Open Data-driven Usability Improvements of Static Code Analysis and its Challenges

Söderberg, Emma; Church, Luke; Höst, Martin

Published in:
EASE'21: Evaluation and Assessment in Software Engineering

DOI:
10.1145/3463274.3463808

2021

Open Data-driven Usability Improvements of Static Code Analysis and its Challenges

Emma Söderberg
emma.soderberg@cs.lth.se
Lund University
Lund, Sweden

Luke Church
luke@church.name
University of Cambridge
Cambridge, United Kingdom
Lund University
Lund, Sweden

Martin Höst
martin.host@cs.lth.se
Lund University
Lund, Sweden

ABSTRACT

Context: Software development is moving towards a place where data about development is gathered in a systematic fashion in order to improve the practice, for example, in tuning of static code analysis. However, this kind of data gathering has so far primarily happened within organizations, which is unfortunate as it tends to favor larger organizations with more resources for maintenance of developer tools. **Objective**: Over the years, we have seen a lot of benefits from open source and recently there has been a lot of development in open data. We see this as an opportunity for cross-organisation community building and wonder to what extent the views on using and sharing open source software developer tools carry across to open data-driven tuning of software development tools. **Method**: An exploratory study with 11 participants divided into 3 focus groups discussing using and sharing of static code analyzers and data about these analyzers. **Results**: While using and sharing open-source code (analyzers in this case) is perceived in a positive light as part of the practice of modern software development, sharing data is met with skepticism and uncertainty. Developers are concerned about threats to the company brand, exposure of intellectual property, legal liabilities, and to what extent data is context-specific to a certain organisation. **Conclusions**: Sharing data in software development is different from sharing data *about* software development. We need to better understand how we can provide solutions for sharing of software development data in a fashion that reduces risk and enables openness.

CCS CONCEPTS

• **Information systems → Open source software**; • **Software and its engineering → Software verification and validation**.

KEYWORDS

data-driven software development, open data, static code analysis

ACM Reference Format:
Emma Söderberg, Luke Church, and Martin Höst. 2021. Open Data-driven Usability Improvements of Static Code Analysis and its Challenges. In *Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3463274.3463808

1 INTRODUCTION

Static code analysis is one approach in software quality assurance, where code is analyzed by an analyzer program statically without running the code. These programs compute metrics, check adherence to coding standards, try and locate common mistakes etc. This checking can be initiated manually, or automatically as part of the build process. It will generate a report often in terms of errors and warnings to be taken care of by software engineers in some way before other activities can proceed.

A number of factors increase the complexity of code analysis. The variation in software development processes in different projects is high, which means that there are different needs in different projects. The quality requirements are different in different projects, which also means that there are different needs in different projects. There is further some organisational cost to the deployment of static analysis including the integration of it into workflows and processes and the effort needed to address the generated results including false positives. To decrease the organisational cost of the deployment analyzers are often tuned as to which issues they report in which contexts, i.e., an example of data-driven decision making in the development process. Data-driven software engineering in general means utilizing data from artifacts and processes to understand software system development (e.g. [3]), and in this case data comes from developer usage, and the affected artifact is the analyser. To some extent it also resembles how data from product usage is used in decisions about features (e.g., [19]), although the investigated product is the analyser.

However, this kind of data gathering for analyzers has so far primarily happened within single organizations [16, 25], and limits the application of the data to centralised development organisations. This raises the question if it is possible to build communities of companies and professionals who contribute data to each other for tuning analyzers, in a way that corresponds to how Open Source Software (OSS) communities collaborate providing much broader access to the potential benefits that such systems provide.

In OSS development, there are accepted ways of sharing results in communities [9], and the term "open data" [8] denotes a trend in which for example public organizations provide data to other organizations. In this paper, we analyse to what extent the view on OSS and open data carry across to data for open data-driven tuning of software development tools. Analyzers and similar tools are normally obtained from external sources, either as commercially licensed products or OSS tools. If static code analysis systems depend on data for tuning, and not only on the source code of the software, the traditional models of OSS sharing no longer feel sufficient. This represents a broad challenge in the open construction of modern socio-technical infrastructure that depends on both source code and data. We investigate this challenge in the context of the development and tuning of analyzers.

We observe an openness from companies to engage with OSS for software development tools, like the Gerrit code review tool, which are often seen as commodity [18], but despite this openness it is not as clear to what extent data is shared between companies for software developer tools. An initial hypothesis was that many organizations use the same analysis tools, and, for example, patterns of false positives could be seen as a non strategic type of information and shared with other organizations, since handling of false positives is a well known industrial problem (e.g., [10, 21]).

