Software Developers' Work Habits and Expertise: Empirical Studies on Sketching, Code Plagiarism, and Expertise Development

Sebastian Baltes

Abstract Analyzing and understanding software developers' work habits and resulting needs is an essential prerequisite to improve software development practice. In our research, we utilize different qualitative and quantitative research methods to empirically investigate three underexplored aspects of software development: First, we analyze how software developers use sketches and diagrams in their daily work and derive requirements for better tool support. Then, we explore to what degree developers copy code from the popular online platform Stack Overflow without adhering to license requirements and motivate why this behavior may lead to legal issues for affected open source software projects. Finally, we describe a novel theory of software development expertise and identify factors fostering or hindering the formation of such expertise. Besides, we report on methodological implications of our research and present the open dataset SOTorrent, which supports researchers in analyzing the origin, evolution, and usage of content on Stack Overflow. The common goal for all studies we conducted was to better understand software developers' work practices. Our findings support researchers and practitioners in making data-informed decisions when developing new tools or improving processes related to either the specific work habits we studied or expertise development in general.

1 Introduction

A work habit, which is a "settled tendency or usual manner of behavior," can positively or negatively influence software developers' daily work. Knowing and understanding such work habits and resulting needs is an essential prerequisite to improve the existing software development processes and tools. However, the software engineering research community is often criticized for not addressing the

S. Baltes (✉)
The University of Adelaide, Adelaide, SA, Australia
e-mail: sebastian.baltes@adelaide.edu.au

M. Felderer et al. (eds.), *Ernst Denert Award for Software Engineering 2019*,
https://doi.org/10.1007/978-3-030-58617-1_4

problems that practitioners actually face during their work [1]. At the same time, software developers' beliefs are rather based on their personal experience than on empirical findings [2]. To fill this gap between academia and practice, we conducted several empirical studies investigating different aspects of *software developers' work habits and expertise*.

While the goal guiding all empirical studies we conducted was to gain a better understanding of software developers' work practices, we drew different conclusions for each of the studied phenomena: Based on our results, we developed novel tool prototypes to better support software developers' sketching and diagramming workflows, we reached out to developers to make them aware of possible licensing issues in their software projects due to code copied from Stack Overflow, and we provide recommendations for researchers, developers, and employers how to utilize our findings on software development expertise and its formation.

For the first part of this research project (see Sect. 2), we studied how software developers use *sketches and diagrams* in their daily work. At the time we started our research, an overall picture of developers' work habits related to the creation and usage of sketches and diagrams was missing. To fill this gap, we conducted an exploratory field study in different software companies, an online survey with software practitioners, and an observational study with software developers. We found that developers frequently create and use sketches and diagrams and that they consider many of those visual artifacts to be helpful in understanding related source code. However, we also identified a lack of tool support to archive and retrieve sketches and diagrams documenting different aspects of software systems. Thus, based on our findings, we derived requirements to better support developers' sketching and diagramming workflows and implemented those requirements in two tool prototypes, named *SketchLink* and *LivelySketches*, which we then evaluated in formative user studies.

The second part (see Sect. 3) presents an extensive empirical study on a rather negative work habit: We investigated to what degree developers adhere to Stack Overflow's license requirements when copying code snippets published on that platform—or, in other words, to what extent they commit *code plagiarism*. Since many developers use the online question-and-answer platform Stack Overflow on a daily basis [3], it is an essential part of their daily work life. If developers copy code snippets from that platform into their open source software projects without adhering to the corresponding license requirements, legal issues may arise. After describing the legal situation around Stack Overflow code snippets, we give a first estimate of how frequently developers copy such snippets into public GitHub projects without the required attribution, provide an analysis of licensing conflicts, and present results from an online survey, which suggest that many developers are not aware of the licensing situation and its implications. Besides publishing our empirical results, we reached out to owners of open source GitHub projects to make them aware of possible licensing conflicts in their projects.

In the third part of this research project (see Sect. 4), we present a first conceptual theory of *software development expertise* that is grounded in the related literature and three online surveys with software developers. The connection to work habits

is that, by learning from past experience, developers may adapt their work habits over time. Moreover, the existing habits related to self-improvement and learning may shape the path of an individual from being a novice toward being an expert in software development. Previously, the software engineering research community was lacking a comprehensive theory on what constitutes software development expertise and how such expertise is formed. Our theory describes important properties of software development expertise and factors fostering or hindering its formation, including how developers' performance may decline over time. Based on that theory, we provide recommendations for researchers who want to study expertise formation, developers who want to improve their software development skills, and employers who want to build a work environment supporting expertise development of their staff.

