It Will Never Work in Theory: To Do

http://neverworkintheory.org

December 30, 2022

References

[**Abate2020**] Pietro Abate, Roberto Di Cosmo, Georgios Gousios, and Stefano Zacchiroli. Dependency solving is still hard, but we are getting better at it. 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), Feb 2020, London, Canada. pp.547-551, 2020, DOI 10.1109/SANER48275.2020.9054837.

[**AbouKhalil2022**] Zeinab Abou Khalil and Stefano Zacchiroli. The general index of software engineering papers. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3528494.

Abstract: We introduce the General Index of Software Engineering Papers, a dataset of fulltext-indexed papers from the most prominent scientific venues in the field of Software Engineering. The dataset includes both complete bibliographic information and indexed n-grams (sequence of contiguous words after removal of stopwords and non-words, for a total of 577 276 382 unique n-grams in this release) with length 1 to 5 for 44 581 papers retrieved from 34 venues over the 1971–2020 period. The dataset serves use cases in the field of meta-research, allowing to introspect the output of software engineering research even when access to papers or scholarly search engines is not possible (e.g., due to contractual reasons). The dataset also contributes to making such analyses reproducible and independently verifiable, as opposed to what happens when they are conducted using 3rd-party and non-open scholarly indexing services. The dataset is available as a portable Postgres database dump and released as open data.

[**Abreu2022**] Rui Abreu. The bumpy road of taking automated debugging to industry, 2022.

[**Ahmed2021**] Umair Z. Ahmed, Zhiyu Fan, Jooyong Yi, Omar I. Al-Bataineh, and Abhik Roychoudhury. Verifix: Verified repair of programming assignments, 2021.

[**Ait2022**] Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. An empirical study on the survival rate of GitHub projects. In *Proc. International*

Conference on Mining Software Repositories (MSR). ACM, May 2022, DOI 10.1145/3524842.3527941.
Abstract: The number of Open Source projects hosted in social coding platforms such as GitHub is constantly growing. However, many of these projects are not regularly maintained and some are even abandoned shortly after they were created. In this paper we analyze early project development dynamics in software projects hosted on GitHub, including their survival rate. To this aim, we collected all 1,127 GitHub repositories from four different ecosystems (i.e., NPM packages, R packages, WordPress plugins and Laravel packages) created in 2016. We stored their activity in a time series database and analyzed their activity evolution along their lifespan, from 2016 to now. Our results reveal that the prototypical development process consists of intensive coding-driven active periods followed by long periods of inactivity. More importantly, we have found that a significant number of projects die in the first year of existence with the survival rate decreasing year after year. In fact, the probability of surviving longer than five years is less than 50% though some types of projects have better chances of survival.

[**AlOmar2022**] Eman Abdullah AlOmar, Moataz Chouchen, Mohamed Wiem Mkaouer, and Ali Ouni. Code review practices for refactoring changes: An empirical study on openstack, 2022.

[**Andersen2020**] Leif Andersen, Michael Ballantyne, and Matthias Felleisen. Adding interactive visual syntax to textual code. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–28, Nov 2020, DOI 10.1145/3428290.
Abstract: Many programming problems call for turning geometrical thoughts into code: tables, hierarchical structures, nests of objects, trees, forests, graphs, and so on. Linear text does not do justice to such thoughts. But, it has been the dominant programming medium for the past and will remain so for the foreseeable future. This paper proposes a novel mechanism for conveniently extending textual programming languages with problem-specific visual syntax. It argues the necessity of this language feature, demonstrates the feasibility with a robust prototype, and sketches a design plan for adapting the idea to other languages.

[**Apple2021**] Jim Apple. Stretching your data with taffy filters, 2021.

[**Asare2022**] Owura Asare, Meiyappan Nagappan, and N. Asokan. Is github's copilot as bad as humans at introducing vulnerabilities in code?, 2022.

[**Barbosa2022**] Leonardo Barbosa, Victor Hugo Santiago, Alberto Luiz Oliveira Tavares de Souza, and Gustavo Pinto. To what extent cognitive-driven development improves code readability?, 2022.

[**Baxter2022**] Amanda L Baxter, Segev Y BenZvi, Walter Bonivento, Adam Brazier, Michael Clark, Alexis Coleiro, David Collom, Marta Colomer-Molla,

Bryce Cousins, Aliwen Delgado Orellana, Damien Dornic, Vladislav Ekimtcov, Shereen ElSayed, Andrea Gallo Rosso, Patrick Godwin, Spencer Griswold, Alec Habig, Remington Hill, Shunsaku Horiuchi, D Andrew Howell, Margaret W G Johnson, Mario Jurić, James P Kneller, Abigail Kopec, Claudio Kopper, Vladimir Kulikovskiy, Mathieu Lamoureux, Rafael F Lang, Shengchao Li, Massimiliano Lincetto, Lindy Lindstrom, Mark W Linvill, Curtis McCully, Jost Migenda, Danny Milisavljevic, Spencer Nelson, Rita Novoseltseva, Erin O'Sullivan, Donald Petravick, Barry W Pointon, Nirmal Raj, Andrew Renshaw, Janet Rumleskie, Tom Sonley, Ron Tapia, Jeffrey C L Tseng, Christopher D Tunnell, Godefroy Vannoye, Carlo F Vigorito, Clarence J Virtue, Christopher Weaver, Kathryn E Weil, Lindley Winslow, Rich Wolski, Xun-Jie Xu, Yiyang Xu, and The SCiMMA and SNEWS Collaborations. Collaborative experience between scientific software projects using agile scrum development. *Softw. Pract. Exp.*, 52(10):2077–2096, Oct 2022, DOI `10.1002/spe.3120`.

Abstract: Developing sustainable software for the scientific community requires expertise in software engineering and domain science. This can be challenging due to the unique needs of scientific software, the insufficient resources for software engineering practices in the scientific community, and the complexity of developing for evolving scientific contexts. While open-source software can partially address these concerns, it can introduce complicating dependencies and delay development. These issues can be reduced if scientists and software developers collaborate. We present a case study wherein scientists from the SuperNova Early Warning System collaborated with software developers from the Scalable Cyberinfrastructure for Multi-Messenger Astrophysics project. The collaboration addressed the difficulties of open-source software development, but presented additional risks to each team. For the scientists, there was a concern of relying on external systems and lacking control in the development process. For the developers, there was a risk in supporting a user-group while maintaining core development. These issues were mitigated by creating a second Agile Scrum framework in parallel with the developers' ongoing Agile Scrum process. This Agile collaboration promoted communication, ensured that the scientists had an active role in development, and allowed the developers to evaluate and implement the scientists' software requirements. The collaboration provided benefits for each group: the scientists actuated their development by using an existing platform, and the developers utilized the scientists' use-case to improve their systems. This case study suggests that scientists and software developers can avoid scientific computing issues by collaborating and that Agile Scrum methods can address emergent concerns.

[**Beasley2022**] Zachariah J Beasley and Ayesha R Johnson. The impact of remote pair programming in an upper-level CS course. In *Proc. Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Jul 2022, DOI `10.1145/3502718.3524772`.

Abstract: Pair programming is an active learning technique with several

benefits to students, including increasing participation and improving outcomes, particularly for female computer science students. However, most of the literature highlights the effects of pair programming in introductory courses, where students have different prior programming experience and thus may experience group issues. This work analyzes the effect of pair programming in an upper-level computer science course, where students have a more consistent background education, particularly in languages learned and coding best practices. Secondly, the effect of remote pair programming on student outcomes is still an open question of increasing importance with the advent of Covid-19. This work utilized split sections with a control and treatment group in a large, public university. In addition to comparing pair programming to individual programming, results were analyzed by modality (remote vs. in person) and by gender, focusing on how pair programming benefits female computer science students in confidence, persistence in the major, and outcomes. We found that pair programming groups scored higher on assignments and exams, that remote pair programming groups performed as well as in person groups, and that female students increased their confidence in asking questions in class and scored 12% higher in the course when utilizing pair programming.

[**Becker2022**] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming is hard – or at least it used to be: Educational opportunities and challenges of ai code generation, 2022.

[**Bendrissou2022**] Bachir Bendrissou, Rahul Gopinath, and Andreas Zeller. "synthesizing input grammars": a replication study. In *Proc. Conference on Programming Language Design and Implementation (PLDI)*. ACM, Jun 2022, DOI `10.1145/3519939.3523716`.

Abstract: When producing test inputs for a program, test generators ("fuzzers") can greatly profit from grammars that formally describe the language of expected inputs. In recent years, researchers thus have studied means to recover input grammars from programs and their executions. The GLADE algorithm by Bastani et al., published at PLDI 2017, was the first black-box approach to claim context-free approximation of input specification for non-trivial languages such as XML, Lisp, URLs, and more. Prompted by recent observations that the GLADE algorithm may show lower performance than reported in the original paper, we have reimplemented the GLADE algorithm from scratch. Our evaluation confirms that the effectiveness score (F1) reported in the GLADE paper is overly optimistic, and in some cases, based on the wrong language. Furthermore, GLADE fares poorly in several real-world languages evaluated, producing grammars that spend megabytes to enumerate inputs.

[**Bi2021**] Tingting Bi, Wei Ding, Peng Liang, and Antony Tang. Architecture information communication in two OSS projects: The why, who, when, and what. *Journal of Systems and Software*, 181:111035, Nov 2021, DOI

10.1016/j.jss.2021.111035.
Abstract: Architecture information is vital for Open Source Software (OSS) development, and mailing list is one of the widely used channels for developers to share and communicate architecture information. This work investigates the nature of architecture information communication (i.e., why, who, when, and what) by OSS developers via developer mailing lists. We employed a multiple case study approach to extract and analyze the architecture information communication from the developer mailing lists of two OSS projects, ArgoUML and Hibernate, during their development lifecycle of over 18 years. Our main findings are: (a) architecture negotiation and interpretation are the two main reasons (i.e., why) of architecture communication; (b) the amount of architecture information communicated in developer mailing lists decreases after the first stable release (i.e., when); (c) architecture communications centered around a few core developers (i.e., who); (d) and the most frequently communicated architecture elements (i.e., what) are Architecture Rationale and Architecture Model. There are a few similarities of architecture communication between the two OSS projects. Such similarities point to how OSS developers naturally gravitate towards the four aspects of architecture communication in OSS development.

[**Bijlsma2022**] Lex A Bijlsma, Arjan J F Kok, Harrie J M Passier, Harold J Pootjes, and Sylvia Stuurman. Evaluation of design pattern alternatives in java. *Softw. Pract. Exp.*, 52(5):1305–1315, May 2022, DOI 10.1002/spe.3061.
Abstract: Design patterns are standard solutions to common design problems. The famous Gang of Four book describes more than twenty design patterns for the object-oriented paradigm. These patterns were developed more than twenty-five years ago, using the programming language concepts available at that time. Patterns do not always fit underlying domain concepts. For example, even when a concrete strategy is a pure function, the classical strategy pattern represents this as a separate subclass and as such obscures the intent of this pattern with extra complexities due to the inheritance-based implementation. Due to the ongoing development of oo-languages, a relevant question is whether the implementation of these patterns can be improved using new language features, such that they fit more closely with the intent. An additional question is then how we can decide which implementation is to be preferred. In this article, we investigate both questions, using the strategy pattern as an example. Our main contribution is that we show how to reason about different implementations, using both the description of a design pattern and design principles as guidance.

[**Bittner2022**] Paul Maximilian Bittner, Christof Tinnes, Alexander Schultheiß, Sören Viegener, Timo Kehrer, and Thomas Thüm. Classifying edits to variability in source code. In *Proc. European Software Engineering Conference/International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov 2022, DOI 10.1145/3540250.3549108.
Abstract: For highly configurable software systems, such as the Linux

kernel, maintaining and evolving variability information along changes to source code poses a major challenge. While source code itself may be edited, also feature-to-code mappings may be introduced, removed, or changed. In practice, such edits are often conducted ad-hoc and without proper documentation. To support the maintenance and evolution of variability, it is desirable to understand the impact of each edit on the variability. We propose the first complete and unambiguous classification of edits to variability in source code by means of a catalog of edit classes. This catalog is based on a scheme that can be used to build classifications that are complete and unambiguous by construction. To this end, we introduce a complete and sound model for edits to variability. In about 21.5ms per commit, we validate the correctness and suitability of our classification by classifying each edit in 1.7 million commits in the change histories of 44 open-source software systems automatically. We are able to classify all edits with syntactically correct feature-to-code mappings and find that all our edit classes occur in practice.

[**Blacher2022**] Mark Blacher, Joachim Giesen, Peter Sanders, and Jan Wassenberg. Vectorized and performance-portable quicksort, 2022.

[**Blackwell2019**] Alan F. Blackwell, Marian Petre, and Luke Church. Fifty years of the psychology of programming. *International Journal of Human-Computer Studies*, 131:52–63, Nov 2019, DOI 10.1016/j.ijhcs.2019.06.009.
Abstract: Abstract This paper reflects on the evolution (past, present and future) of the 'psychology of programming' over the 50 year period of this anniversary issue. The International Journal of Human-Computer Studies (IJHCS) has been a key venue for much seminal work in this field, including its first foundations, and we review the changing research concerns seen in publications over these five decades. We relate this thematic evolution to research taking place over the same period within more specialist communities, especially the Psychology of Programming Interest Group (PPIG), the Empirical Studies of Programming series (ESP), and the ongoing community in Visual Languages and Human-Centric Computing (VL/HCC). Many other communities have interacted with psychology of programming, both influenced by research published within the specialist groups, and in turn influencing research priorities. We end with an overview of the core theories that have been developed over this period, as an introductory resource for new researchers, and also with the authors' own analysis of key priorities for future research.

[**Boag2022**] William Boag, Harini Suresh, Bianca Lepe, and Catherine D'Ignazio. Tech worker organizing for power and accountability. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. ACM, Jun 2022, DOI 10.1145/3531146.3533111.
Abstract: In recent years, there has been a growing interest in the field of "AI Ethics" and related areas. This field is purposefully broad, allowing

for the intersection of numerous subfields and disciplines. However, a lot of work in this area thus far has centered computational methods, leading to a narrow lens where technical tools are framed as solutions for broader sociotechnical problems. In this work, we discuss a less-explored mode of what it can mean to "do" AI Ethics: tech worker collective action. Through collective action, the employees of powerful tech companies can act as a countervailing force against strong corporate impulses to grow or make a profit to the detriment of other values. In this work, we ground these efforts in existing scholarship of social movements and labor organizing. We characterize 150 documented collective actions, and explore several case studies of successful campaigns. Looking forward, we also identify under-explored types of actions, and provide conceptual frameworks and inspiration for how to utilize worker organizing as an effective lever for change.