In the specific context of static code analysis, we investigate the benefits and challenges perceived by professionals. The concrete example we used to motivate this work was the Tricorder System [25] used by Google, which tackles some of the usability problems of static code analysis by construction of a feedback loop built on usability data gathered from its group of software developers, but has limited applicability outside of Google. To shed more light on challenges in trying to expand on this approach beyond one organisation, we present results from an exploratory study with practitioners conducted via three focus groups. We report on trends we see in the gathered data, and outline directions for future work.

We see the sharing of data about analyzers as an interesting direction for the software development community, allowing organizations to collaborate, and especially smaller businesses to reap some of the rewards available to larger organisations with a large population of developers contributing data internally. In this study, analyzers act as an exemplar for modern socio-technical infrastructure.

2 BACKGROUND AND RELATED WORK

In this section we provide some background and related work for the two main areas relevant for work presented in the paper; data-driven usability improvements of static code analysis, open and shared data.

2.1 Data-driven Tuning of Static Code Analysis

Use of static code analysis is prevalent in software development. Vassalo et al. report static code analyzers like FindBugs [2] (now SpotBugs), Checkstyle [1], PMD [7], SonarQube[2], and Error Prone [1] to be used on a daily basis for quality assurance [28]. Analyzer frameworks like these typically provide a list of checks connected to how a programming language is used (e.g., use of Java), but can then be extended to provide custom checks. Checks beyond general language use may, for instance, be of interest to the specific needs of an organisation (e.g., a style guide check), or the specific needs of an API (e.g., checking for anti-patterns in API use). Altogether, this results in an ever-growing list of analyzers.

At the same time as the adoption of analyzers is increasing, so are reports of issues with analyzer use. Ayewah et al. report on the use of FindBugs [2], where they see benefits using the tool but also see problems with low impact warnings and developers suppressing warnings. Johnson et al. [13] later further highlighted these issues in a study where they investigate why developers are not using static analysis tools to find bugs. In this study, they also mention additional issues with lack of integration of analyzers into the developer workflow and issues with lack of precision in analyzer results (false positives). In a recent qualitative study of StackOverflow posts about static analysis, Imtiz et al. [12] found that the most common question about static analysis was about how to ignore the results.

Software development has a wide range of techniques for building feedback mechanisms using empirical data. However many of these such as user studies are fairly expensive, difficult to scale, and difficult to gather contextually sensitive data on relatively infrequently occurring events. An alternative strategy is to incorporate gathering of user feedback into the developer workflow where static analysis results are used [6]. This later strategy reduces the qualitative depth of the analysis but scales to many users. This approach is the one used by Tricorder, where analyzer results are presented as robot comments next to human comments in code review at Google. Each presented robot comment is equipped with a "not useful" button where users can click if they find a comment to not be useful.

This data gathering creates a feedback loop to the maintainers of analyzers who can react quickly to tune analyzers to reduce, for instance, false positives or improve incomprehensible messages. Ljungberg et al. recently re-implemented the Tricorder approach as open-source on an open-tools stack in order to experiment with data-driven deployment of static code analysis [16]. The latter study reaffirmed the usefulness of creating this kind of feedback loop between maintainers and users, especially as new analyzer functionality is rolled out where there is more need for tuning, however the work didn't address how this tuning might be shared between organisations in an open manner.

2.2 Open Source and Open Data

When it comes to OSS software (e.g. [9]), much research has been presented on how to use it as components in software development projects, for example, Höst et al. [11] present a review of research on open source as part of commercial software development. Today, inclusion of OSS components in products is seen as a natural and to some extent necessary part of product development.

Shahrivar et al. [26] investigate what the characteristics of a commercial OSS business model are in a systematic literature review. They find that business models include products and complementarities, clients and users, and competitive strategies, which shows that sharing of OSS is part of many organizations' business.

