). Fira: Fine-grained graph-based code change representation for automated commit message generation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 970–981.

[E. Biswas and Vijay-Shanker, 2020] E. Biswas, M. E. Karabulut, L. P. and Vijay-Shanker, K. (2020). Achieving reliable sentiment analysis in the software engineering domain using bert. In *ICSME*.

[E. Dias and Pinto, 2021] E. Dias, P. Meirelles, F. C. I. S. I. W. and Pinto, G. (2021). What makes a great maintainer of open source projects? In *ICSE*.

[E. Kalliamvakou and German, 2018] E. Kalliamvakou, C. Bird, T. Z. A. B. R. D. and German, D. M. (2018). What makes a great manager of software engineers? In *ICSE*.

[E. Koshchenko and Kovalenko, 2022] E. Koshchenko, E. K. and Kovalenko, V. (2022). Multimodal recommendation of messenger channels. In *MSR*.

[Eberhart and McMillan, 2021] Eberhart, Z. and McMillan, C. (2021). Dialogue management for interactive api search. In *ICSME*.

[Efstathiou et al., 2018] Efstathiou, V., Chatzilenas, C., and Spinellis, D. (2018). Word embeddings for the software engineering domain. In *MSR*.

[Efstathiou and Spinellis, 2019] Efstathiou, V. and Spinellis, D. (2019). Semantic source code models using identifier embeddings. In *MSR*.

[El Zarif et al., 2020] El Zarif, O., Da Costa, D. A., Hassan, S., and Zou, Y. (2020). On the relationship between user churn and software issues. In *MSR*.

[F. Calefato and Novielli, 2018] F. Calefato, F. Lanubile, F. M. and Novielli, N. (2018). Sentiment polarity detection for software development. In *ICSE*.

[F. Ebert and Serebrenik, 2018] F. Ebert, F. Castor, N. N. and Serebrenik, A. (2018). Communicative intention in code review questions. In *ICSME*.

[F. Heseding and Dllner, 2022] F. Heseding, W. S. and Dllner, J. (2022). Tooling for time- and space-efficient git repository mining. In *MSR*.

[F. Khoshnoud and Sami, 2022] F. Khoshnoud, A. R. Nasab, Z. T. and Sami, A. (2022). Which bugs are missed in code reviews: An empirical study on smartshark dataset. In *MSR*.

[F. Palomba and Lucia, 2018] F. Palomba, A. Z. and Lucia, A. D. (2018). Automatic test smell detection using information retrieval techniques. In *ICSME*.

[F. Santos and Gerosa, 2021] F. Santos, I. Wiese, B. T. I. S. A. S. and Gerosa, M. A. (2021). Can i solve it? identifying apis required to complete oss tasks. In *MSR*.

[F. Sarker and Filkov, 2019] F. Sarker, B. Vasilescu, K. B. and Filkov, V. (2019). Socio-technical work-rate increase associates with changes in work patterns in online projects. In *ICSE*.

[F. Silavong and Otter, 2022] F. Silavong, S. Moran, A. G. R. S. and Otter, R. (2022). Senatus - a fast and accurate code-to-code recommendation engine. In *MSR*.

[F. Wen and Bavota, 2021a] F. Wen, E. Aghajani, C. N. M. L. and Bavota, G. (2021a). Siri, write the next method. In *ICSE*.

[F. Wen and Bavota, 2021b] F. Wen, V. Ferrari, E. A. C. N. M. L. and Bavota, G. (2021b). Fears: Recommending complete android method implementations. In *ICSME*.

[Fang et al., 2020] Fang, H., Klug, D., Lamba, H., Herbsleb, J., and Vasilescu, B. (2020). Need for tweet: How open source developers talk about their github work on twitter. In *MSR*.

[Frcz and Dajda, 2018] Frcz, W. and Dajda, J. (2018). Developers' game: A preliminary study concerning a tool for automated developers assessment. In *ICSME*.

[G. Catolino and Ferrucci, 2019] G. Catolino, F. Palomba, A. Z. and Ferrucci, F. (2019). How the experience of development teams relates to assertion density of test classes. In *ICSME*.

[G. Grano and Gall, 2020] G. Grano, C. De Iaco, F. P. and Gall, H. C. (2020). Pizza versus pinsa: On the perception and measurability of unit test code quality. In *ICSME*.

[G. Kudrjavets and Rastogi, 2022a] G. Kudrjavets, N. N. and Rastogi, A. (2022a). The unexplored treasure trove of phabricator code reviews. In *MSR*.

[G. Kudrjavets and Rastogi, 2022b] G. Kudrjavets, A. Kumar, N. N. and Rastogi, A. (2022b). Mining code review data to understand waiting times between acceptance and merging: An empirical analysis. In *MSR*.

[G. L. Scoccia and Autili, 2021] G. L. Scoccia, P. M. and Autili, M. (2021). Challenges in developing desktop web apps: a study of stack overflow and github. In *MSR*.

[G. Nguyen and Rajan, 2022] G. Nguyen, M. J. Islam, R. P. and Rajan, H. (2022). Manas: Mining software repositories to assist automl. In *ICSE*.

[G. Rong and Zhang, 2022] G. Rong, Y. Zhang, L. Y. F. Z. H. K. and Zhang, H. (2022). Modeling review history for reviewer recommendation: A hypergraph approach. In *ICSE*.

[Geiger et al., 2018] Geiger, F.-X., Malavolta, I., Pascarella, L., Palomba, F., Di Nucci, D., and Bacchelli, A. (2018). A graph-based dataset of commit history of real-world android apps. In *MSR*.

[Gerosa et al., 2021] Gerosa, M., Wiese, I., Trinkenreich, B., Link, G., Robles, G., Treude, C., Steinmacher, I., and Sarma, A. (2021). The shifting sands of motivation: Revisiting what drives contributors in open source. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1046–1058.

[Gold and Krinke, 2020] Gold, N. E. and Krinke, J. (2020). Ethical mining: A case study on msr mining challenges. In *MSR*.

[Golubev et al., 2020] Golubev, Y., Eliseeva, M., Povarov, N., and Bryksin, T. (2020). A study of potential code borrowing and license violations in java projects on github. In *MSR*.

[Gong and Chen, 2022] Gong, J. and Chen, T. (2022). Does configuration encoding matter in learning software performance? an empirical study on encoding schemes. In *MSR*.

[Gren and Ralph, 2022] Gren, L. and Ralph, P. (2022). What makes effective leadership in agile software development teams? In *ICSE*.