[**Bogner2022**] Justus Bogner and Manuel Merkel. To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github, 2022.

[**Borg2022**] Markus Borg, Leif Jonsson, Emelie Engström, Béla Bartalos, and Attila Szabó. Adopting automated bug assignment in practice: A longitudinal case study at ericsson, 2022.

[**Britton2017**] Emily Britton, Natalie Simper, Andrew Leger, and Jenn Stephenson. Assessing teamwork in undergraduate education: a measurement tool to evaluate individual teamwork skills. *Assess. Eval. High. Educ.*, 42(3):378–397, Apr 2017, DOI 10.1080/02602938.2015.1116497.
Abstract: Effective teamwork skills are essential for success in an increasingly team-based workplace. However, research suggests that there is often confusion concerning how teamwork is measured and assessed, making it difficult to develop these skills in undergraduate curricula. The goal of the present study was to develop a sustainable tool for assessing individual teamwork skills, with the intention of refining and measuring these skills over time. The TeamUp rubric was selected as the preliminary standardised measure of teamwork and tested in a second year undergraduate course (Phase One). Although the tool displayed acceptable psychometric properties, there was concern that it was too lengthy, compromising student completion. This prompted refinement and modification leading to the development of the Team-Q, which was again tested in the same undergraduate course (Phase Two). The new tool had high internal consistency, as well as conceptual similarity to other measures of teamwork. Estimates of inter-rater reliability were within a satisfactory range, although it was determined that logistical issues limited the feasibility of external evaluations. Preliminary evidence suggests that teamwork skills improve over time when taught and assessed, providing support for the continued application of the Team-Q as a tool for developing teamwork skills in undergraduate education.

[**Brodley2022**] Carla E Brodley, Benjamin J Hescott, Jessica Biron, Ali Ressing, Melissa Peiken, Sarah Maravetz, and Alan Mislove. Broadening par-

ticipation in computing via ubiquitous combined majors (CS+X). In *Proc. Technical Symposium on Computer Science Education (SIGCSE)*. ACM, Feb 2022, DOI 10.1145/3478431.3499352.
Abstract: In 2001, Khoury College of Computer Sciences at Northeastern University created their first combined majors with Cognitive Psychology, Mathematics and Physics. This type of degree has often been referred to as C̈S+Xïn the literature and is increasingly relevant as the need for interdisciplinary computer scientists grows. As of 2021, students at Northeastern can choose among three computing majors (Computer Science, Data Science or Cybersecurity) and 42 combined majors, which combine one of the three computing degrees with one of 29 distinct majors in other fields. Prior to 2014, combined majors were with the sciences, business and design. Over the last seven years, we created 29 new combined majors, explicitly creating combinations with fields where there has traditionally been greater gender diversity. The resulting increase in student interest and gender diversity over the last seven years is compelling. As of Fall 2020, 44.6% of the 2,800+ computing majors at Northeastern are pursuing combined majors, 39% of whom are women. This is substantially higher than the 21.5% reported in IPEDS for 2019 women computing graduates in the U.S. We did not observe any significant differences in racial and ethnic diversity between combined and computing only degrees. In this experience paper, we describe how we create and manage combined majors, and we present results on enrollments, admissions, graduation, internship placements, and how students discover combined majors.

[**Buffardi2020**] Kevin Buffardi. Assessing individual contributions to software engineering projects with Git logs and user stories. In *Proc. Technical Symposium on Computer Science Education (SIGCSE)*. ACM, Feb 2020, DOI 10.1145/3328778.3366948.
Abstract: Software Engineering courses often incorporate large-scale projects with collaboration between students working in teams. However, it is difficult to objectively assess individual students when their projects are a product of collaborative efforts. This study explores measurements of individuals' contributions to their respective teams. I analyzed ten Software Engineering team projects (n=42) and evaluations of individual contributions using automated evaluation of the version control system history (Git logs) and user stories completed on their project management (Kanban) boards. Unique insights from meta-data within the Git history and Kanban board user stories reveal complicated relationships between these measurements and traditional assessments, such as peer review and subjective instructor evaluation. From the results, I suggest supplementing and validating traditional assessments with insights from individuals' commit history and user story contributions.

[**CanovasIzquierdo2022**] Javier Luis Cánovas Izquierdo and Jordi Cabot. On the analysis of non-coding roles in open source development. *Empir. Softw. Eng.*, 27(1), Jan 2022, DOI 10.1007/s10664-021-10061-x.

Abstract: AbstractThe role of non-coding contributors in Open Source Software (OSS) is poorly understood. Most of current research around OSS development focuses on the coding aspects of the project (e.g., commits, pull requests or code reviews) while ignoring the potential of other types of contributions. Often, due to the assumption that these other contributions are not significant in number and that, in any case, they are handled by the same people that are also part of the "coding team". This paper aims to investigate whether this is actually the case by analyzing the frequency and diversity of non-coding contributions in OSS development. As a sample of projects for our study we have taken the 100 most popular projects in the ecosystem of NPM, a package manager for JavaScript. Our results validate the importance of dedicated non-coding contributors in OSS and the diversity of OSS communities as, typically, a contributor specializes in a specific subset of roles. We foresee that projects adopting explicit policies to attract and onboard them could see a positive impact in their long-term sustainability providing they also put in place the right governance strategies to facilitate the migration and collaboration among the different roles. As part of this work, we also provide a replicability package to facilitate further quantitative role-based analysis by other researchers.

[**Chowdhury2022a**] Md Atique Reza Chowdhury, Rabe Abdalkareem, Emad Shihab, and Bram Adams. On the untriviality of trivial packages: An empirical study of npm JavaScript packages. *IEEE Transactions on Software Engineering*, 48(8):2695–2708, Aug 2022, DOI 10.1109/tse.2021.3068901.

[**Chowdhury2022b**] Partha Das Chowdhury, Mohammad Tahaei, and Awais Rashid. Better call saltzer & schroeder: A retrospective security analysis of solarwinds & log4j, 2022.

[**Coleman2022**] Cora Coleman, William G. Griswold, and Nick Mitchell. Do cloud developers prefer clis or web consoles? clis mostly, though it varies by task, 2022.

[**Collaris2022**] Dennis Collaris, Hilde J P Weerts, Daphne Miedema, Jarke J van Wijk, and Mykola Pechenizkiy. Characterizing data scientists' mental models of local feature importance. In *Nordic Human-Computer Interaction Conference*. ACM, Oct 2022, DOI 10.1145/3546155.3546670.

Abstract: Feature importance is an approach that helps to explain machine learning model predictions. It works through assigning importance scores to input features of a particular model. Different techniques exist to derive these scores, with widely varying underlying assumptions of what importance means. Little research has been done to verify whether these assumptions match the expectations of the target user, which is imperative to ensure that feature importance values are not misinterpreted. In this work, we explore data scientists' mental models of (local) feature importance and compare these with the conceptual models of the techniques. We first identify several properties of local feature importance techniques that could potentially lead to misinterpretations. Subsequently, we explore the expectations

data scientists have about local feature importance through an exploratory (qualitative and quantitative) survey of 34 data scientists in industry. We compare the identified expectations to the theory and assumptions behind the techniques and find that the two are not (always) in agreement.

[**Cosden2022**] Ian A. Cosden. An rse group model: Operational and organizational approaches from princeton university's central research software engineering group, 2022.

[**DalSasso2016**] Tommaso Dal Sasso, Andrea Mocci, and Michele Lanza. What makes a satisficing bug report? In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Aug 2016, DOI 10.1109/QRS.2016.28.
Abstract: To ensure quality of software systems, developers use bug reports to track defects. It is in the interest of users and developers that bug reports provide the necessary information to ease the fixing process. Past research found that users do not provide the information that developers deem ideally useful to fix a bug. This raises an interesting question: What is the satisficing information to speed up the bug fixing process? We conducted an observational study on the relation between provided report information and its lifetime, considering more than 650,000 reports from open-source systems using popular bug trackers. We distilled a meta-model for a minimal bug report, establishing a basic layer of core features. We found that few fields influence the resolution time and that customized fields have little impact on it. We performed a survey to investigate what users deem easy to provide in a bug report.

[**DeAlmeida2022**] Eduardo Santana de Almeida, Iftekhar Ahmed, and Andre van der Hoek. Let's go to the whiteboard (again):perceptions from software architects on whiteboard architecture meetings, 2022.

[**DeSantana2022**] Taijara Loiola de Santana, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Iftekhar Ahmed. Bug analysis in jupyter notebook projects: An empirical study, 2022.

[**DiGrazia2022**] Luca Di Grazia and Michael Pradel. The evolution of type annotations in Python: an empirical study. In *Proc. European Software Engineering Conference/International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov 2022, DOI 10.1145/3540250.3549114.
Abstract: Type annotations and gradual type checkers attempt to reveal errors and facilitate maintenance in dynamically typed programming languages. Despite the availability of these features and tools, it is currently unclear how quickly developers are adopting them, what strategies they follow when doing so, and whether adding type annotations reveals more type errors. This paper presents the first large-scale empirical study of the evolution of type annotations and type errors in Python. The study is based on an analysis of 1,414,936 type annotation changes, which we extract from

1,123,393 commits among 9,655 projects. Our results show that (i) type annotations are getting more popular, and once added, often remain unchanged in the projects for a long time, (ii) projects follow three evolution patterns for type annotation usage – regular annotation, type sprints, and occasional uses – and that the used pattern correlates with the number of contributors, (iii) more type annotations help find more type errors (0.704 correlation), but nevertheless, many commits (78.3%) are committed despite having such errors. Our findings show that better developer training and automated techniques for adding type annotations are needed, as most code still remains unannotated, and they call for a better integration of gradual type checking into the development process.

[**Dias2021**] Edson Dias, Paulo Meirelles, Fernando Castor, Igor Steinmacher, Igor Wiese, and Gustavo Pinto. What makes a great maintainer of open source projects? In *Proc. International Conference on Software Engineering (ICSE)*. IEEE, May 2021, DOI 10.1109/icse43902.2021.00093.
Abstract: Although Open Source Software (OSS) maintainers devote a significant proportion of their work to coding tasks, great maintainers must excel in many other activities beyond coding. Maintainers should care about fostering a community, helping new members to find their place, while also saying "no" to patches that although are well-coded and well-tested, do not contribute to the goal of the project. To perform all these activities masterfully, maintainers should exercise attributes that software engineers (working on closed source projects) do not always need to master. This paper aims to uncover, relate, and prioritize the unique attributes that great OSS maintainers might have. To achieve this goal, we conducted 33 semi-structured interviews with well-experienced maintainers that are the gatekeepers of notable projects such as the Linux Kernel, the Debian operating system, and the GitLab coding platform. After we analyzed the interviews and curated a list of attributes, we created a conceptual framework to explain how these attributes are connected. We then conducted a rating survey with 90 OSS contributors. We noted that "technical excellence" and "communication" are the most recurring attributes. When grouped, these attributes fit into four broad categories: management, social, technical, and personality. While we noted that "sustain a long term vision of the project" and being "extremely careful" seem to form the basis of our framework, we noted through our survey that the communication attribute was perceived as the most essential one.

[**Diercks2022**] Philipp Diercks, Dennis Gläser, Ontje Lünsdorf, Michael Selzer, Bernd Flemisch, and Jörg F. Unger. Evaluation of tools for describing, reproducing and reusing scientific workflows, 2022.

[**Dogan2022**] Emre Doğan and Eray Tüzün. Towards a taxonomy of code review smells. *Inf. Softw. Technol.*, 142(106737):106737, Feb 2022, DOI 10.1016/j.infsof.2021.106737.

[**Drage2022**] Eleanor Drage and Kerry Mackereth. Does AI debias recruitment? race, gender, and AI's "eradication of difference". *Philos. Technol.*, 35(4):89, Oct 2022, DOI 10.1007/s13347-022-00543-1.

Abstract: In this paper, we analyze two key claims offered by recruitment AI companies in relation to the development and deployment of AI-powered HR tools: (1) recruitment AI can objectively assess candidates by removing gender and race from their systems, and (2) this removal of gender and race will make recruitment fairer, help customers attain their DEI goals, and lay the foundations for a truly meritocratic culture to thrive within an organization. We argue that these claims are misleading for four reasons: First, attempts to "strip" gender and race from AI systems often misunderstand what gender and race are, casting them as isolatable attributes rather than broader systems of power. Second, the attempted outsourcing of "diversity work" to AI-powered hiring tools may unintentionally entrench cultures of problems within organizations. Third, AI hiring tools' supposedly neutral assessment of candidates' traits belie the power relationship between the observer and the observed. Specifically, the racialized history of character analysis and its associated processes of classification and categorization play into longer histories of taxonomical sorting and reflect the current demands and desires of the job market, even when not explicitly conducted along the lines of gender and race. Fourth, recruitment AI tools help produce the "ideal candidate" that they supposedly identify through by constructing associations between words and people's bodies. From these four conclusions outlined above, we offer three key recommendations to AI HR firms, their customers, and policy makers going forward.

[**Dyer2022**] Robert Dyer and Jigyasa Chauhan. An exploratory study on the predominant programming paradigms in Python code. In *Proc. European Software Engineering Conference/International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov 2022, DOI 10.1145/3540250.3549158.

Abstract: Python is a multi-paradigm programming language that fully supports object-oriented (OO) programming. The language allows writing code in a non-procedural imperative manner, using procedures, using classes, or in a functional style. To date, no one has studied what paradigm(s), if any, are predominant in Python code and projects. In this work, we first define a technique to classify Python files into predominant paradigm(s). We then automate our approach and evaluate it against human judgements, showing over 80% agreement. We then analyze over 100k open-source Python projects, automatically classifying each source file and investigating the paradigm distributions. The results indicate Python developers tend to heavily favor OO features. We also observed a positive correlation between OO and procedural paradigms and the size of the project. And despite few files or projects being predominantly functional, we still found many functional feature uses.

[**Dykstra2022**] Josiah Dykstra, Kelly Shortridge, Jamie Met, and Douglas

Hough. Sludge for good: Slowing and imposing costs on cyber attackers, 2022.

[**Etemadi2022**] Khashayar Etemadi, Aman Sharma, Fernanda Madeiral, and Martin Monperrus. Augmenting diffs with runtime information, 2022.