The question of sharing OSS systems with organizations outside the creating organization has also been studied in different research studies. For example, Linåker et al. [15] present objectives and "complexities" of sharing OSS with other organizations are identified

[1]https://checkstyle.sourceforge.io/
[2]https://www.sonarqube.org/

in a multiple case study involving three organizations. Examples of objectives include objectives related to reputation, and objectives to standardize or build eco-systems. Examples of complexities include misalignment between internal and external agendas, sensitive IPR, security threats, and cost. Munir et al. [18] presents a case study highlighting how participation in OSS for software developer tools, in this case the code review tool Gerrit and the continuous integration tool Jenkins, can facilitate innovation with outcomes such as free features and quality improvements to development tools. This significant amount of research about the commercial use and sharing of OSS can be seen as a basis, or inspiration, for research on commercial sharing and usage of open data, perhaps especially for software development tools.

The questions about open data is relevant for several domains and topics as seen in a recent systematic mapping study [8]. Even if the most common domain is "infomediaries", i.e. "workers, companies and investors who work in identifying and leveraging the market value of consuming information" [8] primary studies from a wide spectrum of domains, such as health, education, and culture, are identified. The topic of Software Engineering is one identified topic out of 11 with 19% of the identified primary studies. Within that topic the most common domain is infomediaries but there are also studies related to e.g., "geospatial" and education.

One area, related to Software Engineering, where there are more mature procedures for sharing data is in the security domain. For example, in [24] a framework and standard for sharing cybersecurity information is presented, with the intention to allow for sharing at a global scale. A more recent example is presented in [27] where award strategies for cybersecurity information sharing are studied with game theory. Related to this is the will to share vulnerabilities in an organisation's own products with other companies using the company's products, where at least some organizations are willing, but actual practice is more reactive than proactive [20].

Even if there are examples of areas where data is shared, like security described above, it is not extensively investigated in research. However, when Capilla et al. [5] recently investigated opportunities for reuse in Software Engineering, open data reuse was identified as one topic. Runeson [22] argues that data in Software Engineering can be divided into commodity, differentiating, and innovative, and proposes a research agenda for open collaborative data in Software Engineering. Beyond the division, infrastructure, licensing, governance, and privacy of open collaborative data are also presented as important topics.

3 METHODOLOGY

In order to understand potential routes to broaden the range of contexts in which data-driven tuning approaches can be used to improve analysers we seek to explore the following research question (**RQ$_1$**): *How do the views of using and sharing open source software developer tools carry across to open data-driven tuning of software development tools?* Given the exploratory nature of our study, we gathered opinions via focus groups, a method that has been deemed suitable for trying to gauge the state of mind among participants in an area with not yet established views [14]. Our assessment is that the views on how to engage with open data is such an area, with few established views.

Using the specific case of data-driven improvement of program analysis, we constructed a protocol for the focus groups with questions focusing on benefits and challenges when: (**Q$_1$**) **Using shared program analyzers?** (as in using program analyzers shared by others as open source). (**Q$_2$**) **Sharing program analyzers?** (as in sharing their own program analyzers as open source, or contributing to existing such open source projects). (**Q$_3$**) **Using shared program analyzer data?** (as in using data about program analyzer use, with focus on 'not useful' feedback, from another organisation), (**Q$_4$**) **Sharing program analyzer data?** (as in sharing your organisation's data about program analyzer use, with focus on 'not useful' feedback, to other organisations).

Data gathering All focus groups were carried out in connection to one university outreach workshop connecting the university with a network of companies in the same region. The workshop ran over two hours, with presentations during the first hour and focus groups during the second. The presentations were all centered around data-driven program analysis deployment similar to the approach in the Tricorder system [25], and recently MEAN [16]. The final part of the presentation concluded with an introduction of the questions to be asked in the focus groups.

After the presentations, the workshop participants who opted to participate in the focus groups were divided into 3 parallel focus groups carried out in video chat rooms (Zoom breakout rooms). The authors of this paper chaired one focus group each. We had a total of 11 participants representing 8 companies and 1 university, excluding facilitators, divided between the focus groups. The composition of the focus groups was primarily based on an even size distribution, but care was taken to spread participants from the same company between groups. Each focus group was recorded after informed consent from participants, and transcribed by the author leading each group. One group was held in Swedish and subsequently translated, the other two were held in English. We will refer to participants using an identifier P, for a person mentioned in a focus group, followed by a number. That is, P0-P14 were present or mentioned in the focus groups, and of those P1-P11 and P13-P14 were participants, P10 was a non-participant mentioned, and the authors leading the focus groups were P0, P8, and P12. We will refer to companies mentioned in a focus group using an identifier C followed by a number.