While the first three sections describe the main contributions of this research project, in Sect. 5, we reflect on *methodological and ethical issues* we faced when sampling software developers for the online surveys we conducted. The goal of that chapter is to inform the research community which strategies worked best for us, but we also want to start a discussion about the ethics of different sampling strategies that researchers currently use. To conclude this article, and to corroborate our *open data* efforts, we present the open dataset *SOTorrent*, which we created in the context of our research on Stack Overflow code snippets, described in Sect. 3. Besides explaining how we built the dataset, we use it to conduct a first analysis of the evolution of content on Stack Overflow and to investigate code snippets copied from external sources into Stack Overflow and duplicates of code snippets within Stack Overflow. We continue to maintain the dataset to support further research on the origin, evolution, and usage of content on Stack Overflow.

2 Sketching: Developers' Usage of Sketches and Diagrams in Practice

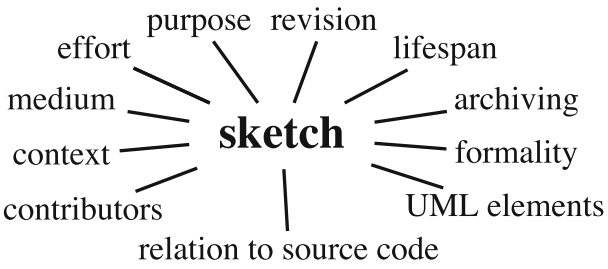

Fig. 1 The 11 dimensions of a sketch or diagram in software development that we used to structure and guide our research

Communication is omnipresent in software development. Requirements are communicated from prospective users to the developers implementing the software, the general architecture is communicated within the development team, developers communicate with each other during pair programming, and after deployment, issues are reported back to developers. Such information flows involve diverse channels [4], including face-to-face communication [5, 6], email [7], videoconferencing [5, 6], and team collaboration tools [8]. Especially in collocated settings, developers use informal sketches and diagrams for communication [9]. Those visual artifacts, spanning different types of media [10], support developers in designing new and understanding existing software systems [11]. Nevertheless, when we started our research on sketches and diagrams in software development practice, an overall picture of how developers use those visual artifacts was missing. Therefore, in the corresponding chapter of the dissertation, we first motivate our notion of sketch *dimensions* to capture the most important characteristics of visual artifacts used in software development (see Fig. 1) and then present the design and results of a mixed-methods study we conducted to investigate how software practitioners use such artifacts. Our research included an exploratory field study in three different software companies, an online survey with 394 participants, and an observational study with six pair programming teams. After describing the state of practice and resulting needs of software practitioners working with sketches and diagrams, we present two tool prototypes that we developed in response to the results of our empirical investigation. The content of this chapter is based on four peer-reviewed publications [10, 12, 13, 14].

Contributions

- A **characterization** of sketches and diagrams in software development practice, which is grounded in related work and in a field study we conducted in three different software companies.
- An assessment of 11 different dimensions of sketches and diagrams in software development using an **online survey** with 394 software practitioners.
- An analysis how developers communicate in a **pair-programming** setting when **locating performance bugs**, including an investigation of the **role of sketches** in this scenario.
- A presentation of two **tool prototypes** supporting software developers' sketching and diagramming workflows.

Overall, we found that software practitioners frequently create and use such visual artifacts. Our online survey revealed that sketches and diagrams are often informal but are considered to be a valuable resource, documenting many aspects of the software development workflow. We showed how sketches are related to source code artifacts on different levels of abstraction and that roughly half of them were rated as helpful to understand the source code. As documentation is frequently poorly written and out of date, sketches could fill in this gap and serve as a supplement to conventional documentation such as source code comments or other textual resources. The majority of sketches and diagrams were created on analog media such as paper or whiteboard. Many of them were archived, but