[Guo and Singh, 2020] Guo, H. and Singh, M. P. (2020). Caspar: Extracting and synthesizing user stories of problems from app reviews. In *ICSE*.

[H. Alsolai and Nassar, 2018] H. Alsolai, M. R. and Nassar, D. (2018). Predicting software maintainability in object-oriented systems using ensemble techniques. In *ICSME*.

[H. Fang and Vasilescu, 2022] H. Fang, H. Lamba, J. H. and Vasilescu, B. (2022). This is damn slick! estimating the impact of tweets on open source project popularity and new contributors. In *ICSE*.

[H. Hata and Ishio, 2019] H. Hata, C. Treude, R. G. K. and Ishio, T. (2019). 9.6 million links in source code comments: Purpose, evolution, and decay. In *ICSE*.

[H. Isotani and Saito, 2021] H. Isotani, H. Washizaki, Y. F. T. N. S. O. and Saito, S. (2021). Duplicate bug report detection by using sentence embedding and fine-tuning. In *ICSME*.

[H. Peleg and Yahav, 2018] H. Peleg, S. S. and Yahav, E. (2018). Programming not only by example. In *ICSE*.

[H. Phan and Nguyen, 2018] H. Phan, H. A. Nguyen, N. M. T. L. H. T. A. T. N. and Nguyen, T. N. (2018). Statistical learning of api fully qualified names in code snippets of online forums. In *ICSE*.

[H. S. Qiu and Vasilescu, 2019] H. S. Qiu, A. Nolte, A. B. A. S. and Vasilescu, B. (2019). Going farther together: The impact of social capital on sustained participation in open source. In *ICSE*.

[H. Sellik and Aniche, 2021] H. Sellik, O. van Paridon, G. G. and Aniche, M. (2021). Learning off-by-one mistakes: An empirical study. In *MSR*.

[Hadar et al., 2018] Hadar, I., Hasson, T., Ayalon, O., Toch, E., Birnhack, M., Sherman, S., and Balissa, A. (2018). [journal first] privacy by designers: Software developers' privacy mindset. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 396–396.

[Hardt, 2020a] Hardt, R. (2020a). A software maintenance-focused process and supporting toolset for academic environments. In *ICSME*.

[Hardt, 2020b] Hardt, R. (2020b). A toolset to support a software maintenance process in academic environments. In *ICSME*.

[Hasabnis, 2022] Hasabnis, N. (2022). Gitrank: A framework to rank github repositories. In *MSR*.

[Hassan and Hill, 2018] Hassan, M. and Hill, E. (2018). Toward automatic summarization of arbitrary java statements for novice programmers. In *ICSME*.

[Hilderbrand et al., 2020] Hilderbrand, C., Perdriau, C., Letaw, L., Emard, J., Steine-Hanson, Z., Burnett, M., and Sarma, A. (2020). Engineering gender-inclusivity into software: Ten teams' tales from the trenches. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 433444, New York, NY, USA. Association for Computing Machinery.

[Hora, 2021] Hora, A. (2021). Googling for software development: What developers search for and what they find. In *MSR*.

[Hosono et al., 2019] Hosono, M., Washizaki, H., Honda, K., Nagumo, H., Sonoda, H., Fukazawa, Y., Munakata, K., Nakagawa, T., Nemoto, Y., Tokumoto, S., and Monpratarnchai, S. (2019). Inappropriate usage examples in web api documentations. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 343–347.

[Hu et al., 2021] Hu, Y., Wang, H., Ji, T., Xiao, X., Luo, X., Gao, P., and Guo, Y. (2021). Champ: Characterizing undesired app behaviors from user comments based on market policies. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 933–945.

[Huang et al., 2019] Huang, Y., Liu, X., Krueger, R., Santander, T., Hu, X., Leach, K., and Weimer, W. (2019). Distilling neural representations of data structure manipulation using fmri and fnirs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 396–407.

[I. Ferreira and Cheng, 2022] I. Ferreira, B. A. and Cheng, J. (2022). How heated is it? understanding github locked issues. In *MSR*.

[I. Malavolta and Lago, 2018] I. Malavolta, R. Verdecchia, B. F. M. B. and Lago, P. (2018). How maintainability issues of android apps evolve. In *ICSME*.

[I. Steinmacher and Gerosa, 2018] I. Steinmacher, G. Pinto, I. S. W. and Gerosa, M. A. (2018). Almost there: A study on quasi-contributors in open-source software projects. In *ICSE*.

[Imran, 2019] Imran, A. (2019). Design smell detection and analysis for open source java software. In *ICSME*.

[Ingram and Drachen, 2020] Ingram, C. and Drachen, A. (2020). How software practitioners use informal local meetups to share software engineering knowledge. In *ICSE*.

[J. Anderson and Rodeghero, 2020] J. Anderson, I. S. and Rodeghero, P. (2020). Assessing the characteristics of foss contributions in network automation projects. In *ICSME*.

[J. Chen and Wang, 2022] J. Chen, P. W. and Wang, W. (2022). Online summarizing alerts through semantic and behavior information. In *ICSE*.

[J. Dominic and Rodeghero, 2020] J. Dominic, B. Tubre, C. R. J. H. C. S. and Rodeghero, P. (2020). Remote pair programming in virtual reality. In *ICSME*.

[J. Fischer and Vogel-Heuser, 2018] J. Fischer, S. Bougouffa, A. S. I. S. and Vogel-Heuser, B. (2018). A qualitative study of variability management of control software for industrial automation systems. In *ICSME*.

[J. G. Young and Bagrow, 2021] J. G. Young, A. Casari, K. M. M. Z. T. L. H.-D. and Bagrow, J. P. (2021). Which contributions count? analysis of attribution in open source. In *MSR*.

[J. Hallett and Rashid, 2021] J. Hallett, N. Patnaik, B. S. and Rashid, A. (2021). Do this! do that!, and nothing will happen do specifications lead to securely stored passwords? In *ICSE*.

[J. Hu and Seltzer, 2020] J. Hu, J. Joung, M. J. K. Z. G. and Seltzer, M. I. (2020). Improving data scientist efficiency with provenance. In *ICSE*.

[J. Johnson and Sharif, 2019] J. Johnson, S. Lubo, N. Y. J. A. and Sharif, B. (2019). An empirical study assessing source code readability in comprehension. In *ICSME*.

[J. Krger and Leich, 2018] J. Krger, J. Wiemann, W. F. G. S. and Leich, T. (2018). Do you remember this source code? In *ICSE*.