In the group of participants, there were 8 developers (P1, P2, P3, P4, P6, P8, P9, P11, and P13) and 3 managers (P5, P7, and P14). Several of the participants have experience of working with static code analyzers, some as engineers and others as managers responsible for development processes and related tools. Experience of developers ranged from junior developers to senior developers and architects with several years of experience. Managers were rather senior, with roles as product managers and architect and department managers.

Analysis We analysed the study to produce two categories of results; the first is a summary of the overall opinion space of the participants with respect to the using and sharing of analysers and data, and the second to produce a richer substantive dataset where we could use the coding scheme as an index into the data to locate discussions of interest. In order to do this, we used two separate coding schemes. In the summary analysis we coded for each participant (e.g. P1) and topic (e.g. Use of Analyzers), whether the participant expressed positive benefits that could be gained or

Table 1: Defined codes

Code area	*Codes*
Situational	Using other analyzer, Sharing analyzer, Using data from others, Sharing data with others
Software	Software Functionality, Non-Functional Property of Software (Security), Non-Functional Property of Software (Other)
Business	Cost (Software), Ecosystem & Collaboration, Team, Innovation, Standards, Competition, Legal issues, IPR, Brand, Other challenge (business), Other benefit (business)
Data	Cost (Data), Value proposition (Data), Privacy, Other challenge (data), Other benefits (data)

made a statement that they were already engaged in the activity (coded "Benefit"), or a statement of a problem or a statement that they would not engage in the activity (coded "Challenge"), or expressed both benefit and challenge (coded "Both"). Any participant that did not express any of the above about a topic was coded as "No Opinion".

For the second thematic signposting approach we first carried out an informal survey to build a tentative coding frame. To do this we gathered a pool of papers found based on our knowledge in the field and an informal search following of references. From this initial pool we picked three papers (focusing especially on adopting OSS [17], sharing OSS [15], and open data in SE [23]) which all of the authors read and each separately extracted codes from, which were then merged and reduced until a final list of codes was reached, presented in Table 1. The codes are divided into the areas of *situational*, i.e. the situations relating to the research questions $\mathbf{Q_1}$–$\mathbf{Q_4}$ above, and *substantive* codes with respect to software, business, and data. Codes were assigned using the Coda [4] tool originally designed for rapid interpretation of short excerpts of qualitative data. The translated transcripts of the text were coded one sentence at a time, and each time a code was given, both a situational code and a substantive code were applied.

For the overall summary analysis the transcripts were coded by the first and second author and an inter-rater reliability test was performed, giving a Cohen's Kappa of 0.931 (95% confidence interval, 0.839 to 1.000). The remaining differences were discussed and a consensus view agreed. For the substantive signpost coding all three authors coded each of the three transcripts. We did a consistency check of codes after a synchronous pilot coding session up to 200 and then we checked for consistency and discussed any differences. This check resulted in a small number of changes around behavior concerning not relevant messages. We proceeded by increasing the number of codes we could give a single message,

Considering the **threats to validity**, this approach is insufficiently rigorous to make quantitative claims about the data, but it does allow us to use the assigned codes as signposts into the data to locate example messages and to observe the absence of some codes. The method by which the sample of participants was recruited means that it consists of developers located in Sweden. The combination of these means that our analysis is focused on statements that we can be confident represent the opinion of one or more developers, and we will avoid making statements about the absence of claims. Further research would be needed to be confident that the claims generalise to other contexts.

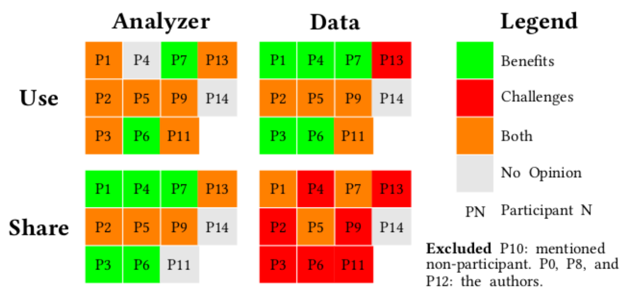

Figure 1: Summary of benefits and challenges as mentioned per situational code and participant.

4 RESULTS

The discussions from the participants show a substantial difference in the attitudes towards sharing open source code for analysers and data associated with the analysers. We have summarized the discussions in Figure 1, as a summary for situational code and participants and then how they responded with regard to mention of benefits, challenges, or both.