our survey participants also named technical issues, for example, that there is no good technique to keep (digital versions of) sketches together with source code. In response to this observation, we developed the tool prototype *SketchLink*, which assists developers in archiving and retrieving sketches related to certain source code artifacts. Regarding the evolution of sketches, our qualitative results indicated that it is a common use case for sketches to be initially created on analog media like paper or whiteboards and then, potentially after some revisions, they end up as an archived digital sketch. To support such workflows, we developed a second tool prototype named *LivelySketches*, which supports transitions from analog to digital media and back. One direction for future work is to merge the features of both prototypes and evaluate the resulting tool in larger context. Moreover, with techniques such as *graphic facilitation* and *sketchnoting* becoming more and more popular, analyzing potential use cases for those techniques in software development projects emerged as another direction for future work. We already interviewed graphic facilitators who worked in software development projects, but also software developers and architects with sketching experience. Based on those interviews, we will derive recommendations for applying visualization techniques in different phases of software development projects.

3 Code Plagiarism: Stack Overflow Code Snippets in GitHub Projects

Stack Overflow is the most popular question-and-answer website for software developers, providing a large amount of copyable code snippets. Using those snippets raises maintenance and legal issues. Stack Overflow's license (CC BY-SA) requires attribution, that is referencing the original question or answer, and requires derived work to adopt a compatible license. While there is a heated debate on Stack Overflow's license model for code snippets and the required attribution, little is known about the extent to which snippets are copied from Stack Overflow without proper attribution. To fill this gap, we conducted a large-scale empirical study analyzing software developers' usage and attribution of non-trivial Java code snippets from Stack Overflow answers in public GitHub projects. We followed three different approaches to triangulate an estimate for the ratio of unattributed usages and conducted two online surveys with software developers to complement our results. For the different sets of GitHub projects that we analyzed, the ratio of projects containing files with a reference to Stack Overflow varied between 3.3 and 11.9%. We found that at most 1.8% of all analyzed repositories containing code from Stack Overflow used the code in a way compatible with CC BY-SA. Moreover, we estimate that at most a quarter of the copied code snippets from Stack Overflow are attributed as required. Of the surveyed developers, almost one half admitted copying code from Stack Overflow without attribution and about two-thirds were not aware of the license of Stack Overflow code snippets and its implications.

The content of this chapter is based on a peer-reviewed journal publication [15]. Moreover, some results have also been published in an extended abstract before [16].

Contributions

- A thorough description of the **legal situation** around Stack Overflow code snippets.
- A **triangulated estimation** of the **attribution ratio** of Stack Overflow code snippets in public GitHub projects.
- An analysis of possible **licensing conflicts** for the GitHub projects containing code from Stack Overflow.
- A qualitative analysis of **how developers refer to** Stack Overflow content.
- An **online survey** suggesting that many **developers are not aware** of the licensing of Stack Overflow code snippets and its implications.

Our research revealed that at most one quarter of the code snippets copied from Stack Overflow into public GitHub Java projects are attributed as required by Stack Overflow's license (CC BY-SA). Moreover, we found that between 3.3 and 11.9% of the analyzed GitHub repositories contained a file with a reference to Stack Overflow (see Table 1). We identified only 1.8% of the GitHub projects with copies of Stack Overflow code snippets to attribute the copy and to use a license that is share-alike compatible with Stack Overflow's license. For the other 98.2% of the projects, especially the share-alike requirement of CC BY-SA may lead to licensing conflicts. Two online surveys have shown that many developers admit copying code from Stack Overflow without attribution. We also found that many of them are not aware of the licensing situation and its implications. In the course of our research on Stack Overflow code snippets, we built the *SOTorrent* dataset (see Sect. 6), which we continue to maintain. Beside closely following how other researchers use the dataset to study different questions related to code on Stack Overflow, we will continue to investigate how such code snippets are maintained and how their evolution can be better supported. Another direction for future work, which is not limited to Stack Overflow, is to build better tool support for developers dealing with online code snippets. On the one hand, continuous integration tools could check whether commits add non-trivial code snippets from online resources to a project; on the other hand, tools could support developers in understanding license compatibility not only for whole software libraries, but also on the level of individual code snippets copied from online resources. Those efforts can help mitigating legal threats for open source projects that intentionally or unintentionally use code from diverse sources.