[**Farzat2021**] Fabio de A. Farzat, Marcio de O. Barros, and Guilherme H. Travassos. Evolving JavaScript code to reduce load time. *IEEE Transactions on Software Engineering*, 47(8):1544–1558, Aug 2021, DOI `10.1109/tse.2019.2928293`.
Abstract: JavaScript is one of the most used programming languages for front-end development of Web applications. The increase in complexity of front-end features brings concerns about performance, especially the load and execution time of JavaScript code. In this paper, we propose an evolutionary program improvement technique to reduce the size of JavaScript programs and, therefore, the time required to load and execute them in Web applications. To guide the development of this technique, we performed an experimental study to characterize the patches applied to JavaScript programs to reduce their size while keeping the functionality required to pass all test cases in their test suites. We applied this technique to 19 JavaScript programs varying from 92 to 15,602 LOC and observed reductions from 0.2 to 73.8 percent of the original code, as well as a relationship between the quality of a program's test suite and the ability to reduce the size of its source code.

[**Feal2020**] Álvaro Feal, Paolo Calciati, Narseo Vallina-Rodriguez, Carmela Troncoso, and Alessandra Gorla. Angel or devil? a privacy study of mobile parental control apps. *Proceedings on Privacy Enhancing Technologies*, 2020(2):314–335, Apr 2020, DOI `10.2478/popets-2020-0029`.
Abstract: Android parental control applications are used by parents to monitor and limit their children's mobile behaviour (e.g., mobile apps usage, web browsing, calling, and texting). In order to offer this service, parental control apps require privileged access to sys-tem resources and access to sensitive data. This may significantly reduce the dangers associated with kids' online activities, but it raises important privacy con-cerns. These concerns have so far been overlooked by organizations providing recommendations regarding the use of parental control applications to the public. We conduct the first in-depth study of the Android parental control app's ecosystem from a privacy and regulatory point of view. We exhaustively study 46 apps from 43 developers which have a combined 20M installs in the Google Play Store. Using a combination of static and dynamic analysis we find that: these apps are on average more permissions-hungry than the top 150 apps in the Google Play Store, and tend to request more dangerous permissions with new releases; 11% of the apps transmit personal data in the clear; 34% of the apps gather and send personal information without appropriate consent; and 72% of the apps share data with third parties (including online advertising and analytics services) without mentioning their presence in their privacy

policies. In summary, parental control applications lack transparency and lack compliance with reg ulatory requirements. This holds even for those applications recommended by European and other national security centers.

[**Ferreira2022**] Fabio Ferreira, Hudson Silva Borges, and Marco Tulio Valente. On the (un-)adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, 52(4):947–966, Oct 2021, DOI 10.1002/spe.3044.
Abstract: JavaScript is characterized by a rich ecosystem of libraries and frameworks. A key element in this ecosystem are frameworks used for implementing the front-end of web-based applications, such as Vue and React. However, despite their relevance, we have few works investigating the factors that drive the adoption—and un-adoption—of front-end-based JavaScript frameworks. Therefore, in this article, we first report the results of a survey with 49 developers where we asked them to describe the factors they consider when selecting a front-end framework. In the second part of the work, we focus on projects that migrate from one framework to another since JavaScript's ecosystem is also very dynamic. Finally, we provide a quantitative characterization of the migration effort and reveal the main barriers faced by the developers during this effort. Although not completely generalizable, our central findings are as follows: (a) popularity and learnability are the key factors that motivate the choice of front-end frameworks in JavaScript; (b) from the 49 surveyed developers, one out of four have plans to migrate to another framework in the future; (c) the time spent performing the migration is greater than or equal to the time spent using the old framework in all studied projects. We conclude with a list of implications for practitioners, framework developers, tool builders, and researchers.

[**FinnieAnsley2022**] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of OpenAI codex on introductory programming. In *Australasian Computing Education Conference*. ACM, Feb 2022, DOI 10.1145/3511861.3511863.

[**Flyvbjerg2022**] Bent Flyvbjerg, Alexander Budzier, Jong Seok Lee, Mark Keil, Daniel Lunn, and Dirk W Bester. The empirical reality of IT project cost overruns: Discovering a power-law distribution. *J. Manag. Inf. Syst.*, 39(3):607–639, Jul 2022, DOI 10.1080/07421222.2022.2096544.
Abstract: ABSTRACT If managers assume a normal or near-normal distribution of Information Technology (IT) project cost overruns, as is common, and cost overruns can be shown to follow a power-law distribution, managers may be unwittingly exposing their organizations to extreme risk by severely underestimating the probability of large cost overruns. In this research, we collect and analyze a large sample comprised of 5,392 IT projects to empirically examine the probability distribution of IT project cost overruns. Further, we propose and examine a mechanism that can explain such a distribution. Our results reveal that IT projects are far riskier in terms of cost than normally assumed by decision makers and scholars. Specifically,

we found that IT project cost overruns follow a power-law distribution in which there are a large number of projects with relatively small overruns and a fat tail that includes a smaller number of projects with extreme overruns. A possible generative mechanism for the identified power-law distribution is found in interdependencies among technological components in IT systems. We propose and demonstrate, through computer simulation, that a problem in a single technological component can lead to chain reactions in which other interdependent components are affected, causing substantial overruns. What the power law tells us is that extreme IT project cost overruns will occur and that the prevalence of these will be grossly underestimated if managers assume that overruns follow a normal or near-normal distribution. This underscores the importance of realistically assessing and mitigating the cost risk of new IT projects up front.

[**Foidl2022**] Harald Foidl, Michael Felderer, and Rudolf Ramler. Data smells. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. ACM, May 2022, DOI 10.1145/3522664.3528590.
Abstract: High data quality is fundamental for today's AI-based systems. However, although data quality has been an object of research for decades, there is a clear lack of research on potential data quality issues (e.g., ambiguous, extraneous values). These kinds of issues are latent in nature and thus often not obvious. Nevertheless, they can be associated with an increased risk of future problems in AI-based systems (e.g., technical debt, data-induced faults). As a counterpart to code smells in software engineering, we refer to such issues as Data Smells. This article conceptualizes data smells and elaborates on their causes, consequences, detection, and use in the context of AI-based systems. In addition, a catalogue of 36 data smells divided into three categories (i.e., Believability Smells, Understandability Smells, Consistency Smells) is presented. Moreover, the article outlines tool support for detecting data smells and presents the result of an initial smell detection on more than 240 real-world datasets.

[**Fregnan2023**] Enrico Fregnan, Josua Fröhlich, Davide Spadini, and Alberto Bacchelli. Graph-based visualization of merge requests for code review. *Journal of Systems and Software*, 195:111506, Jan 2023, DOI 10.1016/j.jss.2022.111506.

[**Friend2022**] Michelle Friend, Monica Mcgill, and Anni Reinking. Solve this! K-12 CS education teachers' problems of practice. In *Koli Calling '22: 22nd Koli Calling International Conference on Computing Education Research*. ACM, Nov 2022, DOI 10.1145/3564721.3564738.
Abstract: Problem. Educational research identifies answerable questions, but often does not address the problems K-12 teachers identify as important. Further, academic research findings can be difficult for teachers to apply to their practices and unique contexts. Currently, little research exists on the lived experiences of primary and secondary instructors who teach computer science (CS) or computational thinking (CT) and also on the specific

problems of practice teachers face when teaching CS. Research Question. What problems of practice do K-12 teachers face when teaching CS/CT? Method. Data for this qualitative study was collected using an online questionnaire distributed to teachers internationally. CS/CT teachers responded to an open-ended prompt asking for problems related to teaching CS. The data was analyzed using descriptive first-round coding and focused second-round coding. Validity was established through collaborative coding. Analysis was theorized using locus of control. Findings. Problems with students encompassed behavioral, cognitive, and attitudinal issues, as well as lack of home support or resources. Teachers identified many problems of policy notably stemming from lack of resources or support from administrators. A smaller number of challenges, such as lack of content knowledge, were situated within teachers themselves. While some problems such as student motivation are general, a number of responses identified unique challenges in CS education compared to other disciplines. Implications. Identifying problems faced by teachers can guide professional development offerings, help researchers develop studies that would result in meaningful improvement to CS education, and suggest policy decisions which would result in better outcomes for students.

[**Gaffney2022**] Kevin P Gaffney, Martin Prammer, Larry Brasfield, D Richard Hipp, Dan Kennedy, and Jignesh M Patel. SQLite. *Proceedings VLDB Endowment*, 15(12):3535–3547, Aug 2022, DOI 10.14778/3554821.3554842.
Abstract: In the two decades following its initial release, SQLite has become the most widely deployed database engine in existence. Today, SQLite is found in nearly every smartphone, computer, web browser, television, and automobile. Several factors are likely responsible for its ubiquity, including its in-process design, standalone codebase, extensive test suite, and cross-platform file format. While it supports complex analytical queries, SQLite is primarily designed for fast online transaction processing (OLTP), employing row-oriented execution and a B-tree storage format. However, fueled by the rise of edge computing and data science, there is a growing need for efficient in-process online analytical processing (OLAP). DuckDB, a database engine nicknamed "the SQLite for analytics", has recently emerged to meet this demand. While DuckDB has shown strong performance on OLAP benchmarks, it is unclear how SQLite compares. Furthermore, we are aware of no work that attempts to identify root causes for SQLite's performance behavior on OLAP workloads. In this paper, we discuss SQLite in the context of this changing workload landscape. We describe how SQLite evolved from its humble beginnings to the full-featured database engine it is today. We evaluate the performance of modern SQLite on three benchmarks, each representing a different flavor of in-process data management, including transactional, analytical, and blob processing. We delve into analytical data processing on SQLite, identifying key bottlenecks and weighing potential solutions. As a result of our optimizations, SQLite is now up to 4.2X faster on SSB. Finally, we discuss the future of SQLite, envisioning how it will evolve to meet new

demands and challenges.

[**Galappaththi2022**] Akalanka Galappaththi, Sarah Nadi, and Christoph Treude. Does this apply to me? In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3528435.

Abstract: Stack Overflow has become an essential technical resource for developers. However, given the vast amount of knowledge available on Stack Overflow, finding the right information that is relevant for a given task is still challenging, especially when a developer is looking for a solution that applies to their specific requirements or technology stack. Clearly marking answers with their technical context, i.e., the information that characterizes the technologies and assumptions needed for this answer, is potentially one way to improve navigation. However, there is no information about how often such context is mentioned, and what kind of information it might offer. In this paper, we conduct an empirical study to understand the occurrence of technical context in Stack Overflow answers and comments, using tags as a proxy for technical context. We specifically focus on additional context, where answers/comments mention information that is not already discussed in the question. Our results show that nearly half of our studied threads contain at least one additional context. We find that almost 50% of the additional context are either a library/framework, a programming language, a tool/application, an API, or a database. Overall, our findings show the promise of using additional context as navigational cues.

[**Gamblin2022**] Todd Gamblin, Massimiliano Culpo, Gregory Becker, and Sergei Shudler. Using answer set programming for hpc dependency solving, 2022.

[**Getseva2022**] Vanesa Getseva and Amruth N Kumar. An empirical analysis of code-tracing concepts. In *Proc. Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Jul 2022, DOI 10.1145/3502718.3524794.

Abstract: Which code-tracing concepts are introductory programming students likely to learn from classroom instruction and which ones need additional problem-solving practice to master? Are there relationships among programming concepts that can be used to build adaptive assessment instruments? To answer these questions, we analyzed the data collected over several semesters by a suite of code-tracing tutors called problets, that administered pre-test, practice, post-test protocol. Each tutor covered a single programming topic, which consisted of 9-25 concepts. For each concept, we used the pretest data to calculate the probability that students knew the concept before using the tutor. Using a weighted average of the concept probabilities, we found that students had learned some topics more than others: if/if-else (0.85), function behavior (0.76), arrays (0.73), while (0.7), for (0.69), switch (0.67), and debugging functions (0.55). Some of the concepts on which students needed additional practice included bugs, nested

loops and back-to-back loops. Expressions, even when used in novel contexts, were not challenging for students. We built a Bayesian network for each topic based on conditional probabilities to discover the concepts that must be covered, and those whose coverage is redundant in the presence of other concepts. A strength of this empirical study is that it uses a large dataset collected from multiple institutions over multiple semesters. We also list threats to the validity of the study.

[**Gorz2022**] Philipp Görz, Björn Mathis, Keno Hassler, Emre Güler, Thorsten Holz, Andreas Zeller, and Rahul Gopinath. How to compare fuzzers, 2022.

[**Graziotin2022**] Daniel Graziotin, Per Lenberg, Robert Feldt, and Stefan Wagner. Psychometrics in behavioral software engineering: A methodological introduction with guidelines. *ACM Trans. Softw. Eng. Methodol.*, 31(1):1–36, Jan 2022, DOI 10.1145/3469888.
Abstract: A meaningful and deep understanding of the human aspects of software engineering (SE) requires psychological constructs to be considered. Psychology theory can facilitate the systematic and sound development as well as the adoption of instruments (e.g., psychological tests, questionnaires) to assess these constructs. In particular, to ensure high quality, the psychometric properties of instruments need evaluation. In this article, we provide an introduction to psychometric theory for the evaluation of measurement instruments for SE researchers. We present guidelines that enable using existing instruments and developing new ones adequately. We conducted a comprehensive review of the psychology literature framed by the Standards for Educational and Psychological Testing. We detail activities used when operationalizing new psychological constructs, such as item pooling, item review, pilot testing, item analysis, factor analysis, statistical property of items, reliability, validity, and fairness in testing and test bias. We provide an openly available example of a psychometric evaluation based on our guideline. We hope to encourage a culture change in SE research towards the adoption of established methods from psychology. To improve the quality of behavioral research in SE, studies focusing on introducing, validating, and then using psychometric instruments need to be more common.

[**Grotov2022**] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. A large-scale comparison of python code in jupyter notebooks and scripts, 2022.

[**Haduong2019**] Paulina Haduong. "i like computers. I hate coding": a portrait of two teens' experiences. *Inf. Learn. Sci.*, 120(5/6):349–365, May 2019, DOI 10.1108/ILS-05-2018-0037.
Abstract: Purpose Some empirical evidence suggests that historically marginalized young people may enter introductory programming experiences with skepticism or reluctance, because of negative perceptions of the computing field. This paper aims to explore how learner identity and motivation can affect their experiences in an introductory computer science (CS) experience, particularly for young people who have some prior experience with

computing. In this program, learners were asked to develop digital media artifacts about civic issues using Scratch, a block-based programming language. Design/methodology/approach Through participant observation as a teacher and designer of the course, artifact analysis of student-generated computer programs and design journals, as well as with two follow-up 1-h interviews, the author used the qualitative method of portraiture to examine how two reluctant learners experienced a six-week introductory CS program. Findings These learners' experiences illuminate the ways in which identity, community and competence can play a role in supporting learner motivation in CS education experiences. Research limitations/implications As more students have multiple introductory computing encounters, educators need to take into account not only their perceptions of the computing field more broadly but also specific prior encounters with programming. Because of the chosen research approach, the research results may lack generalizability. Researchers are encouraged to explore other contexts and examples further. Practical implications This portrait highlights the need for researchers and educators to take into account student motivation in the design of learning environments. Originality/value This portrait offers a novel examination of novice programmer experiences through the choice in method, as well as new examples of how learner identity can affect student motivation.