Using shared analyzers Participants were generally positive about using shared analyzers:

> *"pretty much everything we run here is OSS"* (P7)
>
> *"Absolutely. We use rather many open source tools and OSS in general. So I don't see that as a problem"* (P11)

but also mentioned that they consider aspects of security, licenses, maturity, and liability before they start to use open source tools:

> *"There is certainly a security aspect here ... either complex source or not that widely adopted"* (P1)
>
> *"we are using open-source tools, always there is a concern about the licensing for that open-source"* (P2)
>
> *"a popular well organized project, yeah it would make me feel more confident"* (P3)
>
> *"OSS is a little more new to them, sort of a question of liability"* (P13)

Sharing analyzers The developers were generally positive about the prospect of sharing analysers:

> *"we at C1 love open-source ... we try to open-source our internal tools and I can certainly see that we contribute images or whatever format for existing checkers, I see no problem with that"* (P1)

with one exception:

> *"currently legal question. We have not as a company come to any conclusion about contributing with code to OSS projects."* (P9)

They saw this in broad and general terms associated with professional practice, rather than in seeking specific advantages either for themselves or for the companies they worked for:

> *"For me it is modern software development, culture. Open source is a natural part of our ecosystem. And if that should flourish we need to give back too"* (P9)
>
> *"I don't know if I would write my own, I see that as there a lots of them already, but it would be great fun to contribute to something ... I would like to contribute, if I could"* (P3)

"I would contribute if it seems useful to someone else" (P4)

They were mostly concerned that the companies they worked for might impede their efforts to share in these ways, either through conservatism or bureaucracy:

"Contribution is a tough word in C2" - (P2)

The description organisations that were not willing to contribute had a pejorative tone:

"C2 is still a little in the old fashion way, why should we give away something for free ... they are a bit immature compared to C1" (P13)

"I can say something about the worries we have and why we do not always contribute. Obviously disclosing company secrets. And also some licenses ... a risk that you lose a patent if you contribute in the wrong way" (P5)

The organisations that were more structurally supportive of open source similarly saw the commercial advantage of open source sharing of code in similarly broad terms:

"We share costs, effort, ideas" (P11)

"a good idea to contribute yourself too. Then you get more to say as a company, if you are participating. Otherwise you are completely dependent on everyone else that contribute." (P9)

The advantages they saw in the construction and sharing of open source code was to influence the direction of projects and as a mechanism to recruit and retain development talent:

"The other large advantage is that OSS is an effective marketing channel towards developers. So you build a reputation by being active in the open source community ... Good developers – then it is very effective to go through open source channels" (P5)

"it is fun to work with OSS, and of course, it's branding thing" (P7)

Using shared analyzer data The notion of receiving data about analyzers to tune them was perceived as positive in that analyzer could become more useful:

"whatever data that can make the analyzer more useful ... I think that would be great" (P3)

"Data about usage can be useful ... then you can get early information about what checks do not work, and which that do work" (P6)

but there were also concerns of the potential challenges with usefulness due to data generated in one context might not apply in another:

"a false positive for me may not be that for someone else. A risk with that too" (P9)

"The skeptic inside me says 'what use should that be, that they have used the tool ...?' A completely different system than my system". (P9)

and the possibility of bad actors tampering with the data (in a setting where it is anonymous):

"if it is totally anonymous, how do you know that it is real data, so that you don't disable a check that has actually been the ... government wants everyone to disable the buffer overflow check" (P13)

Sharing analyzer data Participants generally expressed concern about sharing analyzer data, mentioning the risk of revealing information about sensitive projects within the organisation, or information about the presence of security vulnerabilities:

"Defect data we would never disclose" (P9)

"if we share data about checkers we have disabled, what does that tell other projects about our products, or hackers" (P7)

"it might give more information than you actually want from your code" (P13)

and discussed prerequisites for sharing data, such as anonymity and a sufficient level of aggregation of the data:

"I guess if it is anonymized it should be safe to share ... if it is only one company reporting, or only one user reporting for one checker, then you might be able to deduct what company it is, so I guess it is also a question of scale of the complete system" (P1)

The developers further expressed an expectation of significant organisational opposition

"Initially it might be sensitive to share information I think" (P1)

"Especially bureaucratic companies like ours, it may take time" (P2)

"I believe that they would not basically not do it" (P13)

A number of the developers further saw the question on sharing data as being decided on a case by case basis, with an inclination towards the negative:

"Impossible to give a general answer" (P11)

"to contribute it to everyone I would say no" (P9)

"depends on who is asking. If P5 asked and we had a dialogue about how to tune ... that would be possible" (P9)

5 DISCUSSION

As one developer in the focus groups put it *"Sharing is caring, but not in all cases"* (P2). When the developers we spoke to shared code, they did so with broad, unspecific aims. They ranged from a general belief that in doing so they assisted the profession, that it carried little risk to their organisation, and would help in supporting recruitment. In other words, sharing code and using others code is a part of the modern professional software development culture, and companies that didn't understand this were at best conservative and at worse regressive in nature.