Table 1 Summary of results regarding attribution of snippets copied from Stack Overflow (SO): distinct references to answers (A) or questions (Q) on SO in the Java files from GitHub analyzed in each phase of our research; number of analyzed files and repositories, files/repos containing a reference to SO, files/repos containing a copy of a SO snippet, attributed copies of SO snippets

	References		Files				Repositories		
Ph.	A	Q	Count	Ref	Copy	Attr	Count	Ref	Copy
1	5014	16,298	13.3m	18,605	4198	402	336k	11,086	3291
	23.5%	76.5%		0.09%	0.03%	9.6%		3.3%	1.0%
2	209	463	445k	634	297	70	2313	274	199
	31.1%	68.9%		0.14%	0.07%	23.6%		11.9%	8.6%
3	1551	4843	1.7m	5354	1369	104	64,281	3536	1332
	24.3%	75.7%		0.31%	0.08%	7.6%		5.5%	2.1%

4 Expertise Development: Toward a Theory of Software Development Expertise

Software development includes diverse tasks such as implementing new features, analyzing requirements, and fixing bugs. Being an expert in those tasks requires a certain set of skills, knowledge, and experience. Several studies investigated individual aspects of software development expertise, but what is missing is a comprehensive theory. In this chapter, we present a first conceptual theory of software development expertise that is grounded in data from a mixed-methods survey with 335 software developers (see gray boxes in Fig. 2) and in the literature on expertise and expert performance. Our theory currently focuses on programming but already provides valuable insights for researchers, developers, and employers. The theory describes important properties of software development expertise and which factors foster or hinder its formation, including how developers' performance may decline over time. Moreover, our quantitative results show that developers' expertise self-assessments are context-dependent and that experience is not necessarily related to expertise. The content of this chapter is based on a peer-reviewed publication [17].

Contributions

- A first **conceptual theory** of **software development expertise** grounded in a survey with 335 software developers and in the literature on expertise and expert performance.
- Quantitative results that point to the **context-dependence** of software developers' **expertise self-assessments**.
- A **theory-building approach** involving inductive and deductive steps that other (software engineering) researchers can apply or adapt (see Fig. 2).

With our research, we identified different characteristics of software development experts, but also factors fostering or hindering the formation of software development expertise. Besides building a first conceptual theory, we found that

expertise self-assessments are context-dependent and do not always correspond to experience measured in years. Researchers can use our findings when designing studies involving expertise self-assessments. We recommend to explicitly describe what distinguishes novices from experts in the specific setting under study when asking participants for expertise self-assessments. Our theory enables researchers to design new experiments, but also to re-evaluate results from previous experiments. Software developers can use our results to learn which properties are distinctive for experts in their field, and which behaviors may lead to becoming a better software developer. For example, the concept of deliberate practice, and in particular having challenging goals, a supportive work environment, and getting feedback from peers are important factors. For "senior" developers, our results provide suggestions for being a good mentor. Mentors should know that they are considered to be an important source for feedback and motivation, and that being patient and being open-minded are desired characteristics. We also provide first results on the consequences of age-related performance decline, which is an important direction for future work. Employers can learn what typical reasons for demotivation among their employees are, and how they can build a work environment supporting the self-improvement of their staff. Besides obvious strategies such as offering training sessions or paying for conference visits, our results suggest that employers should think carefully about how information is shared between their developers and also between development teams and other departments of the company. Finally, employers should make sure to have a good mix of continuity and change in their software development process because non-challenging work, often caused by tasks becoming routine, is an important demotivating factor for software developers. One important direction for future work, which emerged in the course of our research, is *the role of older software developers*. Especially in industrialized countries, the demographic change leads to an older work force, since people are expected to retire later. Still, the challenges that older developers face in a competitive field like software development are largely unknown. Our study participants already mentioned different age-related challenges. We plan to study those challenges to be able to mitigate them where possible, preventing those experienced developers from dropping out of software development.