[**Hartel2022**] Johannes Härtel and Ralf Lämmel. Operationalizing threats to MSR studies by simulation-based testing. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3527960.

Abstract: Quantitative studies on the border between Mining Software Repository (MSR) and Empirical Software Engineering (ESE) apply data analysis methods, like regression modeling, statistic tests or correlation analysis, to commits or pulls to better understand the software development process. Such studies assure the validity of the reported results by following a sound methodology. However, with increasing complexity, parts of the methodology can still go wrong. This may result in MSR/ESE studies with undetected threats to validity. In this paper, we propose to systematically protect against threats by operationalizing their treatment using simulations. A simulation substitutes observed and unobserved data, related to an MSR/ESE scenario, with synthetic data, carefully defined according to plausible assumptions on the scenario. Within a simulation, unobserved data becomes transparent, which is the key difference to a real study, necessary to detect threats to an analysis methodology. Running an analysis methodology on synthetic data may detect basic technical bugs and misinterpretations, but it also improves the trust in the methodology. The contribution of a simulation is to operationalize testing the impact of important assumptions. Assumptions still need to be rated for plausibility. We evaluate simulation-based testing by operationalizing undetected threats in the context of four published MSR/ESE studies. We recommend that future research uses such more systematic treatment of threats, as a contribution against the repro-

ducibility crisis.

[**Head2020**] Andrew Head, Jason Jiang, James Smith, Marti A Hearst, and Björn Hartmann. Composing flexibly-organized step-by-step tutorials from linked source code, snippets, and outputs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Apr 2020, DOI 10.1145/3313831.3376798.

Abstract: Programming tutorials are a pervasive, versatile medium for teaching programming. In this paper, we report on the content and structure of programming tutorials, the pain points authors experience in writing them, and a design for a tool to help improve this process. An interview study with 12 experienced tutorial authors found that they construct documents by interleaving code snippets with text and illustrative outputs. It also revealed that authors must often keep related artifacts of source programs, snippets, and outputs consistent as a program evolves. A content analysis of 200 frequently-referenced tutorials on the web also found that most tutorials contain related artifacts—duplicate code and outputs generated from snippets—that an author would need to keep consistent with each other. To address these needs, we designed a tool called Torii with novel authoring capabilities. An in-lab study showed that tutorial authors can successfully use the tool for the unique affordances identified, and provides guidance for designing future tools for tutorial authoring.

[**Hellman2022**] Jazlyn Hellman, Jiahao Chen, Md Sami Uddin, Jinghui Cheng, and Jin L C Guo. Characterizing user behaviors in open-source software user forums. In *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*. ACM, May 2022, DOI 10.1145/3528579.3529178.

Abstract: User forums of Open Source Software (OSS) enable end-users to collaboratively discuss problems concerning the OSS applications. Despite decades of research on OSS, we know very little about how end-users engage with OSS communities on these forums, in particular, the challenges that hinder their continuous and meaningful participation in the OSS community. Many previous works are developer-centric and overlook the importance of end-user forums. As a result, end-users' expectations are seldom reflected in OSS development. To better understand user behaviors in OSS user forums, we carried out an empirical study analyzing about 1.3 million posts from user forums of four popular OSS applications: Zotero, Audacity, VLC, and RStudio. Through analyzing the contribution patterns of three common user types (end-users, developers, and organizers), we observed that end-users not only initiated most of the threads (above 96% of threads in three projects, 86% in the other), but also acted as the significant contributors for responding to other users' posts, even though they tended to lack confidence in their activities as indicated by psycho-linguistic analyses. Moreover, we found end-users more open, reflecting a more positive emotion in communication than organizers and developers in the forums. Our work contributes new knowledge about end-users' activities and behaviors in OSS user forums

that the vital OSS stakeholders can leverage to improve end-user engagement in the OSS development process.

[**Hidellaarachchi2022**] Dulaji Hidellaarachchi, John Grundy, Rashina Hoda, and Ingo Mueller. Does personality impact requirements engineering activities?, 2022.

[**Hoda2021**] Rashina Hoda. Socio-technical grounded theory for software engineering. *IEEE Transactions on Software Engineering*, pages 1–1, 2021, DOI 10.1109/tse.2021.3106280.
Abstract: Grounded Theory (GT), a sociological research method designed to study social phenomena, is increasingly being used to investigate the human and social aspects of software engineering (SE). However, being written by and for sociologists, GT is often challenging for a majority of SE researchers to understand and apply. Additionally, SE researchers attempting ad hoc adaptations of traditional GT guidelines for modern socio-technical (ST) contexts often struggle in the absence of clear and relevant guidelines to do so, resulting in poor quality studies. To overcome these research community challenges and leverage modern research opportunities, this paper presents Socio-Technical Grounded Theory (STGT) designed to ease application and achieve quality outcomes. It defines what exactly is meant by an ST research context and presents the STGT guidelines that expand GT's philosophical foundations, provide increased clarity and flexibility in its methodological steps and procedures, define possible scope and contexts of application, encourage frequent reporting of a variety of interim, preliminary, and mature outcomes, and introduce nuanced evaluation guidelines for different outcomes. It is hoped that the SE research community and related ST disciplines such as computer science, data science, artificial intelligence, information systems, human computer/robot/AI interaction, human-centered emerging technologies (and increasingly other disciplines being transformed by rapid digitalisation and AI-based augmentation), will benefit from applying STGT to conduct quality research studies and systematically produce rich findings and mature theories with confidence.

[**Hundhausen2022**] C D Hundhausen, P T Conrad, A S Carter, and O Adesope. Assessing individual contributions to software engineering projects: a replication study. *Comput. Sci. Educ.*, 32(3):335–354, Jul 2022, DOI 10.1080/08993408.2022.2071543.
Abstract: ABSTRACT Background and Context Assessing team members' indivdiual contributions to software development projects poses a key problem for computing instructors. While instructors typically rely on subjective assessments, objective assessments could provide a more robust picture. To explore this possibility, In a 2020 paper, Buffardi presented a correlational analysis of objective metrics and subjective metrics in an advanced software engineering project course (n= 41 students and 10 teams), finding only two significant correlations. Objective To explore the robustness of Buffardi's findings and gain further insight, we conducted a larger scale replication of

the Buffardi study (n = 118 students and 25 teams) in three courses at three institutions. Method We collected the same data as in the Buffardi study and computed the same measures from those data. We replicated Buffardi's exploratory, correlational and regression analyses of objective and subjective measures. Findings While replicating four of Buffardi's five significant correlational findings and partially replicating the findings of Buffardi's regression analyses, our results go beyond those of Buffardi by identifying eight additional significant correlations. Implications In contrast to Buffardi's study, our larger scale study suggests that subjective and objective measures of individual performance in team software development projects can be fruitfully combined to provide consistent and complementary assessments of individual performance.

[**Huszar2022**] Ferenc Huszár, Sofia Ira Ktena, Conor O'Brien, Luca Belli, Andrew Schlaikjer, and Moritz Hardt. Algorithmic amplification of politics on twitter. *Proc. Natl. Acad. Sci. U. S. A.*, 119(1):e2025334119, Jan 2022, DOI `10.1073/pnas.2025334119`.

Abstract: Significance The role of social media in political discourse has been the topic of intense scholarly and public debate. Politicians and commentators from all sides allege that Twitter's algorithms amplify their opponents' voices, or silence theirs. Policy makers and researchers have thus called for increased transparency on how algorithms influence exposure to political content on the platform. Based on a massive-scale experiment involving millions of Twitter users, a fine-grained analysis of political parties in seven countries, and 6.2 million news articles shared in the United States, this study carries out the most comprehensive audit of an algorithmic recommender system and its effects on political content. Results unveil that the political right enjoys higher amplification compared to the political left. Content on Twitter's home timeline is selected and ordered by personalization algorithms. By consistently ranking certain content higher, these algorithms may amplify some messages while reducing the visibility of others. There's been intense public and scholarly debate about the possibility that some political groups benefit more from algorithmic amplification than others. We provide quantitative evidence from a long-running, massive-scale randomized experiment on the Twitter platform that committed a randomized control group including nearly 2 million daily active accounts to a reverse-chronological content feed free of algorithmic personalization. We present two sets of findings. First, we studied tweets by elected legislators from major political parties in seven countries. Our results reveal a remarkably consistent trend: In six out of seven countries studied, the mainstream political right enjoys higher algorithmic amplification than the mainstream political left. Consistent with this overall trend, our second set of findings studying the US media landscape revealed that algorithmic amplification favors right-leaning news sources. We further looked at whether algorithms amplify far-left and far-right political groups more than moderate ones; contrary to prevailing public belief, we did not find evidence to support this hypothesis.

We hope our findings will contribute to an evidence-based debate on the role personalization algorithms play in shaping political content consumption.

[**Idowu2022**] Samuel Idowu, Daniel Strüber, and Thorsten Berger. Asset management in machine learning: State-of-research and state-of-practice. *ACM Comput. Surv.*, Jun 2022, DOI 10.1145/3543847.
Abstract: Machine learning components are essential for today's software systems, causing a need to adapt traditional software engineering practices when developing machine-learning-based systems. This need is pronounced due to many development-related challenges of machine learning components such as asset, experiment, and dependency management. Recently, many asset management tools addressing these challenges have become available. It is essential to understand the support such tools offer to facilitate research and practice on building new management tools with native supports for machine learning and software engineering assets. This article positions machine learning asset management as a discipline that provides improved methods and tools for performing operations on machine learning assets. We present a feature-based survey of 18 state-of-practice and 12 state-of-research tools supporting machine-learning asset management. We overview their features for managing the types of assets used in machine learning experiments. Most state-of-research tools focus on tracking, exploring, and retrieving assets to address development concerns such as reproducibility, while the state-of-practice tools also offer collaboration and workflow-execution-related operations. In addition, assets are primarily tracked intrusively from the source code through APIs and managed via web dashboards or command-line interfaces. We identify asynchronous collaboration and asset reusability as directions for new tools and techniques.

[**Imam2021**] Ahmed Imam and Tapajit Dey. Tracking hackathon code creation and reuse. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, May 2021, DOI 10.1109/msr52588.2021.00085.
Abstract: Background: Hackathons have become popular events for teams to collaborate on projects and develop software prototypes. Most existing research focuses on activities during an event with limited attention to the evolution of the code brought to or created during a hackathon. Aim: We aim to understand the evolution of hackathon-related code, specifically, how much hackathon teams rely on pre-existing code or how much new code they develop during a hackathon. Moreover, we aim to understand if and where that code gets reused. Method: We collected information about 22,183 hackathon projects from Devpost—a hackathon database—and obtained related code (blobs), authors, and project characteristics from the World of Code. We investigated if code blobs in hackathon projects were created before, during, or after an event by identifying the original blob creation date and author, and also checked if the original author was a hackathon project member. We tracked code reuse by first identifying all commits containing blobs created during an event before determining all projects that contain those commits.

Result: While only approximately 9.14% of the code blobs are created during hackathons, this amount is still significant considering time and member constraints of such events. Approximately a third of these code blobs get reused in other projects. Conclusion: Our study demonstrates to what extent pre-existing code is used and new code is created during a hackathon and how much of it is reused elsewhere afterwards. Our findings help to better understand code reuse as a phenomenon and the role of hackathons in this context and can serve as a starting point for further studies in this area.

[**Jeffries2022**] Bryn Jeffries, Jung A Lee, and Irena Koprinska. 115 ways not to say hello, world! In *Proc. Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Jul 2022, DOI 10.1145/3502718.3524809.

Abstract: Online programming courses can provide detailed automatic feedback for code that fails to meet various test conditions, but novice students often struggle with syntax errors and are unable to write valid testable code. Even for very simple exercises, the range of incorrect code can be surprising to educators with mastery of a programming language. This research paper presents an analysis of the error messages from code run by students in an introductory Python 3 programming course, participated in by 8680 primary and high-school students from 680 institutions. The invalid programs demonstrate a wide diversity of mistakes: even for a one-line Ḧello World!" exercise there were 115 unique invalid programs. The most common errors are identified and compared to the topics introduced in the course. The most generic errors in selected exercises are investigated in greater detail to understand the underlying causes. While the majority of students attempting an exercise reach a successful outcome, many students encounter at least one error in their code. Of these, many such errors indicate basic mistakes, such as unquoted string literals, even in exercises late in the course for which some proficiency of earlier concepts is assumed. These observations suggest there is significant scope to provide greater reinforcement of students' understanding of earlier concepts.

[**Joblin2022**] Mitchell Joblin and Sven Apel. How do successful and failed projects differ? a socio-technical analysis. *ACM Trans. Softw. Eng. Methodol.*, Feb 2022, DOI 10.1145/3504003.

Abstract: Software development is at the intersection of the social realm , involving people who develop the software, and the technical realm , involving artifacts (code, docs, etc.) that are being produced. It has been shown that a socio-technical perspective provides rich information about the state of a software project. In particular, we are interested in socio-technical factors that are associated with project success . For this purpose, we frame the task as a network classification problem. We show how a set of heterogeneous networks composed of social and technical entities can be jointly embedded in a single vector space enabling mathematically sound comparisons between distinct software projects. Our approach is specifically designed using intuitive

metrics stemming from network analysis and statistics to ease the interpretation of results in the context of software engineering wisdom. Based on a selection of 32 open-source projects, we perform an empirical study to validate our approach considering three prediction scenarios to test the classification model's ability generalizing to: (1) randomly held-out project snapshots, (2) future project states, and (3) entirely new projects. Our results provide evidence that a socio-technical perspective is superior to a pure social or technical perspective when it comes to early indicators of future project success. To our surprise, the methodology proposed here even shows evidence of being able to generalize to entirely novel (project hold-out set) software projects reaching predication accuracies of 80%, which is a further testament to the efficacy of our approach and beyond what has been possible so far. In addition, we identify key features that are strongly associated with project success. Our results indicate that even relatively simple socio-technical networks capture highly relevant and interpretable information about the early indicators of future project success.

