# Towards improved Adoption: Effectiveness of Research Tools in the Real World

Richa Awasthy
Australian National University
Canberra, Australia
Email: richa.awasthy@anu.edu.au

Shayne Flint
Research School of Computer Science
Australian National University
Canberra, Australia
shayne.flint@anu.edu.au

Ramesh Sankaranarayana
Research School of Computer Science
Australian National University
Canberra, Australia
ramesh@cs.anu.edu.au

***Abstract*—One of the challenges in the area of software engineering research has been the low rate of adoption by industry of the tools and methods produced by university researchers. We present a model to improve the situation by providing tangible evidence that demonstrates the real-world effectiveness of such tools and methods. A survey of practising software engineers indicates that the approach in the model is valid and applicable. We apply and test the model for providing such evidence and demonstrate its effectiveness in the context of static analysis using FindBugs. This model can be used to analyse the effectiveness of academic research contributions to industry and contribute towards improving their adoption.**

## I. Introduction

The success of software engineering research in universities can be measured in terms of the industrial adoption of methods and tools developed by researchers. Current adoption rates are low [1] and this contributes to a widening gap between software engineering research and practice. Consider, for example, code inspections, which according to Fagan's law, are effective in reducing errors in software, increasing productivity and achieving project stability [2]. Static analysis tools developed by university researchers help automate the code inspection process. However, the use of such tools has not obtained widespread adoption in industry. One reason for this limited adoption is that researchers often fail to provide real-world evidence that the methods and tools they develop are of potential value to industry practitioners [1], [3].

One approach to providing such evidence is to conduct experiments that demonstrate the effectiveness of research tools in a real-world context. We apply this approach to analyse the effectiveness of a static analysis tool. In doing so, we demonstrate that such experimentation can contribute to closing the gap between research and practice.

The structure of this paper is as follows: Section II provides the background of our work leading to the proposed model; Section III presents a survey which shows that real world evidence can positively influence the decision of software engineers to use research tools; Section IV explains our experimental method which uses FindBugs [4] to analyse real world bugs in Eclipse [5] code; Section V discusses how simple experiments like ours can encourage more developers to use tools developed by researchers and thus contribute to closing the gap between research and practice. Section VI provides an overview of related research; Section VII presents conclusions and discusses future research.

## II. Background

Since the 1970's there have been ongoing efforts to increase the adoption of research outcomes outside of universities [6]. As a result of the United States Bayh Dole Act in 1980 [7], universities began to establish *Technology Transfer Offices* (TTOs) to facilitate the transfer of knowledge from universities to industry [8]. However, the effectiveness of TTOs has been questioned in recent years [9], [10] and there is a need to look beyond TTOs to improve adoption of academic research in industry.

Researchers in universities are working towards addressing significant problems. The outcome of their work can be a tool or method which may or may not achieve wide-spread industry adoption. A key factor limiting the readiness of these research outcomes for adoption in industry is a lack of tangible evidence that they would be effective in practice [1]. This suggests that demonstrating the effectiveness of a tool in practice can contribute to improved adoption.

Figure 1 depicts our model for demonstrating the real-world effectiveness of research tools and methods. The model involves 4 steps with intermediate activities. First step is to identify a problem to address. Step 2 is to develop a tool or a method to address the problem. The intermediate iterative activity involved between these 2 steps is the process of solution formulation, which involves adding new ideas to the available state of the art. These steps are followed by iterative testing for validation in Step 3. Step 3 confirms the readiness of the research outcome for adoption. An idea should be validated in a practical setting [11] to improve its adoption. Our model respects this viewpoint and emphasises the importance of tangible evidence from a practical setting in Step 4. Researchers should test their research outcomes in a scenario that involves real world users who are an important stakeholder for industry to increase the relevance of the evidence for industry. Demonstrating the effectiveness of research outcomes in real-world scenario will lead to change in industry perception and improved adoption of the research outcomes.

We test the applicability of this model in the static analysis context by identifying a static analysis tool created by university researchers and analysing its effectiveness in a real-world scenario.

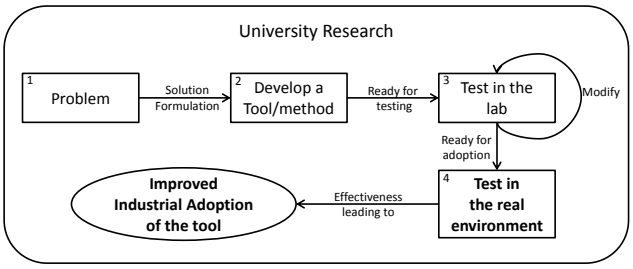


Fig. 1. Proposed model for improving adoption of university research by industry

## III. The impact of evidence from real-world scenario

In order to understand the impact of evidence from a user-scenario on real-world decisions to use a research tool, we conducted an on-line survey of software developers.