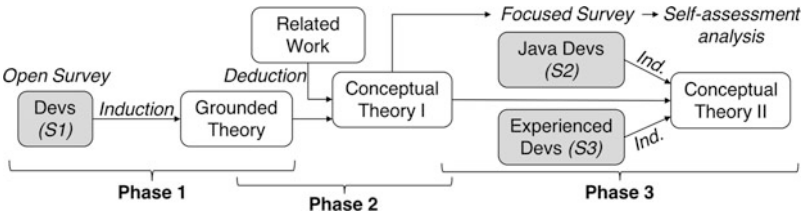

Fig. 2 Structure of our iterative theory-building approach

5 Methodological Insights: Issues in Sampling Software Developers

Online surveys like the ones we conducted for this research project are considered to be a feasible means for investigating the state of practice [18]. In particular, surveys are an important empirical method used in software engineering (SE) research that can be employed to explore and describe various characteristics of a broad population [19]. However, reaching professional software developers with surveys is a difficult task. Except for single companies or institutions that allow researchers to use a list of their employees, random sampling of software developers is impossible most of the time. Researchers therefore often rely on the available subjects, which is known as *convenience sampling*. Applying non-random sampling techniques like convenience sampling may lead to biased samples with limited external validity. To mitigate the threats to external validity, researchers need detailed knowledge about the population of software developers they want to target, but this information is often not available. Further, some of the sampling techniques that researchers employ raise ethical concerns, such as contacting developers on GitHub using email addresses users did not provide for this purpose. In this chapter, we summarize what we learned while conducting online surveys with software developers. The content of this chapter is based on a peer-reviewed publication [20].

Contributions

- **Experience reports** for different survey sampling strategies.
- Presentation of the idea of a **systematic database with software developer demographics** to assess the external validity of surveys conducted using non-random sampling techniques.
- Building awareness about **ethical issues** that may arise with sampling approaches that researchers currently utilize.

We found that the most efficient and effective sampling strategies were to use public media and "testimonials" that advertise the survey. We also highlighted the importance of gatekeepers who provide access to companies or communities. Another finding is that, to be able to assess the external validity of studies involving non-random samples, researchers need a collection of typical software developer demographics, which currently does not exist. Using a systematic literature review, one could collect published demographics about developers. Further, authors of studies with software developers could be contacted and asked to provide basic demographic information about their participants, if available. This information, together with the data from the Stack Overflow developer surveys, would be a solid basis to assess the external validity of future studies. Conferences and journals may recommend authors to describe certain key demographics for published studies, and reviewers could motivate authors to explicitly address their sampling approach and effects on the generalizability of their results. We also pointed at ethical issues with some of the sampling techniques researchers currently employ, in particular using email addresses collected from GitHub to contact developers.

6 Open Data: Building and Maintaining the SOTorrent Dataset

For all studies conducted in the context of this research project, we provide supplementary material packages that enable other researchers to reproduce our results. Besides publishing (anonymized) data on the preserved archive *Zenodo*, we also published the software and scripts used to retrieve and analyze that data. Moreover, pre- or postprints of all papers are available online. Beside these general open science efforts, in this chapter, we want to particularly highlight the open dataset *SOTorrent*[1] that we created to support future research about code snippets on Stack Overflow. The dataset allows researchers to investigate and understand the evolution of Stack Overflow content on the level of individual text and code blocks, which is not possible with the official data dump that Stack Overflow provides. Beside supporting our own research, we published and promoted the dataset to be used by other researchers. Those efforts resulted in the dataset being selected as the official mining challenge of the *16th International Conference on Mining Software Repositories (MSR 2019)* [21].

The content of this chapter is based on two peer-reviewed publications: One full paper describing the creation of *SOTorrent* and first analyses using the dataset [20] as well as our accepted mining challenge proposal [21]. Moreover, we present additional analyses that we conducted for an upcoming journal extension of our initial *SOTorrent* paper (see research questions three and four).

Contributions

- An open **dataset** that allows researchers to investigate and understand the evolution of Stack Overflow posts and their relation to other platforms such as GitHub.
- A thorough **evaluation of 134 string similarity metrics** regarding their applicability for reconstructing the version history of Stack Overflow text and code blocks.
- A **first analysis of the evolution** of content on Stack Overflow, including the description of a close relationship between post edits and comments.
- An **analysis of code clones** on Stack Overflow together with an investigation of possible licensing risks.