[J. Kubelka and Bergel, 2018] J. Kubelka, R. R. and Bergel, A. (2018). The road to live programming: Insights from the practice. In *ICSE*.

[J. Li and Tan, 2021] J. Li, R. Huang, W. L. K. Y. and Tan, W. (2021). Toward less hidden cost of code completion with acceptance and ranking models. In *ICSME*.

[J. Lin and Cleland-Huang, 2021] J. Lin, Y. Liu, Q. Z. M. J. and Cleland-Huang, J. (2021). Traceability transformed: Generating more accurate links with pre-trained bert models. In *ICSE*.

[J. Pantiuchina and Bavota, 2018] J. Pantiuchina, M. L. and Bavota, G. (2018). Improving code: The (mis) perception of quality metrics. In *ICSME*.

[J. Pasuksmit and Karunasekera, 2021] J. Pasuksmit, P. T. and Karunasekera, S. (2021). Towards just-enough documentation for agile effort estimation: What information should be documented? In *ICSME*.

[J. Pasuksmit and Karunasekera, 2022] J. Pasuksmit, P. T. and Karunasekera, S. (2022). Towards reliable agile iterative planning via predicting documentation changes of work items. In *MSR*.

[J. Pei and Guan, 2021] J. Pei, Y. Wu, Z. Q. Y. C. and Guan, J. (2021). Attention-based model for predicting question relatedness on stack overflow. In *MSR*.

[J. Sun and Peng, 2019] J. Sun, Z. Xing, R. C. H. B. J. W. and Peng, X. (2019). Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph. In *ICSME*.

[J. Zhang and Liu, 2020] J. Zhang, X. Wang, H. Z. H. S. and Liu, X. (2020). Retrieval-based neural source code summarization. In *ICSE*.

[K. Damevski and Pollock, 2018] K. Damevski, H. Chen, D. C. S. N. A. K. and Pollock, L. (2018). Predicting future developer behavior in the ide using topic models. In *ICSE*.

[K. Gallagher and Kozaitis, 2019] K. Gallagher, M. F. and Kozaitis, S. (2019). Teaching software maintenance. In *ICSME*.

[K. M. Subash and Baysal, 2022] K. M. Subash, L. P. Kumar, S. L. V. P. C. and Baysal, O. (2022). Disco: A dataset of discord chat conversations for software engineering research. In *MSR*.

[K. Noda and Kikuchi, 2021] K. Noda, H. Y. and Kikuchi, S. (2021). Sirius: Static program repair with dependence graph-based systematic edit patterns. In *ICSME*.

[K. P. Neupane and Wang, 2019] K. P. Neupane, K. C. and Wang, Y. (2019). Emod: An end-to-end approach for investigating emotion dynamics in software development. In *ICSME*.

[K. Truong and Kstner, 2022] K. Truong, C. Miller, B. V. and Kstner, C. (2022). The unsolvable problem or the unheard answer? a dataset of 24,669 open-source software conference talks. In *MSR*.

[Khalil, 2019] Khalil, Z. A. (2019). Studying the impact of policy changes on bug handling performance. In *ICSME*.

[Kim et al., 2018] Kim, K., Kim, D., Bissyand, T. F., Choi, E., Li, L., Klein, J., and Le Traon, Y. (2018). Facoy a code-to-code search engine. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 946–957.

[Klein and Henley, 2021] Klein, B. P. and Henley, A. Z. (2021). Coderibbon: More efficient workspace management and navigation for mainstream development environments. In *ICSME*.

[L. Braz and Bacchelli, 2022] L. Braz, C. Aeberhard, G. a. and Bacchelli, A. (2022). Less is more: Supporting developers in vulnerability detection during code review. In *ICSE*.

[L. D. Silva and Moisakis, 2020] L. D. Silva, P. Borba, W. M. T. B. and Moisakis, J. (2020). Detecting semantic conflicts via automated behavior change detection. In *ICSME*.

[L. Moldon and Wachs, 2021] L. Moldon, M. S. and Wachs, J. (2021). How gamification affects software developers: Cautionary evidence from a natural experiment on github. In *ICSE*.

[L. Shi and Wang, 2020] L. Shi, M. Xing, M. L. Y. W. S. L. and Wang, Q. (2020). Detection of hidden feature requests from massive chat messages via deep siamese network. In *ICSE*.

[L. Sui and Fourtounis, 2020] L. Sui, J. Dietrich, A. T. and Fourtounis, G. (2020). On the recall of static call graph construction in practice. In *ICSE*.

[L. Yin and Filkov, 2021] L. Yin, Z. Zhang, Q. X. and Filkov, V. (2021). Apache software foundation incubator project sustainability dataset. In *MSR*.

[Lamothe and Shang, 2020] Lamothe, M. and Shang, W. (2020). When apis are intentionally bypassed: An exploratory study of api workarounds. In *ICSE*.

[Le and Zhang, 2022] Le, V. H. and Zhang, H. (2022). Log-based anomaly detection with deep learning: How far are we? In *ICSE*.

[Lee and Carver, 2019] Lee, A. and Carver, J. C. (2019). Floss participants' perceptions about gender and inclusiveness: A survey. In *ICSE*.

[Li et al., 2018] Li, H., Li, S., Sun, J., Xing, Z., Peng, X., Liu, M., and Zhao, X. (2018). Improving api caveats accessibility by mining api caveats knowledge graph. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 183–193.

[Liu et al., 2019] Liu, K., Kim, D., Bissyand, T. F., Kim, T., Kim, K., Koyuncu, A., Kim, S., and Le Traon, Y. (2019). Learning to spot and refactor inconsistent method names. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1–12.

[Luan et al., 2019] Luan, S., Yang, D., Barnaby, C., Sen, K., and Chandra, S. (2019). Aroma: Code recommendation via structural code search. *Proc. ACM Program. Lang.*, 3(OOPSLA).

[Lyulina and Jahanshahi, 2021] Lyulina, E. and Jahanshahi, M. (2021). Building the collaboration graph of open-source software ecosystem. In *MSR*.

[M. A. Al Alamin and Iqbal, 2021] M. A. Al Alamin, S. Malakar, G. U. S. A. T. B. H. and Iqbal, A. (2021). An empirical study of developer discussions on low-code software development challenges. In *MSR*.

[M. Ciniselli and Bavota, 2022] M. Ciniselli, L. P. and Bavota, G. (2022). To what extent do deep learning-based code recommenders generate predictions by cloning code from the training set? In *MSR*.