The survey uses static analysis as an example and was prepared and delivered using our university's online polling system [12]. Participants were invited by email which included a link to the on-line survey and a participant information statement. On completion of the survey, we manually analysed the results.

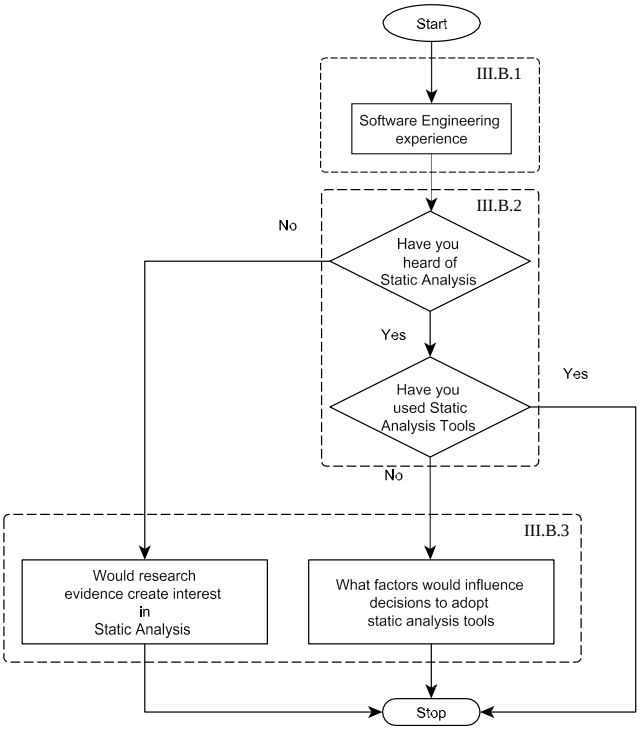


Fig. 2. Flowchart for the survey questionnaire design

### A. Participants

We invited 20 software industry practitioners and around 10 computer science researchers with industry experience in software development.

### B. Survey Questions

The survey data consisted of responses to the sequence of questions depicted in Figure 2 and described below.

*1) Software engineering experience:* We gathered information about the level of software development expertise of each participant so that we can understand any relationship between experience and use of static analysis tools.

*2) Static analysis knowledge:* We asked the following questions to determine each participant's level of understanding and use of static analysis tools.

a) '*Static analysis is the analysis of computer software to find potential bugs without actually executing the software. Have you heard of static analysis before?*'
b) '*Have you used any automated static analysis tools during software development (e.g. FindBugs, Coverity)?*'

Answers to these questions were used to determine the final question we asked, as indicated in Figure 2.

*3) The impact of the tangible evidence:* At the end of the survey each participant who has not used static analysis tools was asked a question to determine the impact that tangible evidence (that the tool can identify real bugs early in the software development life-cycle) might have on their approach to static analysis. The exact question asked depended on their answers to questions described in Section III-B2. Specifically:

1) Participants who had no knowledge of static analysis were asked '*Would our research results interest you in gaining knowledge of static analysis and adoption of automated static analysis tools?*'.
2) Participants who knew about static analysis but had not used any static analysis tools (our primary group of interest) were asked to rate the impact that the following factors would have on their decision to adopt static analysis tools. A Likert scale was used with 5 options (*No influence, May be, Likely, Highly Likely, and Definitely*).
   a) Effectiveness of tool in finding bugs
   b) Ease of use
   c) Integration of tool to development environment.
   d) License type.
   e) The availability of tangible evidence that the tool can identify real bugs early in the software development life-cycle - before they are reported by users.

### C. Analysis of Survey Results

The response rate for our survey was high with 27 responses out of 30 invitations. Responses to the survey indicate that tangible evidence of real-world effectiveness of a tool has positive impact on decisions to adopt static analysis tools.