Whilst this broad, non-instrumental approach to development practice benefits the open source community, it presents challenges when transferred to sharing of data. Here, there are no such norms and practices, as the possibility of using data to adjust development practices is a new phenomena. In the absence of these norms, the lack of clarity of the organisational logic in making the decision around whether to contribute or not to open source, suggests that it is not possible to use the structure to reason by analogy about sharing data. Consequently, all that is left is the sense of risk and expected institutional resistance. Figure 1 highlights the magnitude of this effect, moving from a broad optimism about sharing code to a broad concern about sharing data.

However if we accept the hypothesis that the ability to share data about development practices will have a positive beneficial effect on developer effectiveness, as the Tricorder system demonstrates, then this is a concerning finding. It means that this benefit will only be available to larger, centralised organisations which operate at sufficient scale to utilise data that they internally gather, in order to benefit their own development practice. This further will competitively disadvantage smaller, and public, development organisations who rely on open source development practices.

Given the lack of perceived benefit, the limits of analogical arguments from open source, and the perceived risks, we suggest this requires a considerable rethink. The discussion in the focus group showed a strong sense of solidarity amongst developers, we suggest that a positive route forward for building data-driven development is to explore mechanisms where this solidarity can be used to share behaviour and non-site specific learnings about code analysis, rather than a straight-forward mapping from either open source or the practices of centralised institutions.

We note that previous work focuses on data used *in* software engineering (e.g., map data [22]) and mention a need for technical solutions, licences, governance, and privacy for individuals, while in the case studied here the data is *about* software engineering, and the primary challenge discussed was that of confidentiality on an organisational level. Even though developer tools (and analyzers) often are seen as a space where it is beneficial to engage with OSS, when we discussed the notion of sharing data in the same space the main focus was on risks prohibiting sharing.

6 CONCLUSIONS

We have explored how sharing of open source and sharing of data relate to each other in the setting of data-driven development, using the specific case of tuning of static code analyzers as a driver for our exploration. We gathered data via 3 focus groups with a total of 11 participants (representing 8 companies), and then analyzed the material using codes from the literature to identify beacons in the material. Our results suggest that sharing of data is different from sharing code, to an extent where it can not ride successfully on the existing organisational OSS support. We further see a difference in sharing data *about* software development, as opposed to data used *in* software development, in that the risk to the brand increases significantly. We see this as prohibitive when trying to implement data-driven development across organisations.

Future Work. We several possible ways to continue this exploration: 1) explore mechanisms that build on the sense of a shared professional practice among developers to share behavioral data in a non-organisational specific manner, 2) explore possibilities for knowledge transfer from the security community where there is a process for sharing of vulnerabilities, and 3) study other cases in open data-driven tuning of software development tools to pinpoint how they differ from tuning of static code analyzers.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is supported by the Swedish Foundation for Strategic Research under Grant No. FFL18-0231, the Swedish Research Council under Grant No. 2019-05658, and Vinnova under Grant No. 2018-03965.

REFERENCES

[1] E. Aftandilian, R. Sauciuc, S. Priya, and S. Krishnan. 2012. Building Useful Program Analysis Tools Using an Extensible Java Compiler. In *SCAM'12*. 14–23.

[2] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. 2008. Using Static Analysis to Find Bugs. *IEEE Software* 25, 5 (2008), 22–29.

[3] C. Bird, B. Murphy, N. Nagappan, and T. Zimmermann. 2011. Empirical Software Engineering at Microsoft Research. In *CSCW'11*.

[4] A. Blackwell, L. Church, M. Jones, R. Jones, M. Mahmoudi, M. Marasoiu, S. Makins, D. Nauck, K. Prince, A. Semrov, A. Simpson, M. Spott, A. Vuylsteke, and X. Wang. 2018. Computer says 'don't know' – interacting visually with incomplete AI models. In *Workshop on Designing Technologies to Support Human Problem Solving - VL/HCC*. 5–14.