[M. Ciniselli and Bavota, 2021] M. Ciniselli, N. Cooper, L. P. D. P. M. D. P. and Bavota, G. (2021). An empirical study on the usage of bert models for code completion. In *MSR*.

[M. Claes and Adams, 2018] M. Claes, M. V. Mntyl, M. K. and Adams, B. (2018). Do programmers work at night or during the weekend? In *ICSE*.

[M. D. Stefano and Lucia, 2020] M. D. Stefano, F. Pecorelli, D. A. T. F. P. and Lucia, A. D. (2020). Refactoring recommendations based on the optimization of socio-technical congruence. In *ICSME*.

[M. Endres and Weimer, 2022] M. Endres, K. B. and Weimer, W. (2022). Hashing it out: A survey of programmers' cannabis usage, perception, and motivation. In *ICSE*.

[M. Endres and Weimer, 2021] M. Endres, Z. Karas, X. H. I. K. and Weimer, W. (2021). Relating reading, visualization, and coding for new programmers: A neuroimaging study. In *ICSE*.

[M. Galster and Blincoe, 2019] M. Galster, C. T. and Blincoe, K. (2019). Supporting software architecture maintenance by providing task-specific recommendations. In *ICSME*.

[M. H. Houekpetodji and Sudich, 2021] M. H. Houekpetodji, N. Anquetil, S. D. F. D. and Sudich, J. (2021). Report from the trenches a case study in modernizing software development practices. In *ICSME*.

[M. Hammoudi and Egyed, 2021] M. Hammoudi, C. Mayr-Dorn, A. M. and Egyed, A. (2021). A traceability dataset for open source systems. In *MSR*.

[M. Iammarino and Penta, 2019] M. Iammarino, F. Zampetti, L. A. and Penta, M. D. (2019). Self-admitted technical debt removal and refactoring actions: Co-occurrence or more? In *ICSME*.

[M. Jimenez and Papadakis, 2018a] M. Jimenez, C. Maxime, Y. L. T. and Papadakis, M. (2018a). On the impact of tokenizer and parameters on n-gram based code analysis. In *ICSME*.

[M. Jimenez and Papadakis, 2018b] M. Jimenez, C. Maxime, Y. L. T. and Papadakis, M. (2018b). Tuna: Tuning naturalness-based analysis. In *ICSME*.

[M. Kim and Begel, 2018] M. Kim, T. Zimmermann, R. D. and Begel, A. (2018). Data scientists in software teams: State of the art and challenges. In *ICSE*.

[M. Mondal and Schneider, 2019] M. Mondal, B. Roy, C. K. R. and Schneider, K. A. (2019). Investigating context adaptation bugs in code clones. In *ICSME*.

[M. Motwani and Brun, 2018] M. Motwani, S. Sankaranarayanan, R. J. and Brun, Y. (2018). Do automated program repair techniques repair hard and important bugs? In *ICSE*.

[M. O. F. Rokon and Faloutsos, 2021] M. O. F. Rokon, P. Yan, R. I. and Faloutsos, M. (2021). Repo2vec: A comprehensive embedding approach for determining repository similarity. In *ICSME*.

[M. Osama and Ibrahim, 2020] M. Osama, A. Zaki-Ismail, M. A. J. G. and Ibrahim, A. (2020). Score-based automatic detection and resolution of syntactic ambiguity in natural language requirements. In *ICSME*.

[M. P. Robillard and McIntosh, 2018] M. P. Robillard, M. N. and McIntosh, S. (2018). Threats of aggregating software repository data. In *ICSME*.

[M. Papoutsoglou and Kapitsaki, 2021] M. Papoutsoglou, J. W. and Kapitsaki, G. M. (2021). Mining dev for social and technical insights about software development. In *MSR*.

[M. Sridharan and Rantala, 2022] M. Sridharan, M. Mntyl, M. C. and Rantala, L. (2022). Soccminer: A source code-comments and comment-context miner. In *MSR*.

[M. Tufano and Poshyvanyk, 2019] M. Tufano, J. Pantiuchina, C. W. G. B. and Poshyvanyk, D. (2019). On learning meaningful code changes via neural machine translation. In *ICSE*.

[M. Tushev and Mahmoud, 2021] M. Tushev, F. E. and Mahmoud, A. (2021). Analysis of non-discrimination policies in the sharing economy. In *ICSME*.

[M. Tushev and Mahmoud, 2019] M. Tushev, S. K. and Mahmoud, A. (2019). Linguistic change in open source software. In *ICSME*.

[M. Viggiato and Kstner, 2019] M. Viggiato, J. Oliveira, E. F. P. J. and Kstner, C. (2019). How do code changes evolve in different platforms? a mining-based investigation. In *ICSME*.

[M. Wessel and Gerosa, 2020] M. Wessel, A. Serebrenik, I. W. I. S. and Gerosa, M. A. (2020). Effects of adopting code review bots on pull requests to oss projects. In *ICSME*.

[M. Wyrich and Wagner, 2021] M. Wyrich, A. Preikschat, D. G. and Wagner, S. (2021). The mind is a powerful place: How showing code comprehensibility metrics influences code understanding. In *ICSE*.

[Ma et al., 2022] Ma, W., Zhao, M., Soremekun, E., Hu, Q., Zhang, J. M., Papadakis, M., Cordy, M., Xie, X., and Le Traon, Y. (2022). Graphcode2vec: Generic code embedding via lexical and program dependence analyses. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 524–536.

[Majumder et al., 2018] Majumder, S., Balaji, N., Brey, K., Fu, W., and Menzies, T. (2018). 500+ times faster than deep learning: A case study exploring faster methods for text mining stackoverflow. In *MSR*.

[Manes and Baysal, 2021] Manes, S. S. and Baysal, O. (2021). Studying the change histories of stack overflow and github snippets. In *MSR*.

[Mantyla et al., 2018] Mantyla, M. V., Calefato, F., and Claes, M. (2018). Natural language or not (nlon): A package for software engineering text analysis pipeline. In *MSR*.

[Mashhadi and Hemmati, 2021] Mashhadi, E. and Hemmati, H. (2021). Applying codebert for automated program repair of java simple bugs. In *MSR*.

[Mayr-Dorn and Egyed, 2018] Mayr-Dorn, C. and Egyed, A. (2018). Does the propagation of artifact changes across tasks reflect work dependencies? In *ICSE*.

[Mendez et al., 2018] Mendez, C., Padala, H. S., Steine-Hanson, Z., Hildebrand, C., Horvath, A., Hill, C., Simpson, L., Patil, N., Sarma, A., and Burnett, M. (2018). Open source barriers to entry, revisited: A sociotechnical perspective. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1004–1015.