Analysis of the survey results provides the following specific findings:

1) *Software Engineering Experience* - As expected, participants had varied level of experience in software development. However, we do not find any direct relation between the experience and usage of tools.
2) *Static Analysis Knowledge* - Survey results show that 9 participants (33%) had no prior knowledge of static analysis. It is noteworthy that while the remaining 18 participants knew about static analysis, only 4 of them had used static analysis tools.
3) *Impact of the tangible evidence* - Our survey results show that tangible evidence has a positive impact on decisions to adopt static analysis tools. Out of the 9 respondents who had no prior knowledge of static analysis, 8 said that tangible evidence would interest them in gaining knowledge of static analysis and adopting automated static analysis tools. This is a valuable information indicating that providing evidence of effectiveness could contribute to improved adoption of research tools in industry.
   Of the 14 participants who had knowledge of static analysis but who had not used any tools, 7 participants (50%) indicated that tangible evidence would be *Highly Likely* or would *Definitely* influence their decision to adopt static analysis tools (Figure 3). Another four participants indicated that such evidence would be *Likely* to influence their decision. Considering the response of three participants as *May be*, it is possible that they respond positively, which will add to the percentage of participants agreeing that tangible evidence will influence the decision.

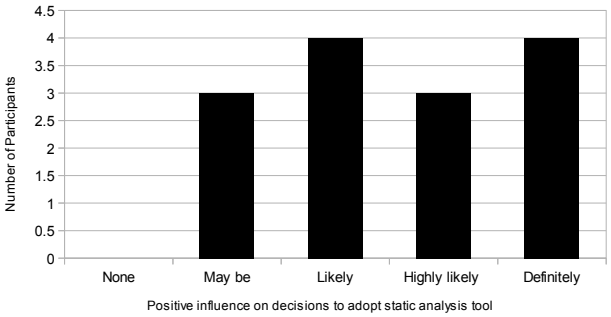


Fig. 3. The impact of tangible evidence on decisions to adopt static analysis tools

As shown in Figure 4, our results also indicate that other factors such as *Ease of use*, *IDE integration* and *License type* have a positive impact on decisions to adopt static analysis tools. It is interesting to note that under the *May be* and *Definitely* category, the top two influencing factors are *License* and *Tangible evidence*.

Our results clearly show that tangible evidence is an important factor in influencing decisions to adopt research tools in industry.

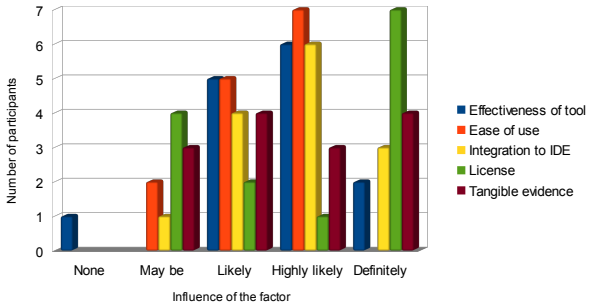


Fig. 4. Other factors influencing decisions to adopt static analysis tools

## IV. Applicability of the Model

In order to test the applicability of our proposed model, we first had to identify an appropriate tool developed by researchers and a scenario to test its effectiveness. FindBugs version 3.0 was chosen for our research as according to the tool's website, there are few organizations using FindBugs [4]. This indicates low adoption of the tool in software industry. Also, it is an open-source static analysis tool with a university's trademark. We conducted an experiment with the FindBugs static analysis tool to analyse its effectiveness in the real world. To analyse the effectiveness in real-world, we wanted to determine if FindBugs is capable of finding bugs reported by real users of Eclipse. To do so, we adopted an approach to establishing a connection between warnings generated by the FindBugs static analysis tool and field bugs reported by Eclipse users on Bugzilla [13] that includes the below steps:

1) Use FindBugs to identify potential bugs in Eclipse class files.
2) Search the Eclipse bug-tracking system Bugzilla to identify bug reports that include stack traces.
3) Match the code pointed in Java classes associated with FindBugs warnings identified in 1) with code pointed by stack trace associated with the bugs identified in 2).

### A. FindBugs

FindBugs analyses Java class files to report warnings and potential errors in the associated source code. The tool performs analysis on Java class files without needing access to the source code. It is stable, easy to use, and as mentioned in [14] has higher precision compared to other tools such as CodePro Analytix, PMD and UCDetector. It has a low rate of false positives [15], and has more than 400 types of bug classification along with categorization based on severity level. The analysis is based on bug patterns which are classified into nine categories: Bad Practice, Correctness, Experimental, Internationalization, Malicious Code Vulnerability, Multithreaded Correctness, Performance, Security, and Dodgy Code. The warnings reported by FindBugs can be further categorised within the tool as: Scariest (Ranks 1-4), Scary (Ranks 1-9) and Troubling (Ranks 1-14). The category includes all the bugs

with the ranking mentioned, for example, the 'Scary' category will list the bugs included under the 'Scariest' category, as well.