[**Johnson2016**] Philip Johnson, Dan Port, and Emily Hill. An athletic approach to software engineering education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. IEEE, Apr 2016, DOI `10.1109/cseet.2016.29`.
Abstract: We present our findings after two years of experience involving three instructors using an "athletic" approach to software engineering education (AthSE). Co-author Johnson developed AthSE in 2013 to address issues he experienced teaching graduate and undergraduate software engineering. Co-authors Port and Hill subsequently adapted the original approach to their own software courses. AthSE is a pedagogy in which the course is organized into a series of skills to be mastered. For each skill, students are given practice "Workouts" along with videos showing the instructor performing the Workout both correctly and quickly. Unlike traditional home-work assignments, students are advised to repeat the Workout not only until they can complete it correctly, but also as quickly as the instructor. In this experience report we investigate the following question: how can software engineering education be redesigned as an athletic endeavor, and will this provide more efficient and effective learning among students and more rapidly lead them to greater competency and confidence?

[**Kacsmar2022**] Bailey Kacsmar. Improving interactive instruction: Faculty engagement requires starting small and telling all. In *Koli Calling '22: 22nd Koli Calling International Conference on Computing Education Research*. ACM, Nov 2022, DOI `10.1145/3564721.3564739`.
Abstract: Interactive instruction, such as student-centered learning or active learning, is known to benefit student success as well as diversity in computer science. However, there is a persistent and substantial dissonance between research and practice of computer science education techniques. Current research on computer science education, while extensive, sees limited adoption beyond the original researchers. The developed educational

technologies can lack sufficient detail for replication or be too specific and require extensive reworking to be employable by other instructors. Furthermore, instructors face barriers to adopting interactive techniques within their classroom due to student reception, resources, and awareness. We argue that the advancement of computer science education, in terms of propagation and sustainability of student-centered teaching, requires guided approaches for incremental instructional changes as opposed to revolutionary pedagogy. This requires the prioritization of lightweight techniques that can fit within existing lecture formats to enable instructors to overcome barriers hindering the adoption of interactive techniques. Furthermore, such techniques and innovations must be documented in the form of computing education research artifacts, building upon the practices of software artifacts.

[**Khan2022**] Faizan Khan, Boqi Chen, Daniel Varro, and Shane McIntosh. An empirical study of type-related defects in python projects. *IEEE Transactions on Software Engineering*, 48(8):3145–3158, Aug 2022, DOI 10.1109/tse.2021.3082068.

[**Kohavi2009**] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. Controlled experiments on the web: survey and practical guide. *Data Min. Knowl. Discov.*, 18(1):140–181, Feb 2009, DOI 10.1007/s10618-008-0114-1.

[**Kudrjavets2022**] Gunnar Kudrjavets, Nachiappan Nagappan, and Ayushi Rastogi. Do small code changes merge faster? a multi-language empirical investigation, 2022, DOI 10.1145/3524842.3528448.

[**Kuhrmann2022**] Marco Kuhrmann, Paolo Tell, Regina Hebig, Jil Klunder, Jurgen Munch, Oliver Linssen, Dietmar Pfahl, Michael Felderer, Christian R. Prause, Stephen G. MacDonell, Joyce Nakatumba-Nabende, David Raffo, Sarah Beecham, Eray Tuzun, Gustavo Lopez, Nicolas Paez, Diego Fontdevila, Sherlock A. Licorish, Steffen Kupper, Gunther Ruhe, Eric Knauss, Ozden Ozcan-Top, Paul Clarke, Fergal McCaffery, Marcela Genero, Aurora Vizcaino, Mario Piattini, Marcos Kalinowski, Tayana Conte, Rafael Prikladnicki, Stephan Krusche, Ahmet Coskuncay, Ezequiel Scott, Fabio Calefato, Svetlana Pimonova, Rolf-Helge Pfeiffer, Ulrik Pagh Schultz, Rogardt Heldal, Masud Fazal-Baqaie, Craig Anslow, Maleknaz Nayebi, Kurt Schneider, Stefan Sauer, Dietmar Winkler, Stefan Biffl, Maria Cecilia Bastarrica, and Ita Richardson. What makes agile software development agile? *IEEE Transactions on Software Engineering*, 48(9):3523–3539, Sep 2022, DOI 10.1109/tse.2021.3099532.

[**Kula2022**] Raula Gaikovina Kula and Christoph Treude. In war and peace: The impact of world politics on software ecosystems, 2022.

[**Kumar2022**] Pranjay Kumar, Davin Ie, and Melina Vidoni. On the developers' attitude towards CRAN checks. In *Proc. International Conference on Program Comprehension (ICPC)*. ACM, May 2022, DOI

10.1145/3524610.3528389.
Abstract: R is a package-based, multi-paradigm programming language for scientific software. It provides an easy way to install third-party code, datasets, tests, documentation and examples through CRAN (Comprehensive R Archive Network). Prior works indicated developers tend to code workarounds to bypass CRAN's automated checks (performed when submitting a package) instead of fixing the code-doing so reduces packages' quality. It may become a threat to those analyses written in R that rely on miss-checked code. This preliminary study card-sorted source code comments and analysed StackOverflow (SO) conversations discussing CRAN checks to understand developers' attitudes. We determined that about a quarter of SO posts aim to bypass a check with a workaround; the most affected are code-related problems, package dependencies, installation and feasibility. We analyse these checks and outline future steps to improve similar automated analyses.

[**Kuttal2021**] Sandeep Kaur Kuttal, Xiaofan Chen, Zhendong Wang, Sogol Balali, and Anita Sarma. Visual resume: Exploring developers' online contributions for hiring. *Information and Software Technology*, 138:106633, Oct 2021, DOI 10.1016/j.infsof.2021.106633.
Abstract: Context: Recruiters and practitioners are increasingly relying on online activities of developers to find a suitable candidate. Past empirical studies have identified technical and soft skills that managers use in online peer production sites when making hiring decisions. However, finding candidates with relevant skills is a labor-intensive task for managers, due to the sheer amount of information online peer production sites contain. Objective: We designed a profile aggregation tool—Visual Resume—that aggregates contribution information across two types of peer production sites: a code hosting site (GitHub) and a technical Q&A forum (Stack Overflow). Visual Resume displays summaries of developers' contributions and allows easy access to their contribution details. It also facilitates pairwise comparisons of candidates through a card-based design. We present the motivation for such a design and design guidelines for creating such recruitment tool. Methods: We performed a scenario-based evaluation to identify how participants use developers' online contributions in peer production sites as well as how they used Visual Resume when making hiring decisions. Results: Our analysis helped in identifying the technical and soft skill cues that were most useful to our participants when making hiring decisions in online production sites. We also identified the information features that participants used and the ways the participants accessed that information to select a candidate. Conclusions: Our results suggest that Visual Resume helps in participants evaluate cues for technical and soft skills more efficiently as it presents an aggregated view of candidate's contributions, allows drill down to details about contributions, and allows easy comparison of candidates via movable cards that could be arranged to match participants' needs.

[**Lamba2020**] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian

Kästner, Heather Miller, and Bogdan Vasilescu. Heard it through the gitvine: an empirical study of tool diffusion across the npm ecosystem. In *Proc. European Software Engineering Conference/International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov 2020, DOI 10.1145/3368089.3409705.
Abstract: Automation tools like continuous integration services, code coverage reporters, style checkers, dependency managers, etc. are all known to provide significant improvements in developer productivity and software quality. Some of these tools are widespread, others are not. How do these automation "best practices" spread? And how might we facilitate the diffusion process for those that have seen slower adoption? In this paper, we rely on a recent innovation in transparency on code hosting platforms like GitHub—the use of repository badges—to track how automation tools spread in open-source ecosystems through different social and technical mechanisms over time. Using a large longitudinal data set, multivariate network science techniques, and survival analysis, we study which socio-technical factors can best explain the observed diffusion process of a number of popular automation tools. Our results show that factors such as social exposure, competition, and observability affect the adoption of tools significantly, and they provide a roadmap for software engineers and researchers seeking to propagate best practices and tools.

[**Langhout2021**] Chris Langhout and Maurício Aniche. Atoms of confusion in java, 2021.

[**LawrenceDill2022**] Carolyn J Lawrence-Dill, Robyn L Allscheid, Albert Boaitey, Todd Bauman, Edward S Buckler, 4th, Jennifer L Clarke, Christopher Cullis, Jack Dekkers, Cassandra J Dorius, Shawn F Dorius, David Ertl, Matthew Homann, Guiping Hu, Mary Losch, Eric Lyons, Brenda Murdoch, Zahra-Katy Navabi, Somashekhar Punnuri, Fahad Rafiq, James M Reecy, Patrick S Schnable, Nicole M Scott, Moira Sheehan, Xavier Sirault, Margaret Staton, Christopher K Tuggle, Alison Van Eenennaam, and Rachael Voas. Ten simple rules to ruin a collaborative environment. *PLoS Comput. Biol.*, 18(4):e1009957, Apr 2022, DOI 10.1371/journal.pcbi.1009957.

[**Leelaprute2022**] Pattara Leelaprute, Bodin Chinthanet, Supatsara Wattanakriengkrai, Raula Gaikovina Kula, Pongchai Jaisri, and Takashi Ishio. Does coding in pythonic zen peak performance? preliminary experiments of nine pythonic idioms at scale, 2022.

[**Leinonen2022**] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A. Becker. Using large language models to enhance programming error messages, 2022.

[**Leven2022**] William Levén, Hampus Broman, Terese Besker, and Richard Torkar. The broken windows theory applies to technical debt, 2022.

[**Li2019**] Chen Li, Emily Chan, Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. Towards a framework for teaching debugging. In *Proceedings of the Twenty-First Australasian Computing Education Conference on - ACE '19*. ACM Press, 2019, DOI 10.1145/3286960.3286970.

Abstract: Debugging is an important component of software development, yet most novice programmers are not explicitly taught to apply systematic strategies or processes for debugging. In this paper we adapt a framework developed for teaching troubleshooting to the debugging domain, and explore how the literature on teaching debugging maps to this framework. We identify debugging processes that are fundamental for novices to learn, aspects of debugging that novices typically struggle to develop, and shortcomings of tools designed to support teaching of debugging.

[**Liang2022**] Jenny T Liang, Thomas Zimmermann, and Denae Ford. Understanding skills for OSS communities on GitHub. In *Proc. European Software Engineering Conference/International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov 2022, DOI 10.1145/3540250.3549082.

Abstract: The development of open source software (OSS) is a broad field which requires diverse skill sets. For example, maintainers help lead the project and promote its longevity, technical writers assist with documentation, bug reporters identify defects in software, and developers program the software. However, it is unknown which skills are used in OSS development as well as OSS contributors' general attitudes towards skills in OSS. In this paper, we address this gap by administering a survey to a diverse set of 455 OSS contributors. Guided by these responses as well as prior literature on software development expertise and social factors of OSS, we develop a model of skills in OSS that considers the many contexts OSS contributors work in. This model has 45 skills in the following 9 categories: technical skills, working styles, problem solving, contribution types, project-specific skills, interpersonal skills, external relations, management, and characteristics. Through a mix of qualitative and quantitative analyses, we find that OSS contributors are actively motivated to improve skills and perceive many benefits in sharing their skills with others. We then use this analysis to derive a set of design implications and best practices for those who incorporate skills into OSS tools and platforms, such as GitHub.

[**Liu2022**] Eric S. Liu, Dylan A. Lukes, and William G. Griswold. Refactoring in computational notebooks. *ACM Transactions on Software Engineering and Methodology*, Dec 2022, DOI 10.1145/3576036.

[**Lu2021**] Kuang-Chen Lu, Ben Greenman, and Shriram Krishnamurthi. Types for tables: A language design benchmark. *Art Sci. Eng. Program.*, 6(2), Nov 2021, DOI 10.22152/programming-journal.org/2022/6/8.

Abstract: Context Tables are ubiquitous formats for data. Therefore, techniques for writing correct programs over tables, and debugging incorrect ones, are vital. Our specific focus in this paper is on rich types that ar-

ticulate the properties of tabular operations. We wish to study both their expressive power and diagnostic quality . Inquiry There is no "standard library" of table operations. As a result, every paper (and project) is free to use its own (sub)set of operations. This makes artifacts very difficult to compare, and it can be hard to tell whether omitted operations were left out by oversight or because they cannot actually be expressed. Furthermore, virtually no papers discuss the quality of type error feedback. Approach We combed through several existing languages and libraries to create a "standard library" of table operations. Each entry is accompanied by a detailed specification of its "type," expressed independent of (and hence not constrained by) any type language. We also studied and categorized a corpus of (student) program edits that resulted in table-related errors. We used this to generate a suite of erroneous programs. Finally, we adapted the concept of a datasheet to facilitate comparisons of different implementations. Knowledge Our benchmark creates a common ground to frame work in this area. Language designers who claim to support typed programming over tables have a clear suite against which to demonstrate their system's expressive power. Our family of errors also gives them a chance to demonstrate the quality of feedback. Researchers who improve one aspect—especially error reporting—without changing the other can demonstrate their improvement, as can those who engage in trade-offs between the two. The net result should be much better science in both expressiveness and diagnostics. We also introduce a datasheet format for presenting this knowledge in a methodical way. ubiquitous, and the expressive power of type systems keeps growing. Our benchmark and datasheet can help lead to more orderly science. It also benefits programmers trying to choose a language.

[**Luders2022**] Clara Marie Lüders, Abir Bouraffa, and Walid Maalej. Beyond duplicates. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3528457.

Abstract: Software projects use Issue Tracking Systems (ITS) like JIRA to track issues and organize the workflows around them. Issues are often interconnected via different links such as the default JIRA link types Duplicate, Relate, Block, or Subtask. While previous research has mostly focused on analyzing and predicting duplication links, this work aims at understanding the various other link types, their prevalence, and characteristics towards a more reliable link type prediction. For this, we studied 607,208 links connecting 698,790 issues in 15 public JIRA repositories. Besides the default types, the custom types Depend, Incorporate, Split, and Cause were also common. We manually grouped all 75 link types used in the repositories into five general categories: General Relation, Duplication, Composition, Temporal/Causal, and Workflow. Comparing the structures of the corresponding graphs, we observed several trends. For instance, Duplication links tend to represent simpler issue graphs often with two components and Composition links present the highest amount of hierarchical tree structures (97.7%). Surprisingly, General Relation links have a significantly higher transitivity score

than Duplication and Temporal/ Causal links. Motivated by the differences between the link types and by their popularity, we evaluated the robustness of two state-of-the-art duplicate detection approaches from the literature on the JIRA dataset. We found that current deep-learning approaches confuse between Duplication and other links in almost all repositories. On average, the classification accuracy dropped by 6% for one approach and 12% for the other. Extending the training sets with other link types seems to partly solve this issue. We discuss our findings and their implications for research and practice.