[Middleton et al., 2018] Middleton, J., Murphy-Hill, E., Green, D., Meade, A., Mayer, R., White, D., and McDonald, S. (2018). Which contributions predict whether developers are accepted into github teams. In *MSR*.

[Moran et al., 2020] Moran, K., Palacio, D. N., Bernal-Crdenas, C., McCrystal, D., Poshyvanyk, D., Shenefiel, C., and Johnson, J. (2020). Improving the effectiveness of traceability link recovery using hierarchical bayesian networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 873–885.

[Motwani and Brun, 2019] Motwani, M. and Brun, Y. (2019). Automatically generating precise oracles from structured natural language specifications. In *ICSE*.

[Murphy-Hill et al., 2019] Murphy-Hill, E., Smith, E. K., Sadowski, C., Jaspan, C., Winter, C., Jorde, M., Knight, A., Trenk, A., and Gross, S. (2019). Do developers discover new tools on the toilet? In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 465–475.

[N. Cassee and Serebrenik, 2021] N. Cassee, C. Kitsanelis, E. C. and Serebrenik, A. (2021). Human, bot or both? a study on the capabilities of classification models on mixed accounts. In *ICSME*.

[N. Cooper and Poshyvanyk, 2021] N. Cooper, C. Bernal-Crdenas, O. C. K. M. and Poshyvanyk, D. (2021). It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In *ICSE*.

[N. Imtiaz and Murphy-Hill, 2019] N. Imtiaz, J. Middleton, J. C. N. R. G. B. and Murphy-Hill, E. (2019). Investigating the effects of gender bias on github. In *ICSE*.

[N. J. Abid and Maletic, 2019] N. J. Abid, B. Sharif, N. D. H. A. and Maletic, J. I. (2019). Developer reading behavior while summarizing java methods: Size and context matters. In *ICSE*.

[N. Nahar and Kstner, 2022] N. Nahar, S. Zhou, G. L. and Kstner, C. (2022). Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *ICSE*.

[N. Rao and Hellendoorn, 2022] N. Rao, J. Tsay, M. H. and Hellendoorn, V. J. (2022). Comments on comments: Where code review and documentation meet. In *MSR*.

[N. Riquet and Vanderose, 2022] N. Riquet, X. D. and Vanderose, B. (2022). Gitdelver enterprise dataset (gded): An industrial closed-source dataset for socio-technical research. In *MSR*.

[N. Sae-Lim and Saeki, 2019] N. Sae-Lim, S. H. and Saeki, M. (2019). Can automated impact analysis techniques help predict decaying modules? In *ICSME*.

[N. Shimada and Matsumoto, 2022] N. Shimada, T. Xiao, H. H. C. T. and Matsumoto, K. (2022). Github sponsors: Exploring a new way to contribute to open source. In *ICSE*.

[N. Shrestha and Parnin, 2020] N. Shrestha, C. Botta, T. B. and Parnin, C. (2020). Here we go again: Why is it difficult for developers to learn another programming language? In *ICSE*.

[N. Tsantalis and Dig, 2018] N. Tsantalis, M. Mansouri, L. E. D. M. and Dig, D. (2018). Accurate and efficient refactoring detection in commit history. In *ICSE*.

[Nayrolles and Hamou-Lhadj, 2018] Nayrolles, M. and Hamou-Lhadj, A. (2018). Clever: Combining code metrics with clone detection for just-in-time fault prevention and resolution in large industrial projects. In *MSR*.

[Nguyen and Nadi, 2022] Nguyen, N. and Nadi, S. (2022). An empirical evaluation of github copilot's code suggestions. In *MSR*.

[Novielli et al., 2018] Novielli, N., Calefato, F., and Lanubile, F. (2018). A gold standard for emotion annotation in stack overflow. In *MSR*.

[O. Dabic and Bavota, 2021] O. Dabic, E. A. and Bavota, G. (2021). Sampling projects in github for msr studies. In *MSR*.

[O. Elazhary and Zaidman, 2019] O. Elazhary, M. A. Storey, N. E. and Zaidman, A. (2019). Do as i do, not as i say: Do contribution guidelines match the github contribution process? In *ICSME*.

[P. Chatterjee and Pollock, 2021] P. Chatterjee, K. D. and Pollock, L. (2021). Automatic extraction of opinion-based q&a from online developer chats. In *ICSE*.

[P. Chatterjee and Ralph, 2022] P. Chatterjee, T. S. and Ralph, P. (2022). Empirical standards for repository mining. In *MSR*.

[P. Gnoyke and Krger, 2021] P. Gnoyke, S. S. and Krger, J. (2021). An evolutionary analysis of software-architecture smells. In *ICSME*.

[P. P. Prachi and Barr, 2020] P. P. Prachi, S. K. Dash, C. T. and Barr, E. T. (2020). Posit: Simultaneously tagging natural and programming languages. In *ICSE*.

[P. R. Mazrae and Heydarnoori, 2021] P. R. Mazrae, M. I. and Heydarnoori, A. (2021). Automated recovery of issue-commit links leveraging both textual and non-textual data. In *ICSME*.

[P. Sri-iesaranusorn and Ishio, 2021] P. Sri-iesaranusorn, R. G. K. and Ishio, T. (2021). Does code review promote conformance? a study of openstack patches. In *MSR*.

[P. T. Nguyen and Penta, 2019] P. T. Nguyen, J. Di Rocco, D. D. R. L. O. T. D. and Penta, M. D. (2019). Focus: A recommender system for mining api function calls and usage patterns. In *ICSE*.

[Pecorelli et al., 2020] Pecorelli, F., Palomba, F., Khomh, F., and De Lucia, A. (2020). Developer-driven code smell prioritization. In *MSR*.

[Perera et al., 2020] Perera, H., Hussain, W., Whittle, J., Nurwidyantoro, A., Mougouei, D., Shams, R. A., and Oliver, G. (2020). A study on the prevalence of human values in software engineering publications, 2015 – 2018. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 409420, New York, NY, USA. Association for Computing Machinery.

[Peruma, 2019] Peruma, A. (2019). Towards a model to appraise and suggest identifier names. In *ICSME*.

[Pfeiffer, 2020] Pfeiffer, R.-H. (2020). What constitutes software? an empirical, descriptive study of artifacts. In *MSR*.

[Q. Feng and Hong, 2020] Q. Feng, C. F. and Hong, W. (2020). Graph neural network-based vulnerability predication. In *ICSME*.