The *SOTorrent* dataset has allowed us to study the phenomenon of post editing on SO in detail. We found that a total of 13.9 million SO posts (36.1% of all posts) have been edited at least once. Many of these edits (44.1%) modify only a single line of text or code, and while posts grow over time in terms of the number of text and code blocks they contain, the size of these individual blocks is relatively stable. Interestingly, only in 6.1% of all cases are code blocks changed without corresponding changes in text blocks of the same post, suggesting that SO users

[1] http://sotorrent.org.

typically update the textual description accompanying code snippets when they are edited. We also found that edits are mostly made shortly after the creation of a post (78.2% of all edits are made on the same day when the post was created), and the vast majority of edits are made by post authors (87.4%)—although the remaining 12.6% will be of particular interest for our future work. The number of comments on posts without edits is significantly smaller than the number of comments on posts with edits, suggesting an interplay of these two features. We find evidence which suggests that commenting on a post on SO helps to bring attention to it. Of the comments that were made on the same day as an edit, 47.9% were made before an edit and 52.1% afterwards, typically (median value) only 18 min before or after the edit. Motivated by this quantitative analysis of the temporal relationship between edits and comments, we conducted a qualitative study and developed a visual analysis tool to explore the communication structure of SO threads. Our analysis using this tool revealed several communication and edit patterns that provide further evidence for the connection between post edits and comments. We found comments which explain, trigger, and announce edits as well as content overlap between edits and comments. The fact that SO users rely on the commenting feature to make others aware of post edits—and in some cases even duplicate content between comments and posts—suggests that users are worried that content evolution will be missed if it is buried in a comment or has been added to a post later via an edit. At the same time, we found evidence that edits can play a vital role in attracting answers to a question. In our future work, we will explore how changes to Stack Overflow's user interface could make the evolution of content more explicit and remove the need for users to repurpose the commenting feature as an awareness mechanism. Besides, we investigated code clones on SO, revealing that, just like in regular software projects, code clones on SO can affect the maintainability of posts and lead to licensing issues. Depending on the outcome of the discussion we started on Stack Overflow Meta, we plan to implement means to add the missing attribution to posts and mark threads as related based on the similarity of the code blocks they contain.

7 Summary and Future Work

In this research project, we utilized diverse research designs to empirically investigate yet underexplored aspects of *software developers' work habits and expertise*. We started by analyzing how developers use *sketches and diagrams* in their daily work, then derived requirements for tool support, and finally implemented and evaluated two tool prototypes. In a second research project, we investigated how common it is for developers to *copy non-trivial code snippets* from the popular question-and-answer platform Stack Overflow into open source software projects hosted on GitHub, without adhering to the terms of Stack Overflow's license. In that project, we also assessed developers' awareness of the licensing situation and its implications. While those first two research projects can be regarded as analyses of a rather positive and a rather negative work habit, the third project aimed at analyzing

behaviors that may lead to developers becoming experts in certain software development tasks. However, we not only identified factors influencing expertise formation over time but also developed a first conceptual theory structuring the broad concept of *software development expertise*.

In the complete version of the dissertation that this article is based on, we not only present the designs and results of the different empirical studies we conducted, but also highlighted how we use those results to guide further actions. We already used our empirical results to: (1) motivate and implement novel tools to support software developers' sketching workflows, (2) inform developers about possible licensing issues in their open source software projects, (3) build a first conceptual theory of software development expertise that researchers as well as practitioners can use, (4) point to the underexplored phenomenon of age-related performance decline, (5) grow awareness in the research community about ethical implications of certain sampling strategies, motivated by participants' feedback, and (6) create an open dataset that the research community can use for future research projects on Stack Overflow content. Our work supports researchers and practitioners in making data-informed decisions when developing new tools or improving processes related to either the specific work habits we studied or expertise development in general.

References

1. Lionel Briand. Embracing the Engineering Side of Software Engineering. *IEEE Software*, July/August:92–95, 2012.
2. Prem Devanbu, Thomas Zimmermann, and Christian Bird. Belief & Evidence in Empirical Software Engineering. In Laura Dillon, Willem Visser, and Laurie Williams, editors, *38th International Conference on Software Engineering (ICSE 2016)*, pages 108–119, Austin, TX, USA, 2016. ACM.
3. Stack Exchange Inc. Stack Overflow Developer Survey Results 2018. https://insights.stackoverflow.com/survey/2018, 2018.
4. Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M. German. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.
5. Hayward P. Andres. A comparison of face-to-face and virtual software development teams. *Team Performance Management: An International Journal*, 8(1/2):39–48, 2002.
6. Daniela Damian, Armin Eberlein, Mildred L.G. Shaw, and Brian R. Gaines. Using Different Communication Media in Requirements Negotiation. *IEEE Software*, May/June:28–36, 2000.
7. Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining Email Social Networks. In Stephan Diehl, Harald C. Gall, and Ahmed E. Hassan, editors, *3rd International Workshop on Mining Software Repositories (MSR 2006)*, pages 137–143, Shanghai, China, 2006. ACM.
8. Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. Why Developers are Slacking Off: Understanding How Software Teams Use Slack. In Darren Gergle and Meredith Ringel Morris, editors, *19th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW 2016): Companion*, pages 333–336, New York, NY, USA, 2016. ACM.