[**MacNeil2022**] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. Experiences from using code explanations generated by large language models in a web software development e-book, 2022.

[**Martin2023**] Florence Martin, Swapna Kumar, Albert D Ritzhaupt, and Drew Polly. Bichronous online learning: Award-winning online instructor practices of blending asynchronous and synchronous online modalities. *Internet High. Educ.*, 56(100879):100879, Jan 2023, DOI `10.1016/j.iheduc.2022.100879`.

[**Mashey2021**] John R. Mashey. Interactions, impacts, and coincidences of the first golden age of computer architecture. *IEEE Micro*, 41(6):131–139, Nov 2021, DOI `10.1109/mm.2021.3112876`.

[**May2019**] Anna May, Johannes Wachs, and Anikó Hannák. Gender differences in participation and reward on stack overflow. *Empirical Software Engineering*, 24(4):1997–2019, Feb 2019, DOI `10.1007/s10664-019-09685-x`. **Abstract:** Programming is a valuable skill in the labor market, making the underrepresentation of women in computing an increasingly important issue. Online question and answer platforms serve a dual purpose in this field: they form a body of knowledge useful as a reference and learning tool, and they provide opportunities for individuals to demonstrate credible, verifiable expertise. Issues, such as male-oriented site design or overrepresentation of men among the site's elite may therefore compound the issue of women's underrepresentation in IT. In this paper we audit the differences in behavior and outcomes between men and women on Stack Overflow, the most popular of these Q&A sites. We observe significant differences in how men and women participate in the platform and how successful they are. For example, the average woman has roughly half of the reputation points, the primary measure of success on the site, of the average man. Using an Oaxaca-Blinder decomposition, an econometric technique commonly applied to analyze differences in wages between groups, we find that most of the gap in success between men and women can be explained by differences in their activity on the site and differences in how these activities are rewarded. Specifically, 1) men give more answers than women and 2) are rewarded more for their answers on average, even when controlling for possible confounders such as

tenure or buy-in to the site. Women ask more questions and gain more reward per question. We conclude with a hypothetical redesign of the site's scoring system based on these behavioral differences, cutting the reputation gap in half.

[**Mehrpour2022**] Sahar Mehrpour and Thomas D. LaToza. Can static analysis tools find more defects? *Empirical Software Engineering*, 28(1), Nov 2022, DOI 10.1007/s10664-022-10232-4.

[**Miedema2022**] Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. So many brackets! In *Proc. International Conference on Program Comprehension (ICPC)*. ACM, May 2022, DOI 10.1145/3524610.3529158.
Abstract: The Structured Query Language (SQL) is a widely taught database query language in computer science, data science, and software engineering programs. While highly expressive, SQL is challenging to learn for novices. Various research has explored the errors and mistakes that SQL users make. Specific attributes of SQL code, such as the number of tables and the degree of nesting, have been found to impact its understandability and maintainability. Furthermore, prior studies have shown that novices have significant issues using SQL correctly, due to factors such as expressive ease, existing knowledge and misconceptions, and the impact of cognitive load. In this paper we identify another factor: self-inflicted query complexity, where users hinder their own problem solving process. We analyse 8K intermediate and final student attempts to six SQL exer-cises, approaching complexity from four perspective: correctness, execution order, edit distance and query intricacy. Through our analyses, we find that our students are hindered in their query formulation process by mismanaging complexity through writing overly elaborate queries containing unnecessary elements, overusing brackets and nesting, and incrementally building queries with persistent errors.

[**NandSharma2022**] Pankajeshwara Nand Sharma, Bastin Tony Roy Savarimuthu, and Nigel Stanger. Unearthing open source decision-making processes: A case study of python enhancement proposals. *Softw. Pract. Exp.*, 52(10):2312–2346, Oct 2022, DOI 10.1002/spe.3128.
Abstract: Good governance practices are pivotal to the success of Open Source Software (OSS) projects. However, the decision-making processes that are made available to stakeholders are at times incomplete and may remain buried and hidden in large amounts of software repository data. This work bridges this gap by unearthing enacted decision-making processes available for Python Enhancement Proposals (PEPs) from 1.54 million email messages that embody decisions made during the evolution of the Python language. This work employs a design science approach in operationalizing a framework called DeMaP miner that is used to discover hidden processes using information retrieval and information extraction techniques. It also uses process mining techniques to visualize the processes, and comparative structural analysis techniques to compare different decision processes. The work identifies a richer set of decision-making activities than those

reported on the Python website and in prior research work (48 new decision activities, 199 new pathways and 6 new stages). The extracted decision process has been positively evaluated by a prominent member of the Python steering council. The extracted process can be used for process compliance checking and process improvement in OSS communities. Additionally, the DeMaP Miner framework can be extended and customized to suit other OSS projects, such as the OpenJDK project.

[**Nguyen2022**] Nhan Nguyen and Sarah Nadi. An empirical evaluation of GitHub copilot's code suggestions. In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3528470.
Abstract: GitHub and OpenAI recently launched Copilot, an "AI pair programmer" that utilizes the power of Natural Language Processing, Static Analysis, Code Synthesis, and Artificial Intelligence. Given a natural language description of the target functionality, Copilot can generate corresponding code in several programming languages. In this paper, we perform an empirical study to evaluate the correctness and understandability of Copilot's suggested code. We use 33 LeetCode questions to create queries for Copilot in four different programming languages. We evaluate the correctness of the corresponding 132 Copilot solutions by running LeetCode's provided tests, and evaluate understandability using SonarQube's cyclomatic complexity and cognitive complexity metrics. We find that Copilot's Java suggestions have the highest correctness score (57%) while JavaScript is the lowest (27%). Overall, Copilot's suggestions have low complexity with no notable differences between the programming languages. We also find some potential Copilot shortcomings, such as generating code that can be further simplified and code that relies on undefined helper methods.

[**Nicacio2022**] Jalves Nicacio and Fabio Petrillo. An approach to build consistent software architecture diagrams using devops system descriptors, 2022, DOI 10.1145/3550356.3561567.

[**Noble2022**] James Noble, David Streader, Isaac Oscar Gariano, and Miniruwani Samarakoon. More programming than programming: Teaching formal methods in a software engineering programme, 2022.

[**Olejniczak2020**] Anthony J. Olejniczak and Molly J. Wilson. Who's writing open access (OA) articles? characteristics of OA authors at ph.d.-granting institutions in the united states. *Quantitative Science Studies*, 1(4):1429–1450, Dec 2020, DOI 10.1162/qss_a_00091.
Abstract: The open access (OA) publication movement aims to present research literature to the public at no cost and with no restrictions. While the democratization of access to scholarly literature is a primary focus of the movement, it remains unclear whether OA has uniformly democratized the corpus of freely available research, or whether authors who choose to publish in OA venues represent a particular subset of scholars—those with access to

resources enabling them to afford article processing charges (APCs). We investigated the number of OA articles with article processing charges (APC OA) authored by 182,320 scholars with known demographic and institutional characteristics at American research universities across 11 broad fields of study. The results show, in general, that the likelihood for a scholar to author an APC OA article increases with male gender, employment at a prestigious institution (AAU member universities), association with a STEM discipline, greater federal research funding, and more advanced career stage (i.e., higher professorial rank). Participation in APC OA publishing appears to be skewed toward scholars with greater access to resources and job security.

[**Palomba2021**] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Transactions on Software Engineering*, 47(1):108–129, Jan 2021, DOI 10.1109/tse.2018.2883603.

Abstract: Code smells are poor implementation choices applied by developers during software evolution that often lead to critical flaws or failure. Much in the same way, community smells reflect the presence of organizational and socio-technical issues within a software community that may lead to additional project costs. Recent empirical studies provide evidence that community smells are often—if not always—connected to circumstances such as code smells. In this paper we look deeper into this connection by conducting a mixed-methods empirical study of 117 releases from 9 open-source systems. The qualitative and quantitative sides of our mixed-methods study were run in parallel and assume a mutually-confirmative connotation. On the one hand, we survey 162 developers of the 9 considered systems to investigate whether developers perceive relationship between community smells and the code smells found in those projects. On the other hand, we perform a fine-grained analysis into the 117 releases of our dataset to measure the extent to which community smells impact code smell intensity (i.e., criticality). We then propose a code smell intensity prediction model that relies on both technical and community-related aspects. The results of both sides of our mixed-methods study lead to one conclusion: community-related factors contribute to the intensity of code smells. This conclusion supports the joint use of community and code smells detection as a mechanism for the joint management of technical and social problems around software development communities.

[**Pelnek2022**] Radek Pelánek and Tomáš Effenberger. The landscape of computational thinking problems for practice and assessment. *ACM Transactions on Computing Education*, Dec 2022, DOI 10.1145/3578269.

[**Pereira2017**] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software*

Language Engineering. ACM, Oct 2017, DOI 10.1145/3136014.3136031.
Abstract: This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. Finally, we show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern.

[**Pinckney2022**] Donald Pinckney, Federico Cassano, Arjun Guha, Jon Bell, Massimiliano Culpo, and Todd Gamblin. Flexible and optimal dependency management via max-smt, 2022.

[**Pinto2022**] Gustavo Pinto and Alberto de Souza. Cognitive-driven development helps software teams to keep code units under the limit!, 2022.

[**Poulos2021**] Alexandra Poulos, Sally A. McKee, and Jon C. Calhoun. Posits and the state of numerical representations in the age of exascale and edge computing. *Software: Practice and Experience*, 52(2):619–635, Sep 2021, DOI 10.1002/spe.3022.
Abstract: Growing constraints on memory utilization, power consumption, and I/O throughput have increasingly become limiting factors to the advancement of high performance computing (HPC) and edge computing applications. IEEE-754 floating-point types have been the de facto standard for floating-point number systems for decades, but the drawbacks of this numerical representa- tion leave much to be desired. Alternative representations are gaining traction, both in HPC and machine learning environments. Posits have recently been proposed as a drop-in replacement for the IEEE-754 floating-point representa- tion. We survey the state-of-the-art and state-of-the-practice in the development and use of posits in edge computing and HPC. The current literature supports posits as a promising alternative to traditional floating-point systems, both as a stand-alone replacement and in a mixed-precision environment. Development and standardization of the posit type is ongoing, and much research remains to explore the application of posits in different domains, how to best implement them in hardware, and where they fit with other numerical representations.

[**Prana2019**] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the content of GitHub README files. *Empir. Softw. Eng.*, 24(3):1296–1327, Jun 2019, DOI 10.1007/s10664-018-9660-3.

[**PreslerMarshall2022a**] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. Identifying struggling teams in software engineering courses through weekly surveys. In *Proc. Technical Symposium on Computer Science Education (SIGCSE)*. ACM, Feb 2022, DOI 10.1145/3478431.3499367.

Abstract: Teaming is increasingly a core aspect of professional software engineering and most undergraduate computer science curricula. At NC State University, we teach communication and project-management skills explicitly through a junior-level software engineering course. However, some students may have a dysfunctional team experience that imperils their ability to learn these skills. Identifying these teams during a team project is important so the teaching staff can intervene early and hopefully alleviate the issues. We propose a weekly reflection survey to help the course teaching staff proactively identify teams that may not be on track to learn the course outcomes. The questions on the survey focus on team communication and collaboration over the previous week. We evaluate our survey on two semesters of the undergraduate software engineering course by comparing teams with poor end-of-project grades or peer evaluations against teams flagged on a weekly basis through the surveys. We find that the survey can identify most teams that later struggled on the project, typically by the half-way mark of the project, and thus may provide instructors with an actionable early-warning about struggling teams. Furthermore, a majority of students (64.4%) found the survey to be a helpful tool for keeping their team on track. Finally, we discuss future work for improving the survey and engaging with student teams.

[**PreslerMarshall2022b**] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. What makes team[s] work? a study of team characteristics in software engineering projects. In *Proc. Conference on International Computing Education Research (ICER)*. ACM, Aug 2022, DOI 10.1145/3501385.3543980.

Abstract: Teaming is a core component in practically all professional software engineering careers, and as such, is a key skill taught in many undergraduate Computer Science programs. However, not all teams manage to work together effectively, and in education, this can deprive some students of successful teaming experiences. In this work, we seek to gain insights into the characteristics of successful and unsuccessful undergraduate student teams in a software engineering course. We conduct semi-structured interviews with 18 students who have recently completed a team-based software engineering course to understand how they worked together, what challenges they faced, and how they tried to overcome these challenges. Our results show that common problems include communicating, setting and holding to deadlines, and effectively identifying tasks and their relative difficulty. Additionally, we find that self-reflection on what is working and not working or external motivators such as grades help some, but not all, teams overcome these challenges. Finally, we conclude with recommendations for educators on successful behaviours to steer teams towards, and recommendations for researchers on future work to better understand challenges that teams face.

[**Qamar2022**] Khushbakht Ali Qamar, Emre Sülün, and Eray Tüzün. Taxonomy of bug tracking process smells: Perceptions of practitioners and an

empirical analysis. *Inf. Softw. Technol.*, 150(106972):106972, Oct 2022, DOI 10.1016/j.infsof.2022.106972.

[**Queiroz2022**] Francisco Queiroz, Maria Lonsdale, and Rejane Spitz. Science as a game: conceptual model and application in scientific software design. *Int. j. des. creat. innov.*, 10(4):222–246, Oct 2022, DOI 10.1080/21650349.2022.2088623.
Abstract: ABSTRACT Scientific inquiry is often described as, and compared to, a game. This paper expands on that analogy to propose a conceptual model of scientific practice built upon Jesper Juul's game definition, and informed by parallels between the two activities collected from selected works from history and philosophy of science. Moreover, the paper presents a design method, based on the model described, for fostering creative solutions in scientific software user interface design. Results from pilot case studies suggest both model and method are helpful, allowing participants to describe requirements and ideate solutions, as well providing a framework for the exploration of the game-science analogy within the context of scientific research conducted through computational resources.

[**Ragkhitwetsagul2022**] Chaiyong Ragkhitwetsagul and Matheus Paixao. Recommending code improvements based on stack overflow answer edits, 2022.