[R. Alsuhaibani and Maletic, 2021] R. Alsuhaibani, C. Newman, M. D. M. C. and Maletic, J. (2021). On the naming of methods: A survey of professional developers. In *ICSE*.

[R. Degiovanni and Frias, 2018] R. Degiovanni, P. Castro, M. A. M. R. N. A. and Frias, M. (2018). Goal-conflict likelihood assessment based on model counting. In *ICSE*.

[R. Feng and Zhang, 2022] R. Feng, Z. Yan, S. P. and Zhang, Y. (2022). Automated detection of password leakage from public github repositories. In *ICSE*.

[R. Kallis and Panichella, 2019] R. Kallis, A. Di Sorbo, G. C. and Panichella, S. (2019). Ticket tagger: Machine learning driven issue classification. In *ICSME*.

[R. Krueger and Leach, 2020] R. Krueger, Y. Huang, X. L. T. S. W. W. and Leach, K. (2020). Neurological divide: An fmri study of prose and code writing. In *ICSE*.

[R. Pan and Kaufman, 2021] R. Pan, V. Le, N. N. S. G. S. L. and Kaufman, M. (2021). Can program synthesis be used to learn merge conflict resolutions? an empirical analysis. In *ICSE*.

[R. Paul and Bosu, 2021] R. Paul, A. K. T. and Bosu, A. (2021). Why security defects go unnoticed during code reviews? a case-control study of the chromium os project. In *ICSE*.

[R. Ramsauer and Mauerer, 2019] R. Ramsauer, D. L. and Mauerer, W. (2019). The list is the process: Reliable pre-integration tracking of commits on mailing lists. In *ICSE*.

[R. Rwemalika and Lorrach, 2019] R. Rwemalika, M. Kintis, M. P. Y. L. T. and Lorrach, P. (2019). An industrial study on the differences between pre-release and post-release bugs. In *ICSME*.

[R. Tufano and Bavota, 2022] R. Tufano, S. Masiero, A. M. L. P. D. P. and Bavota, G. (2022). Using pre-trained models to boost code review automation. In *ICSE*.

[R. Tufano and Bavota, 2021] R. Tufano, L. Pascarella, M. T. D. P. and Bavota, G. (2021). Towards automating code review activities. In *ICSE*.

[R. Wen and McIntosh, 2018] R. Wen, D. Gilbert, M. G. R. and McIntosh, S. (2018). Blimp tracer: Integrating build impact analysis with code review. In *ICSME*.

[Rabbani et al., 2018] Rabbani, N., Harvey, M. S., Saquif, S., Gallaba, K., and McIntosh, S. (2018). Revisiting "programmers' build errors" in the visual studio context: A replication study using ide interaction traces. In *MSR*.

[Rahman, 2018] Rahman, A. (2018). Comprehension effort and programming activities: Related? or not related? In *MSR*.

[Rahman et al., 2018] Rahman, M. M., Barson, J., Paul, S., Kayani, J., Lois, F. A., Quezada, S. F., Parnin, C., Stolee, K. T., and Ray, B. (2018). Evaluating how developers use general-purpose web-search for code retrieval. In *MSR*.

[Ramirez-Mora and Oktaba, 2018] Ramirez-Mora, S. L. and Oktaba, H. (2018). Team maturity in agile software development: The impact on productivity. In *ICSME*.

[Rodriguez et al., 2018] Rodriguez, A., Tanaka, F., and Kamei, Y. (2018). Empirical study on the relationship between developer's working habits and efficiency. In *MSR*.

[Rossi and Zacchiroli, 2022] Rossi, D. and Zacchiroli, S. (2022). Geographic diversity in public code contributions: An exploratory large-scale study over 50 years. In *MSR*.

[Ruvimova et al., 2022] Ruvimova, A., Lill, A., Gugler, J., Howe, L., Huang, E., Murphy, G., and Fritz, T. (2022). An exploratory study of productivity perceptions in software teams. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 99–111.

[S. Biswas and Rajan, 2022] S. Biswas, M. W. and Rajan, H. (2022). The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. In *ICSE*.

[S. Chattopadhyay and Sarma, 2019] S. Chattopadhyay, N. Nelson, Y. R. G. A. A. L. R. P. and Sarma, A. (2019). Latent patterns in activities: A field study of how developers manage context. In *ICSE*.

[S. Datta and Sarkar, 2021] S. Datta, A. Mysore, H. W. and Sarkar, S. (2021). Clustering, separation, and connection: A tale of three characteristics. In *ICSME*.

[S. E. Ponta and Sabetta, 2018] S. E. Ponta, H. P. and Sabetta, A. (2018). Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. In *ICSME*.

[S. Hassan and Hassan, 2018] S. Hassan, C. Tantithamthavorn, C. P. B. and Hassan, A. E. (2018). Studying the dialogue between users and developers of free apps in the google play store. In *ICSE*.

[S. Mondal and Roy, 2020] S. Mondal, G. U. and Roy, C. K. (2020). Automatic identification of rollback edit with reasons in stack overflow q&a site. In *ICSME*.

[S. Mondal and Roy, 2021] S. Mondal, G. U. and Roy, C. K. (2021). Rollback edit inconsistencies in developer forum. In *MSR*.

[S. N.C. and Menzies, 2021] S. N.C., S. M. and Menzies, T. (2021). Early life cycle software defect prediction. why? how? In *ICSE*.

[S. Nguyen and Nguyen, 2020] S. Nguyen, H. Phan, T. L. and Nguyen, T. N. (2020). Suggesting natural method names to check name consistencies. In *ICSE*.

[S. Reddy and Sen, 2020] S. Reddy, C. Lemieux, R. P. and Sen, K. (2020). Quickly generating diverse valid test inputs with reinforcement learning. In *ICSE*.

[S. Wang and Hassan, 2018] S. Wang, T. H. C. and Hassan, A. E. (2018). Understanding the factors for fast answers in technical q&a websites: An empirical study of four stack exchange websites. In *ICSE*.

[S. Wang and Hassan, 2021] S. Wang, D. M. German, T. H. C. Y. T. and Hassan, A. E. (2021). Is reputation on stack overflow always a good indicator for users' expertise? no! In *ICSME*.

[S. Yatish and Tantithamthavorn, 2019] S. Yatish, J. Jiarpakdee, P. T. and Tantithamthavorn, C. (2019). Mining software defects: Should we consider affected releases? In *ICSE*.

[Sabetta and Bezzi, 2018] Sabetta, A. and Bezzi, M. (2018). A practical approach to the automatic classification of security-relevant commits. In *ICSME*.