9. Uri Dekel and James D. Herbsleb. Notation and representation in collaborative object-oriented design: An observational study. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *22nd ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007)*, pages 261–280, Montreal, QC, Canada, 2007. ACM.
10. Sebastian Baltes and Stephan Diehl. Sketches and diagrams in practice. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, pages 530–541, Hong Kong, China, 2014. ACM.
11. Mauro Cherubini, Gina Venolia, Robert DeLine, and Andrew J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In Mary Beth Rosson and David J. Gilmore, editors, *2007 Conference on Human Factors in Computing Systems (CHI 2007)*, pages 557–566, San Jose, CA, USA, 2007. ACM.
12. Sebastian Baltes, Peter Schmitz, and Stephan Diehl. Linking sketches and diagrams to source code artifacts. In Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey, editors, *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, pages 743–746, Hong Kong, China, 2014. ACM.
13. Sebastian Baltes, Fabrice Hollerich, and Stephan Diehl. Round-Trip Sketches: Supporting the Lifecycle of Software Development Sketches from Analog to Digital and Back. In Kang Zhang, Ivan Beschastnikh, and Andrea Mocci, editors, *2017 IEEE Working Conference on Software Visualization (VISSOFT 2017)*, pages 94–98, Shanghai, China, 2017. IEEE.
14. Sebastian Baltes, Oliver Moseler, Fabian Beck, and Stephan Diehl. Navigate, Understand, Communicate: How Developers Locate Performance Bugs. In Qing Wang, Guenther Ruhe, Jeff Carver, and Oscar Dieste, editors, *9th International Symposium on Empirical Software Engineering and Measurement (ESEM 2015)*, pages 225–234, Beijing, China, 2015. IEEE.
15. Sebastian Baltes and Stephan Diehl. Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects. *Empirical Software Engineering*, Online First:1–37, 2018.
16. Sebastian Baltes, Richard Kiefer, and Stephan Diehl. Attribution required: Stack overflow code snippets in GitHub projects. In Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard, editors, *39th International Conference on Software Engineering (ICSE 2017), Companion Volume*, pages 161–163, Buenos Aires, Argentina, 2017. IEEE Computer Society.
17. Sebastian Baltes and Stephan Diehl. Towards a Theory of Software Development Expertise. In Gary Leavens, Alessandro Garcia, and Corina Pasareanu, editors, *26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*, pages 187–200, Lake Buena Vista, FL, USA, 2018. ACM.
18. Aileen Cater-Steel, Mark Toleman, and Terry Rout. Addressing the Challenges of Replications of Surveys in Software Engineering Research. In Ross Jeffery, June Verner, and Guilherme H. Travassos, editors, *2005 International Symposium on Empirical Software Engineering (ISESE 2005)*, pages 10–pp, Noosa Heads, Queensland, Australia, 2005. IEEE.
19. Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Chapter 11: Selecting Empirical Methods for Software Engineering Research. In Forrest Shull, Janice Singer, and Dag I.K. Sjoberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, London, UK, 2008.
20. Sebastian Baltes and Stephan Diehl. Worse Than Spam: Issues In Sampling Software Developers. In Marcela Genero, Andreas Jedlitschka, and Magne Jorgensen, editors, *10th International Symposium on Empirical Software Engineering and Measurement (ESEM 2016)*, pages 52:1–52:6, Ciudad Real, Spain, 2016. ACM.
21. Sebastian Baltes, Christoph Treude, and Stephan Diehl. SOTorrent: Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets. In Margaret-Anne Storey, Bram Adams, and Sonia Haiduc, editors, *16th International Conference on Mining Software Repositories (MSR 2019)*, Montreal, QC, Canada, 2019. IEEE.