[**Rahman2020b**] Mohammad Masudur Rahman, Foutse Khomh, and Marco Castelluccio. Why are some bugs non-reproducible? an empirical investigation using data fusion. In *Proc. International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Sep 2020, DOI 10.1109/icsme46990.2020.00063.
Abstract: Software developers attempt to reproduce software bugs to understand their erroneous behaviours and to fix them. Unfortunately, they often fail to reproduce (or fix) them, which leads to faulty, unreliable software systems. However, to date, only a little research has been done to better understand what makes the software bugs non-reproducible. In this paper, we conduct a multimodal study to better understand the non-reproducibility of software bugs. First, we perform an empirical study using 576 non-reproducible bug reports from two popular software systems (Firefox, Eclipse) and identify 11 key factors that might lead a reported bug to non-reproducibility. Second, we conduct a user study involving 13 professional developers where we investigate how the developers cope with non-reproducible bugs. We found that they either close these bugs or solicit for further information, which involves long deliberations and counter-productive manual searches. Third, we offer several actionable insights on how to avoid non-reproducibility (e.g., false-positive bug report detector) and improve reproducibility of the reported bugs (e.g., sandbox for bug reproduction) by combining our analyses from multiple studies (e.g., empirical study, developer study).

[**Rahman2021**] Akond Rahman, Md Rayhanur Rahman, Chris Parnin, and Laurie Williams. Security smells in ansible and chef scripts. *ACM Transactions on Software Engineering and Methodology*, 30(1):1–31, Jan 2021, DOI 10.1145/3408897.

Abstract: Context: Security smells are recurring coding patterns that are indicative of security weakness and require further inspection. As infrastructure as code (IaC) scripts, such as Ansible and Chef scripts, are used to provision cloud-based servers and systems at scale, security smells in IaC scripts could be used to enable malicious users to exploit vulnerabilities in the provisioned systems. Goal: The goal of this article is to help practitioners avoid insecure coding practices while developing infrastructure as code scripts through an empirical study of security smells in Ansible and Chef scripts. Methodology: We conduct a replication study where we apply qualitative analysis with 1,956 IaC scripts to identify security smells for IaC scripts written in two languages: Ansible and Chef. We construct a static analysis tool called Security Linter for Ansible and Chef scripts (SLAC) to automatically identify security smells in 50,323 scripts collected from 813 open source software repositories. We also submit bug reports for 1,000 randomly selected smell occurrences. Results: We identify two security smells not reported in prior work: missing default in case statement and no integrity check. By applying SLAC we identify 46,600 occurrences of security smells that include 7,849 hard-coded passwords. We observe agreement for 65 of the responded 94 bug reports, which suggests the relevance of security smells for Ansible and Chef scripts amongst practitioners. Conclusion: We observe security smells to be prevalent in Ansible and Chef scripts, similarly to that of the Puppet scripts. We recommend practitioners to rigorously inspect the presence of the identified security smells in Ansible and Chef scripts using (i) code review, and (ii) static analysis tools.

[**Rao2022**] Nikitha Rao, Jason Tsay, Martin Hirzel, and Vincent J. Hellendoorn. Comments on comments: Where code review and documentation meet, 2022, DOI 10.1145/3524842.3528475.

[**Ritschel2022a**] Nico Ritschel, Vladimir Kovalenko, Reid Holmes, Ronald Garcia, and David C. Shepherd. Comparing block-based programming models for two-armed robots. *IEEE Transactions on Software Engineering*, 48(5):1630–1643, May 2022, DOI 10.1109/tse.2020.3027255.

[**Ritschel2022b**] Nico Ritschel, Anand Ashok Sawant, David Weintrop, Reid Holmes, Alberto Bacchelli, Ronald Garcia, Chandrika K R, Avijit Mandal, Patrick Francis, and David C. Shepherd. Training industrial end-user programmers with interactive tutorials. *Software: Practice and Experience*, Nov 2022, DOI 10.1002/spe.3167.

[**Rule2019**] Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H Nguyen, Sara Brin Rosenthal, Fernando Pérez, and Peter W Rose. Ten simple rules for writing

and sharing computational analyses in jupyter notebooks. *PLoS Comput. Biol.*, 15(7):e1007007, Jul 2019, DOI 10.1371/journal.pcbi.1007007.
Abstract: 1 Design Lab, UC San Diego, La Jolla, California, United States of America, 2 Center for Computational Biology and Bioinformatics, UC San Diego, La Jolla, California, United States of America, 3 Department of Pediatrics, UC San Diego, La Jolla, California, United States of America, 4 Data Science Hub, San Diego Supercomputer Center, UC San Diego, La Jolla, California, United States of America, 5 Departments of Bioengineering, and Computer Science and Engineering, and Center for Microbiome Innovation, UC San Diego, La Jolla, California, United States of America, 6 Bioinformatics and Systems Biology Graduate Program, UC San Diego, La Jolla, California, United States of America, 7 Department of Statistics and Berkeley Institute for Data Science, UC Berkeley, and Lawrence Berkeley National Laboratory, Berkeley, California, United States of America

[**Sanders2019**] Kate Sanders, Judy Sheard, Brett A Becker, Anna Eckerdal, Sally Hamouda, and Simon. Inferential statistics in computing education research. In *Proc. Conference on International Computing Education Research (ICER)*. ACM, Jul 2019, DOI 10.1145/3291279.3339408.
Abstract: The goal of most computing education research is to effect positive change in how computing is taught and learned. Statistical techniques are one important tool for achieving this goal. In this paper we report on an analysis of ICER papers that use inferential statistics. We present the most commonly used techniques; an overview of the techniques the ICER community has used over its first 14 years of papers, grouped according to the purpose of the technique; and a detailed analysis of three of the most commonly used techniques (t-test, chi-squared test, and Mann-Whitney-Wilcoxon). We identify common flaws in reporting and give examples of papers where statistics are reported well. In sum, the paper draws a picture of the use of inferential statistics by the ICER community. This picture is intended to help orient researchers who are new to the use of statistics in computing education research and to encourage reflection by the ICER community on how it uses statistics and how it can improve that use.

[**Schmitt2022**] Paul Schmitt, Jana Iyengar, Christopher Wood, and Barath Raghavan. The decoupling principle. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. ACM, Nov 2022, DOI 10.1145/3563766.3564112.

[**Schurhoff2022**] Christian Schürhoff, Stefan Hanenberg, and Volker Gruhn. An empirical study on a single company's cost estimations of 338 software projects. *Empirical Software Engineering*, 28(1), Nov 2022, DOI 10.1007/s10664-022-10245-z.

[**Shankar2022**] Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. Operationalizing machine learning: An interview study, 2022.

[**Sharafi2022**] Zohreh Sharafi, Ian Bertram, Michael Flanagan, and Westley Weimer. Eyes on code: A study on developers' code navigation strategies. *IEEE Transactions on Software Engineering*, 48(5):1692–1704, May 2022, DOI 10.1109/tse.2020.3032064.

[**Shome2022**] Arumoy Shome, Luís Cruz, and Arie van Deursen. Data smells in public datasets. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. ACM, May 2022, DOI 10.1145/3522664.3528621.

Abstract: The adoption of Artificial Intelligence (AI) in high-stakes domains such as healthcare, wildlife preservation, autonomous driving and criminal justice system calls for a data-centric approach to AI. Data scientists spend the majority of their time studying and wrangling the data, yet tools to aid them with data analysis are lacking. This study identifies the recurrent data quality issues in public datasets. Analogous to code smells, we introduce a novel catalogue of data smells that can be used to indicate early signs of problems or technical debt in machine learning systems. To understand the prevalence of data quality issues in datasets, we analyse 25 public datasets and identify 14 data smells.

[**Shrestha2021**] Nischal Shrestha, Titus Barik, and Chris Parnin. Unravel: A fluent code explorer for data wrangling. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. ACM, Oct 2021, DOI 10.1145/3472749.3474744.

Abstract: Data scientists have adopted a popular design pattern in programming called the fluent interface for composing data wrangling code. The fluent interface works by combining multiple transformations on a data table—or dataframes—with a single chain of expressions, which produces an output. Although fluent code promotes legibility, the intermediate dataframes are lost, forcing data scientists to unravel the chain through tedious code edits and re-execution. Existing tools for data scientists do not allow easy exploration or support understanding of fluent code. To address this gap, we designed a tool called Unravel that enables structural edits via drag-and-drop and toggle switch interactions to help data scientists explore and understand fluent code. Data scientists can apply simple structural edits via drag-and-drop and toggle switch interactions to reorder and (un)comment lines. To help data scientists understand fluent code, Unravel provides function summaries and always-on visualizations highlighting important changes to a dataframe. We discuss the design motivations behind Unravel and how it helps understand and explore fluent code. In a first-use study with 14 data scientists, we found that Unravel facilitated diverse activities such as validating assumptions about the code or data, exploring alternatives, and revealing function behavior.

[**Silva2022**] Yasin N Silva, Alexis Loza, and Humberto Razente. DBSnap-eval. In *Proc. Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM, Jul 2022, DOI 10.1145/3502718.3524822.

Abstract: Learning to construct database queries can be a challenging task because students need to learn the specific query language syntax as well as properly understand the effect of each query operator and how multiple operators interact in a query. While some previous studies have looked into the types of database query errors students make and how the availability of expected query results can help to increase the success rate, there is very little that is known regarding the patterns that emerge while students are constructing a query. To be able to look into the process of constructing a query, in this paper we introduce DBSnap-Eval, a tool that supports tree-based queries (similar to SQL query plans) and a block-based querying interface to help separate the syntax and semantics of a query. DBSnap-Eval closely monitors the actions students take to construct a query such as adding a dataset or connecting a dataset with an operator. This paper presents an initial set of results about database query construction patterns using DBSnap-Eval. Particularly, it reports identified patterns in the process students follow to answer common database queries.

[**Soltani2020**] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. The significance of bug report elements. *Empir. Softw. Eng.*, 25(6):5255–5294, Nov 2020, DOI 10.1007/s10664-020-09882-z.
Abstract: AbstractOpen source software projects often use issue repositories, where project contributors submit bug reports. Using these repositories, more bugs in software projects may be identified and fixed. However, the content and therefore quality of bug reports vary. In this study, we aim to understand the significance of different elements in bug reports. We interviewed 35 developers to gain insights into their perceptions on the importance of various contents in bug reports. To assess our findings, we surveyed 305 developers. The results show developers find it highly important that bug reports include crash description, reproducing steps or test cases, and stack traces. Software version, fix suggestions, code snippets, and attached contents have lower importance for software debugging. Furthermore, to evaluate the quality of currently available bug reports, we mined issue repositories of 250 most popular projects on Github. Statistical analysis on the mined issues shows that crash reproducing steps, stack traces, fix suggestions, and user contents, have statistically significant impact on bug resolution times, for $\tilde{7}0\%$, $\tilde{7}6\%$, $\tilde{5}5\%$, and $\tilde{3}3\%$ of the projects. However, on avarage, over 70% of bug reports lack these elements.

[**Spinellis2018**] Diomidis Spinellis. Modern debugging. *Commun. ACM*, 61(11):124–134, Oct 2018, DOI 10.1145/3186278.

[**Stokes2022**] Chase Stokes and Marti Hearst. Why more text is (often) better: Themes from reader preferences for integration of charts and text, 2022.

[**Storey2022**] Margaret-Anne Storey, Brian Houck, and Thomas Zimmermann. How developers and managers define and trade productivity for quality. In *Proc. Conference on Human Aspects of Software Engineering*

(CHASE). ACM, May 2022, DOI 10.1145/3528579.3529177.
Abstract: Background: Developer productivity and software quality are different but related multi-dimensional lenses into the software engineering process. The terms are used liberally in industry settings, but there is a lack of consensus and awareness of what these terms mean in specific contexts and which trade-offs should be considered. Objective & Method: Through an exploratory survey study with developers and managers at Microsoft, we investigated how these cohorts define productivity and quality, how aligned they are in their views, how aware they are of other views, and if and how they trade quality for productivity. Results: We find developers and managers, as cohorts, are not well-aligned in their views of productivity–developers think more about work activities, while more managers consider performance or quality outcomes. We find developers and managers have more aligned views of what quality means, with the majority defining quality in terms of robustness, while the timely delivery of evolvable features that delight users are also key quality aspects. Over half of the developers and managers we surveyed make productivity and quality trade-offs but with good reasons for doing so. Conclusion: Alignment on how developers and managers define productivity and quality is essential if they are to design effective improvement interventions and meaningful metrics to measure productivity and quality improvements. Our research provides a frame for developers and managers to align their views and to make informed decisions on productivity and quality trade-offs.

[**Strode2022**] Diane Strode, Torgeir Dingsøyr, and Yngve Lindsjorn. A teamwork effectiveness model for agile software development. *Empir. Softw. Eng.*, 27(2), Mar 2022, DOI 10.1007/s10664-021-10115-0.
Abstract: AbstractTeamwork is crucial in software development, particularly in agile development teams which are cross-functional and where team members work intensively together to develop a cohesive software solution. Effective teamwork is not easy; prior studies indicate challenges with communication, learning, prioritization, and leadership. Nevertheless, there is much advice available for teams, from agile methods, practitioner literature, and general studies on teamwork to a growing body of empirical studies on teamwork in the specific context of agile software development. This article presents the agile teamwork effectiveness model (ATEM) for colocated agile development teams. The model is based on evidence from focus groups, case studies, and multi-vocal literature and is a revision of a general team effectiveness model. Our model of agile teamwork effectiveness is composed of shared leadership, team orientation, redundancy, adaptability, and peer feedback. Coordinating mechanisms are needed to facilitate these components. The coordinating mechanisms are shared mental models, communication, and mutual trust. We critically examine the model and discuss extensions for very small, multi-team, distributed, and safety-critical development contexts. The model is intended for researchers, team members, coaches, and leaders in the agile community.

[**Tan2022**] Wen Siang Tan, Markus Wagner, and Christoph Treude. Detecting outdated code element references in software repository documentation, 2022.

[**Tian2022**] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. What makes a good commit message? In *Proc. International Conference on Software Engineering (ICSE)*. ACM, May 2022, DOI 10.1145/3510003.3510205.

Abstract: A key issue in collaborative software development is communication among developers. One modality of communication is a commit message, in which developers describe the changes they make in a repository. As such, commit messages serve as an "audit trail" by which developers can understand how the source code of a project has changed-and why. Hence, the quality of commit messages affects the effectiveness of communication among developers. Commit messages are often of poor quality as developers lack time and motivation to craft a good message. Several automatic approaches have been proposed to generate commit messages. However, these are based on uncurated datasets including considerable proportions of poorly phrased commit messages. In this multi-method study, we first define what constitutes a "good" commit message, and then establish what proportion of commit messages lack information using a sample of almost 1,600 messages from five highly active open source projects. We find that an average of circa 44% of messages could be improved, suggesting the use of uncurated datasets may be a major threat when commit message generators are trained with such data. We also observe that prior work has not considered semantics of commit messages, and there is surprisingly little guidance available for writing good commit messages. To that end, we develop a taxonomy based on recurring patterns in commit messages' expressions. Finally, we investigate whether "good" commit messages can be automatically identified; such automation could prompt developers to write better commit messages.