[Saha et al., 2022] Saha, R. K., Ura, A., Mahajan, S., Zhu, C., Li, L., Hu, Y., Yoshida, H., Khurshid, S., and Prasad, M. R. (2022). Sapientml: Synthesizing machine learning pipelines by learning from human-written solutions. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 1932–1944.

[Scalabrino et al., 2017] Scalabrino, S., Bavota, G., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D., and Oliveto, R. (2017). Automatically assessing code understandability: How far are we? In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 417–427. IEEE.

[Sharma and Kessentini, 2021a] Sharma, T. and Kessentini, M. (2021a). Qscored: A large dataset of code smells and quality metrics. In *MSR*.

[Sharma and Kessentini, 2021b] Sharma, T. and Kessentini, M. (2021b). Qscored: A large dataset of code smells and quality metrics. In *MSR*.

[Sharma and Sangwan, 2021] Sharma, T. and Sangwan, O. P. (2021). Sine-cosine algorithm for software fault prediction. In *ICSME*.

[Shen et al., 2021] Shen, B., Zhang, W., Yu, A., Wei, Z., Liang, G., Zhao, H., and Jin, Z. (2021). Cross-language code coupling detection: A preliminary study on android applications. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 378–388.

[Shi et al., 2022] Shi, L., Mu, F., Zhang, Y., Yang, Y., Chen, J., Chen, X., Jiang, H., Jiang, Z., and Wang, Q. (2022). Buglistener: Identifying and synthesizing bug reports from collaborative live chats. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 299311, New York, NY, USA. Association for Computing Machinery.

[Sillito and Kutomi, 2020] Sillito, J. and Kutomi, E. (2020). Failures and fixes: A study of software system incident response. In *ICSME*.

[T. Besker and Bosch, 2018] T. Besker, A. Martini, R. E. L. K. B. and Bosch, J. (2018). Embracing technical debt, from a startup company perspective. In *ICSME*.

[T. Dey and Mockus, 2021] T. Dey, A. K. and Mockus, A. (2021). Representation of developer expertise in open source software. In *ICSE*.

[T. Hoang and Lawall, 2020] T. Hoang, H. J. Kang, D. L. and Lawall, J. (2020). Cc2vec: Distributed representations of code changes. In *ICSE*.

[T. Ishio and Inoue, 2018] T. Ishio, N. Maeda, K. S. and Inoue, K. (2018). Cloned buggy code detection in practice using normalized compression distance. In *ICSME*.

[T. Lorey and Felderer, 2022] T. Lorey, P. R. and Felderer, M. (2022). Social science theories in software engineering research. In *ICSE*.

[T. Nguyen and Nguyen, 2019a] T. Nguyen, P. V. and Nguyen, T. (2019a). Personalized code recommendation. In *ICSME*.

[T. Nguyen and Nguyen, 2019b] T. Nguyen, P. V. and Nguyen, T. (2019b). Recommending exception handling code. In *ICSME*.

[T. Sedano and Praire, 2019] T. Sedano, P. R. and Praire, C. (2019). The product backlog. In *ICSE*.

[T. Zhang and Kim, 2018] T. Zhang, G. Upadhyaya, A. R. H. R. and Kim, M. (2018). Are code examples on an online q&a forum reliable?: A study of api misuse on stack overflow. In *ICSE*.

[T. Zhang and Kim, 2019] T. Zhang, D. Yang, C. L. and Kim, M. (2019). Analyzing and supporting adaptation of online code examples. In *ICSE*.

[T. Zhang and Jiang, 2020] T. Zhang, B. Xu, F. T. S. A. H. D. L. and Jiang, L. (2020). Sentiment analysis for software engineering: How far can pre-trained transformer models go? In *ICSME*.

[Tao et al., 2021] Tao, W., Wang, Y., Shi, E., Du, L., Han, S., Zhang, H., Zhang, D., and Zhang, W. (2021). On the evaluation of commit message generation models: An experimental study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 126–136.

[Theeten et al., 2019] Theeten, B., Vandeputte, F., and Van Cutsem, T. (2019). Import2vec learning embeddings for software libraries. In *MSR*.

[Trockman et al., 2018] Trockman, A., Cates, K., Mozina, M., Nguyen, T., Kastner, C., and Vasilescu, B. (2018). Automatically assessing code understandability" reanalyzed: Combined metrics matter. In *MSR*.

[Uchoa et al., 2021] Uchoa, A., Barbosa, C., Coutinho, D., Oizumi, W., Assuncao, W. K. G., Vergilio, S. R., Pereira, J. A., Oliveira, A., and Garcia, A. (2021). Predicting design impactful changes in modern code review: A large-scale empirical study. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 471–482.

[Uchoa et al., 2020] Uchoa, A., Barbosa, C., Oizumi, W., Blenilio, P., Lima, R., Garcia, A., and Bezerra, C. (2020). How does modern code review impact software design degradation? an in-depth empirical study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 511–522.

[V. Frick and Pinzger, 2018] V. Frick, T. Grassauer, F. B. and Pinzger, M. (2018). Generating accurate and compact edit scripts using tree differencing. In *ICSME*.

[V. Tawosi and Sarro, 2022] V. Tawosi, A. Al-Subaihin, R. M. and Sarro, F. (2022). A versatile dataset of agile open source software projects. In *MSR*.

[V. Zyrianov and Maletic, 2020] V. Zyrianov, D. T. Guarnera, C. S. P. B. S. and Maletic, J. I. (2020). Automated recording and semantics-aware replaying of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks. In *ICSME*.

[W. Amme and Schfer, 2021] W. Amme, T. S. H. and Schfer, A. (2021). You look so different: Finding structural clones and subclones in java source code. In *ICSME*.

[W. Kopec and Casati, 2018] W. Kopec, B. Balcerzak, R. N. G. K. A. W. and Casati, F. (2018). Older adults and hackathons: A qualitative study. In *ICSE*.

[W. Luo and Chen, 2021] W. Luo, D. Chai, X. R. J. W. C. F. and Chen, Z. (2021). Graph-based fuzz testing for deep learning inference engines. In *ICSE*.

[W. Xiao and Zhou, 2022] W. Xiao, H. He, W. X. X. T. J. D. and Zhou, M. (2022). Recommending good first issues in github oss projects. In *ICSE*.

[W. Yuan and Liu, 2021] W. Yuan, Y. Xiong, H. S. and Liu, X. (2021). Incorporating multiple features to predict bug fixing time with neural networks. In *ICSME*.