[5] R. Capilla, B. Gallina, C. Cetina, and J. Favaro. 2019. Opportunities for software reuse in an uncertain world: From past to emerging trends. *Journal of Software: Evolution and Process* 31, 8 (2019).

[6] L. Church and E. Söderberg. 2019. Probes and Sensors: The Design of Feedback Loops for Usability Improvements. In *PPIG'19*.

[7] T. Copeland. 2005. *PMD applied*. Vol. 10. Centennial Books Arexandria, Va, USA.

[8] R. Enríquez-Reyes, S. Cadena-Vela, A. Fuster-Guilló, J. N. Mazón, L. D. Ibáñez, and E. Simperl. 2021. Systematic Mapping of Open Data Studies: Classification and Trends From a Technological Perspective. *IEEE Access* 9 (2021), 12968–12988.

[9] J. Feller and B. Fitzgerald. 2002. *Understanding open source software development*. Addison-Wesley.

[10] S. Heckman and L. Williams. 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology* 53, 4 (2011), 363–387.

[11] M. Höst and A. Orucevic-Alagic. 2011. A systematic review of research on open source software in commercial software product development. *Information and Software Technology* 53, 6 (2011), 616–624.

[12] N. Imtiaz, A. Rahman, E. Farhana, and L. Williams. 2019. Challenges with Responding to Static Analysis Tool Alerts.. In *MSR'19*. 245–249.

[13] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *ICSE'13*. 672–681.

[14] J. Kontio, J. Bragge, and L. Lehtola. 2008. The Focus Group Method as an Empirical Tool in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, Sjøberg D.I.K. Shull F., Singer J. (Ed.). Springer, 93–116.

[15] J. Linåker and B. Regnell. 2020. What to share, when, and where : balancing the objectives and complexities of open source software contributions. 25, 5 (2020), 3799–3840.

[16] A. Ljungberg, D. Åkesson, E. Söderberg, J. Sten, G. Lundh, and L. Church. 2021. Case Study of Data-driven Deployment of Program Analysis on an Open Tools Stack. In *ICSE-SEIP'21*. IEEE.

[17] L. Morgan and P. Finnegan. 2007. How Perceptions of Open Source Software Influence Adoption: An Exploratory Study. In *ECIS'07*. 973–984.

[18] H. Munir, J. Linåker, K. Wnuk, P. Runeson, and B. Regnell. 2018. Open innovation using open source tools: a case study at Sony Mobile. *Empirical Software Engineering* 23 (2018), 186–223. Issue 1.

[19] H. H. Olsson and J. Bosch. 2014. From Opinions to Data-Driven Software R&D: A Multi-case Study on How to Close the 'Open Loop' Problem. In *SEAA'14*.

[20] T. Olsson, M. Hell, M. Höst, U. Franke, and M. Borg. 2019. Sharing of Vulnerability Information Among Companies – A Survey of Swedish Companies. In *SEAA'19)*. 284–291.

[21] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. R. Raje, and J. H. Hill. 2017. Identifying and Documenting False Positive Patterns Generated by Static Code Analysis Tools. In *SER&IP'17*. 55–61.

[22] P. Runeson. 2019. Open collaborative data: using OSS principles to share data in SW engineering. In *ICSE-NIER'19*. 25–28.

[23] P. Runeson and T. Olsson. 2020. Challenges and Opportunities in Open Data Collaboration – a focus group study. In *SEAA'20*. 205–212.

[24] A. Rutkowski, Y. Kadobayashi, I. Furey, D. Rajnovic, R. Martin, T. Takahashi, C. Schultz, G. Reid, G. Schudel, M. Hird, and S. Adegbite. 2010. CYBEX: The Cybersecurity Information Exchange Framework (x.1500). *SIGCOMM Comput. Commun. Rev.* 40, 5 (2010), 59–64.

[25] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *ICSE'15*. 598–608.

[26] S. Shahrivar, S. Elahi, A. Hassanzadeh, and G. Montazer. 2018. A business model for commercial open source software: A systematic literature review. *Information and Software Technology* 103 (2018), 202–214.

[27] I. Vakilinia and S. Sengupta. 2019. Fair and private rewarding in a coalitional game of cybersecurity information sharing. *IET Information Security* 13, 6 (2019), 530–540.

[28] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman. 2020. How developers engage with static analysis tools in different contexts. *Empirical Software Engineering* 25 (2020), 1419–1457. Issue 2.