[**Timperley2020**] Christopher S. Timperley, Lauren Herckis, Claire Le Goues, and Michael Hilton. Understanding and improving artifact sharing in software engineering research, 2020, DOI 10.1007/s10664-021-09973-5.

[**Tissenbaum2021**] Mike Tissenbaum, David Weintrop, Nathan Holbert, and Tamara Clegg. The case for alternative endpoints in computing education. *Br. J. Educ. Technol.*, 52(3):1164–1177, May 2021, DOI 10.1111/bjet.13072.

Abstract: This paper argues for a reexamination of the nature and goals of broad computing education initiatives. Instead of starting with specific values or goals, we instead begin by considering various desired endpoints of computing instruction and then work backward to reason about what form learning activities might take and what are the underlying values and principles that support learners in reaching these endpoints. The result of this exercise is a push for rethinking the form of contemporary computing education with an eye toward more diverse, equitable and meaningful endpoints.

With a focus on the role that constructionistfocused pedagogies and designs can play in supporting these endpoints, we examine four distinct cases and the endpoints they support. This paper is not intended to encompass all the possible alternate endpoints for computer science education; rather, this work seeks to start a conversation around the nature of and need for alternate endpoints, as a means to reevaluate the current tools and curricula to prepare learners for a future of active and empowered computingliterate citizens.

[**Truong2022**] Kimberly Truong, Courtney Miller, Bogdan Vasilescu, and Christian Kästner. The unsolvable problem or the unheard answer? In *Proc. International Conference on Mining Software Repositories (MSR)*. ACM, May 2022, DOI 10.1145/3524842.3528488.
Abstract: Talks at practitioner-focused open-source software conferences are a valuable source of information for software engineering researchers. They provide a pulse of the community and are valuable source material for grey literature analysis. We curated a dataset of 24,669 talks from 87 open-source conferences between 2010 and 2021. We stored all relevant metadata from these conferences and provide scripts to collect the transcripts. We believe this data is useful for answering many kinds of questions, such as: What are the important/highly discussed topics within practitioner communities? How do practitioners interact? And how do they present themselves to the public? We demonstrate the usefulness of this data by reporting our findings from two small studies: a topic model analysis providing an overview of open-source community dynamics since 2011 and a qualitative analysis of a smaller community-oriented sample within our dataset to gain a better understanding of why contributors leave open-source software.

[**Tshukudu2020**] Ethel Tshukudu and Quintin Cutts. Understanding conceptual transfer for students learning new programming languages. In *Proc. Conference on International Computing Education Research (ICER)*. ACM, Aug 2020, DOI 10.1145/3372782.3406270.
Abstract: Prior research has shown that students face transition challenges between programming languages (PL) over the course of their education. We could not find research attempting to devise a model that describes the transition process and how students' learning of programming concepts is affected during the shift. In this paper, we propose a model to describe PL transfer for relative novices. In the model, during initial stages of learning a new language, students will engage in learning three categories of concepts, True Carryover Concepts, False Carryover Concepts, or Abstract True Carryover Concepts; during the transition, learners automatically effect a transfer of semantics between languages based on syntax matching. In order to find support for the model, we conducted two empirical studies. Study 1 investigated near-novice undergraduate students transitioning from procedural Python to object-oriented Java while Study 2 investigated near-novice postgraduate students doing a transfer from object-oriented Java to procedural

Python. Results for both studies indicate that students had little or no difficulty with transitioning on TCC due to positive semantic transfer based on syntax similarities while they had the most difficulty transitioning on FCC due to negative semantic transfer. Students had little or no semantic transfer on ATCC due to differences in syntax between the languages. We suggest ways in which the model can inform pedagogy on how to ease the transition process.

[**Tuna2022**] Erdem Tuna, Vladimir Kovalenko, and Eray Tüzün. Bug tracking process smells in practice. In *Proc. International Conference on Software Engineering (ICSE)*. ACM, May 2022, DOI 10.1145/3510457.3513080.
Abstract: Software teams use bug tracking (BT) tools to report and manage bugs. Each record in a bug tracking system (BTS) is a reporting entity consisting of several information fields. The contents of the reports are similar across different tracking tools, though not the same. The variation in the workflow between teams prevents defining an ideal process of running BTS. Nevertheless, there are best practices reported both in white and gray literature. Developer teams may not adopt the best practices in their BT process. This study investigates the non-compliance of developers with best practices, so-called smells, in the BT process. We mine bug reports of four projects in the BTS of JetBrains, a software company, to observe the prevalence of BT smells in an industrial setting. Also, we survey developers to see (1) if they recognize the smells, (2) their perception of the severity of the smells, and (3) the potential benefits of a BT process smell detection tool. We found that (1) smells occur, and their detection requires a solid understanding of the BT practices of the projects, (2) smell severity perception varies across smell types, and (3) developers considered that a smell detection tool would be useful for six out of the 12 smell categories.

[**Turk2021**] Tomaž Turk. SDFunc: Modular spreadsheet design with sheet-defined functions in microsoft excel. *Software: Practice and Experience*, 52(2):415–426, Sep 2021, DOI 10.1002/spe.3027.
Abstract: The goal of the SDFunc tool is to enable spreadsheet developers to build their model computations in Microsoft Excel according to the modular design approach, that is, the separation of the functionalities into independent, interchangeable modules with interfaces that provide input and output elements. This concept has been theoretically developed in recent years and is known as sheet-defined functions in the literature. In this article, we are presenting our implementation of the tool and the evaluation steps that we took to make the tool interesting and suitable for the assessment of the modular approach in spreadsheet development by the industry, specifically within organizational and companies' settings where the spreadsheet developers and end-users involved in experiments expect to use a well-established spreadsheet platform. We also demonstrated that sheet-defined functions can be implemented by development tools already present in Microsoft Excel.

[**Vidoni2021**] Melina Vidoni. Evaluating unit testing practices in r packages. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, May 2021, DOI 10.1109/icse43902.2021.00136.
Abstract: "Testing Technical Debt (TTD) occurs due to shortcuts (non-optimal decisions) taken about testing; it is the test dimension of technical debt. R is a package-based programming ecosystem that provides an easy way to install third-party code, datasets, tests, documentation and examples. This structure makes it especially vulnerable to TTD because errors present in a package can transitively affect all packages and scripts that depend on it. Thus, TTD can effectively become a threat to the validity of all analysis written in R that rely on potentially faulty code. This two-part study provides the first analysis in this area. First, 177 systematically-selected, open-source R packages were mined and analysed to address quality of testing, testing goals, and identify potential TTD sources. Second, a survey addressed how R package developers perceive testing and face its challenges (response rate of 19.4%). Results show that testing in R packages is of low quality; the most common smells are inadequate and obscure unit testing, improper asserts, inexperienced testers and improper test design. Furthermore, skilled R developers still face challenges such as time constraints, emphasis on development rather than testing, poor tool documentation and a steep learning curve."

[**Vidoni2022**] Melina Vidoni. Understanding roxygen package documentation in r. *Journal of Systems and Software*, 188:111265, Jun 2022, DOI 10.1016/j.jss.2022.111265.
Abstract: R is a package-based programming ecosystem that provides an easy way to install third-party code, datasets, and examples. Thus, R developers rely heavily on the documentation of the packages they import to use them correctly and accurately. This documentation is often written using Roxygen, equivalent to Java's well-known Javadoc. This two-part study provides the first analysis in this area. First, 379 systematically-selected, open-source R packages were mined and analysed to address the quality of their documentation in terms of presence, distribution, and completeness to identify potential sources of documentation debt of technical debt that describes problems in the documentation. Second, a survey addressed how R package developers perceive documentation and face its challenges (with a response rate of 10.04%). Results show that incomplete documentation is the most common smell, with several cases of incorrect use of the Roxygen utilities. Unlike in traditional API documentation, developers do not focus on how behaviour is implemented but on common use cases and parameter documentation. Respondents considered the examples section the most useful, and commonly perceived challenges were unexplained examples, ambiguity, incompleteness and fragmented information.

[**Wang2020**] Zhendong Wang, Yang Feng, Yi Wang, James A Jones, and David Redmiles. Unveiling elite developers' activities in open source projects. *ACM Trans. Softw. Eng. Methodol.*, 29(3):1–35, Jul 2020, DOI 10.1145/3387111.

Abstract: Open source developers, particularly the elite developers who own the administrative privileges for a project, maintain a diverse portfolio of contributing activities. They not only commit source code but also exert significant efforts on other communicative, organizational, and supportive activities. However, almost all prior research focuses on specific activities and fails to analyze elite developers' activities in a comprehensive way. To bridge this gap, we conduct an empirical study with fine-grained event data from 20 large open source projects hosted on GITHUB. We investigate elite developers' contributing activities and their impacts on project outcomes. Our analyses reveal three key findings: (1) elite developers participate in a variety of activities, of which technical contributions (e.g., coding) only account for a small proportion; (2) as the project grows, elite developers tend to put more effort into supportive and communicative activities and less effort into coding; and (3) elite developers' efforts in nontechnical activities are negatively correlated with the project's outcomes in terms of productivity and quality in general, except for a positive correlation with the bug fix rate (a quality indicator). These results provide an integrated view of elite developers' activities and can inform an individual's decision making about effort allocation, which could lead to improved project outcomes. The results also provide implications for supporting these elite developers.

[**Wolter2022**] Thomas Wolter, Ann Barcomb, Dirk Riehle, and Nikolay Harutyunyan. Open source license inconsistencies on GitHub. *ACM Transactions on Software Engineering and Methodology*, Dec 2022, DOI 10.1145/3571852.

[**Wyrich2022**] Marvin Wyrich, Justus Bogner, and Stefan Wagner. 40 years of designing code comprehension experiments: A systematic mapping study, 2022.

[**Young2021**] Jean-Gabriel Young, Amanda Casari, Katie McLaughlin, Milo Z. Trujillo, Laurent Hebert-Dufresne, and James P. Bagrow. Which contributions count? analysis of attribution in open source. In *Proc. International Conference on Mining Software Repositories (MSR)*. IEEE, May 2021, DOI 10.1109/msr52588.2021.00036.

Abstract: Open source software projects usually acknowledge contributions with text files, websites, and other idiosyncratic methods. These data sources are hard to mine, which is why contributorship is most frequently measured through changes to repositories, such as commits, pushes, or patches. Recently, some open source projects have taken to recording contributor actions with standardized systems; this opens up a unique opportunity to understand how community-generated notions of contributorship map onto codebases as the measure of contribution. Here, we characterize contributor acknowledgment models in open source by analyzing thousands of projects that use a model called All Contributors to acknowledge diverse contributions like outreach, finance, infrastructure, and community management. We analyze the life cycle of projects through this model's lens and con-

trast its representation of contributorship with the picture given by other methods of acknowledgment, including GitHub's top committers indicator and contributions derived from actions taken on the platform. We find that community-generated systems of contribution acknowledgment make work like idea generation or bug finding more visible, which generates a more extensive picture of collaboration. Further, we find that models requiring explicit attribution lead to more clearly defined boundaries around what is and is not a contribution.

[**Zakaria2022**] Farid Zakaria, Thomas R. W. Scogland, Todd Gamblin, and Carlos Maltzahn. Mapping out the hpc dependency chaos, 2022.

[**Zerouali2021**] Ahmed Zerouali, Tom Mens, and Coen De Roover. On the usage of JavaScript, python and ruby packages in docker hub images. *Science of Computer Programming*, 207:102653, Jul 2021, DOI `10.1016/j.scico.2021.102653`.

Abstract: Docker is one of the most popular containerization technologies. A Docker container can be saved into an image including all environmental packages required to run it, such as system and third-party packages from language-specific package repositories. Relying on its modularity, an image can be shared and included in other images to simplify the way of building and packaging new software. However, some package managers allow to include duplicated packages in an image, increasing its footprint; and outdated packages may miss new features and bug fixes or contain reported security vulnerabilities, putting the image in which they are contained at risk. Previous research has focused on studying operating system packages within Docker images, but little attention has been given to third-party packages. This article empirically studies installation practices, outdatedness and vulnerabilities of JavaScript, Python and Ruby packages installed in 3,000 popular community Docker Hub images. In many cases, these installed packages missed important releases leading to potential vulnerabilities of the images. Our findings suggest that maintainers of Docker Hub community images should invest more effort in updating outdated packages contained in those images in order to significantly reduce the number of vulnerabilities. In addition to this, Python community images are generally much less outdated and much less subject to vulnerabilities than NodeJS and Ruby community images. Specifically for NodeJS community images, elimination of duplicate package releases could lead to a significant reduction in their image footprint.

[**Zhang2022a**] Haiyin Zhang, Luís Cruz, and Arie van Deursen. Code smells for machine learning applications. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. ACM, May 2022, DOI `10.1145/3522664.3528620`.

Abstract: The popularity of machine learning has wildly expanded in recent years. Machine learning techniques have been heatedly studied in academia and applied in the industry to create business value. However, there is a lack of guidelines for code quality in machine learning applications. In particu-

lar, code smells have rarely been studied in this domain. Although machine learning code is usually integrated as a small part of an overarching system, it usually plays an important role in its core functionality. Hence ensuring code quality is quintessential to avoid issues in the long run. This paper proposes and identifies a list of 22 machine learning-specific code smells collected from various sources, including papers, grey literature, GitHub commits, and Stack Overflow posts. We pinpoint each smell with a description of its context, potential issues in the long run, and proposed solutions. In addition, we link them to their respective pipeline stage and the evidence from both academic and grey literature. The code smell catalog helps data scientists and developers produce and maintain high-quality machine learning application code.

[**Zhang2022b**] Kaiwen Zhang and Guanjun Liu. Automatically transform rust source to petri nets for checking deadlocks, 2022.

[**Zheng2019**] Wei Zheng, Chen Feng, Tingting Yu, Xibing Yang, and Xiaoxue Wu. Towards understanding bugs in an open source cloud management stack: An empirical study of OpenStack software bugs. *J. Syst. Softw.*, 151:210–223, May 2019, DOI 10.1016/j.jss.2019.02.025.