[W. Zhou and Shen, 2020] W. Zhou, Y. Zhao, G. Z. and Shen, X. (2020). Harp: Holistic analysis for refactoring python-based analytics programs. In *ICSE*.

[Wang et al., 2022] Wang, D., Jia, Z., Li, S., Yu, Y., Xiong, Y., Dong, W., and Liao, X. (2022). Bridging pre-trained models and downstream tasks for source code understanding. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 287–298.

[Wang and Kagdi, 2018] Wang, H. and Kagdi, H. (2018). A conceptual replication study on bugs that get fixed in open source software. In *ICSME*.

[Wessel et al., 2022] Wessel, M., Abdellatif, A., Wiese, I., Conte, T., Shihab, E., Gerosa, M. A., and Steinmacher, I. (2022). Bots for pull requests: The good, the bad, and the promising. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 274–286.

[Wilkie et al., 2018] Wilkie, J., Halabi, Z. A., Karaoglu, A., Liao, J., Ndungu, G., Ragkhitwetsagul, C., Paixao, M., and Krinke, J. (2018). Who's this? developer identification using ide event data. In *MSR*.

[X. Hu and Zimmermann, 2022] X. Hu, X. Xia, D. L. Z. W. Q. C. and Zimmermann, T. (2022). Practitioners' expectations on automated code comment generation. In *ICSE*.

[X. Huang and Yang, 2018] X. Huang, H. Zhang, X. Z. M. A. B. and Yang, S. (2018). Synthesizing qualitative research in software engineering: A critical review. In *ICSE*.

[X. Li and Orso, 2018] X. Li, S. Zhu, M. d. and Orso, A. (2018). Enlightened debugging. In *ICSE*.

[X. Liu and Ng, 2018] X. Liu, L. Huang, C. L. and Ng, V. (2018). Linking source code to untangled change intents. In *ICSME*.

[X. Xia and Li, 2018] X. Xia, L. Bao, D. L. Z. X. A. E. H. and Li, S. (2018). Measuring program comprehension: A large-scale field study with professionals. In *ICSE*.

[X. Xia and Lo, 2019] X. Xia, Z. Wan, P. S. K. and Lo, D. (2019). How practitioners perceive coding proficiency. In *ICSE*.

[X. Xu and Liu, 2021] X. Xu, Q. Zheng, Z. Y. M. F. A. J. and Liu, T. (2021). Interpretation-enabled software reuse detection based on a multi-level birthmark model. In *ICSE*.

[X. Zhou and Lo, 2021] X. Zhou, D. H. and Lo, D. (2021). Assessing generalizability of codebert. In *ICSME*.

[Xia et al., 2020] Xia, H., Zhang, Y., Zhou, Y., Chen, X., Wang, Y., Zhang, X., Cui, S., Hong, G., Zhang, X., Yang, M., and Yang, Z. (2020). How android developers handle evolution-induced api compatibility issues: A large-scale study. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 886–898.

[Y. Huang and Zimmermann, 2021] Y. Huang, D. F. and Zimmermann, T. (2021). Leaving my fingerprints: Motivations and challenges of contributing to oss for social good. In *ICSE*.

[Y. Li and Nguyen, 2022] Y. Li, S. W. and Nguyen, T. N. (2022). Dear: A novel deep learning-based approach for automated program repair. In *ICSE*.

[Y. Lu and Li, 2020] Y. Lu, X. Mao, M. Z. Y. Z. T. W. and Li, Z. (2020). Haste makes waste: An empirical study of fast answers in stack overflow. In *ICSME*.

[Y. Tang and Raja, 2021] Y. Tang, R. Khatchadourian, M. B. R. S. A. S. and Raja, A. (2021). An empirical study of refactorings and technical debt in machine learning systems. In *ICSE*.

[Y. Tian and Liu, 2022] Y. Tian, Y. Zhang, K. J. S. L. J. and Liu, H. (2022). What makes a good commit message? In *ICSE*.

[Y. Wan and Jin, 2022] Y. Wan, W. Zhao, H. Z. Y. S. G. X. and Jin, H. (2022). What do they capture? - a structural analysis of pre-trained language models for source code. In *ICSE*.

[Y. Wang and Wang, 2022] Y. Wang, J. Wang, H. Z. X. M. L. S. and Wang, Q. (2022). Where is your app frustrating users? In *ICSE*.

[Yu et al., 2022] Yu, H., Lou, Y., Sun, K., Ran, D., Xie, T., Hao, D., Li, Y., Li, G., and Wang, Q. (2022). Automated assertion generation via information retrieval and its integration with deep learning. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 163–174.

[Z. Li and Xu, 2022] Z. Li, G. Q. Chen, C. C. Y. Z. and Xu, S. (2022). Ropgen: Towards robust code authorship attribution via automatic coding style transformation. In *ICSE*.

[Z. Sun and Li, 2022] Z. Sun, L. Li, Y. L. X. D. and Li, L. (2022). On the importance of building high-quality training datasets for neural code search. In *ICSE*.

[Zacchiroli, 2022] Zacchiroli, S. (2022). A large-scale dataset of (open source) license text variants. In *MSR*.

[Zampetti et al., 2018] Zampetti, F., Serebrenik, A., and Di Penta, M. (2018). Was self-admitted technical debt removal a real removal? an in-depth perspective. In *MSR*.

[Zhang and Harman, 2021] Zhang, J. M. and Harman, M. (2021). Ignorance and prejudice" in software fairness. In *ICSE*.

[Zhang, 2020] Zhang, W. (2020). Efficient bug triage for industrial environments. In *ICSME*.

[Zhang et al., 2018] Zhang, X., Chen, Y., Gu, Y., Zou, W., Xie, X., Jia, X., and Xuan, J. (2018). How do multiple pull requests change the same code: A study of competing pull requests in github. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 228–239.

[Zhang and Hu, 2022] Zhang, X. and Hu, Z. (2022). Towards bidirectional live programming for incomplete programs. In *ICSE*.

[Zheng et al., 2021] Zheng, Y., Liu, Y., Xie, X., Liu, Y., Ma, L., Hao, J., and Liu, Y. (2021). Automatic web testing using curiosity-driven reinforcement learning. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 423–435.

[Zhu and Godfrey, 2021] Zhu, W. and Godfrey, M. W. (2021). Mea culpa: How developers fix their own simple bugs differently from other developers. In *MSR*.

[Zieris and Prechelt, 2020] Zieris, F. and Prechelt, L. (2020). Explaining pair programming session dynamics from knowledge gaps. In *ICSE*.