### B. Eclipse

We identified Eclipse as the object of analysis as it is a large widely used open source project with good records of user reported bugs over many years and versions of the software. For our experimentation we focused on the analysis of Eclipse version 4.4 (Luna) because it was the current version at the time of our experimentation.

### C. Identification of potential bugs

Java Jars associated with Eclipse versions 4.4 were analysed using FindBugs version 3.0. Findbugs generated a list of warnings pertaining to code considered as faulty.

### D. Search for user reported bugs

The Eclipse project uses Bugzilla to track bugs reported by users. In order to identify bugs that could be associated with FindBugs warnings, we needed to identify bug reports that included a documented stack-trace. This was achieved by performing an advanced Bugzilla search for bugs that satisfied the following criteria:

- *Version:* 4.4,
- *Classification:* Eclipse,
- *Status:* NEW, ASSIGNED, VERIFIED [1].

We then inspected the query results to identify those bug reports that included a documented stack-trace.

### E. Match FindBugs warnings with user reported Bugs

Our last step was to match warnings generated by FindBugs (Section IV-C) with user-reported bugs (Section IV-D). This was achieved using the following steps:

1) For each of the bugs identified using the procedure described in Section IV-D, we identified the Java class that was the likely source of the reported bug. This class was usually the one appearing at the top of the stack-trace. In some cases, we had to traverse through lower levels in the stack-trace to find matching classes.
2) We then searched for the above classes in the warnings generated by FindBugs (Section IV-C) and analysed the code associated with the warning. We did this by using the FindBugs class name filter feature to show warnings related to the class of interest.
3) Finding a matching line of code in the FindBugs warnings establishes a connection between the warnings generated by FindBugs and the bugs reported by users.

[1]Because our experiment looks at the ability of FindBugs to find bugs that have not been fixed, we ignore the bugs with CLOSED status. In addition, we do not consider the possibility of bugs that have been closed incorrectly.

### F. Results of the Experiment

*1) Analysis of FindBugs warnings for Eclipse version 4.4:* Our analysis of FindBugs warnings for Eclipse version 4.4 showed that static analysis of Eclipse version 4.4 generated warnings in the categories of *Correctness* (652), *Bad Practice* (547), *Experimental* (1), *Multithreaded correctness* (390), *Security* (3), and *Dodgy code* (55) under the rank range of *Troubling* (Rank 1-14), as depicted in Figure 5. We focused on the *Scariest* warnings (Rank 1-4), considering them as real problems requiring attention. There were 82 *Scariest* warnings in total. These comprised 81 warnings in the *Correctness* category and one in the *Multi-threaded correctness* category.

Additional investigation found that warnings in the *Correctness* category included a range of coding errors such as comparisons of incompatible types, null pointer dereferences and reading of uninitialized variables.

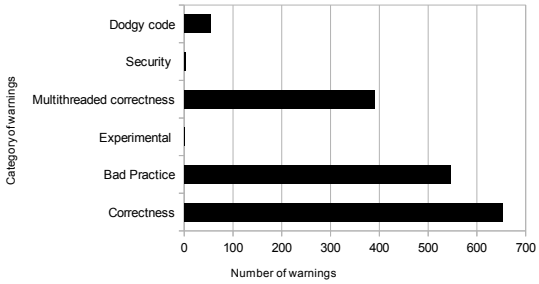


Fig. 5. Number of warnings in each category in Eclipse 4.4

*2) Connection between FindBugs warnings and user reported bugs:* Execution of the query described in Section IV-D resulted in a dataset of 2575 bugs, which included 347 enhancements. We excluded the enhancements from our analysis. Out of the remaining bugs, we have analysed 1185 bug reports so far, 90 of which included a documented stack-trace.

We used the method described in Section IV-E to compare the stack-trace in these 90 bug reports with the warnings generated by FindBugs (Section IV-F1). We found that six of the user reported bugs could be associated with FindBugs warnings as presented in Table I. The data presented in the table includes the Bugzilla Bug ID, a description of the bug, and a description of the warning generated by FindBugs.

## V. Discussion

The model proposed in this paper considers that in order to improve the adoption of university research outcomes, researchers need to demonstrate its effectiveness in a real-world scenario and think about the value of their research outcomes beyond the lab boundaries. Our main purpose in conducting the experiment described was to test the applicability of our proposed model by analysing the value of a research tool in industrial practice. Specifically, we evaluated the performance of the FindBugs static analysis tool by analysing its capability

TABLE I
USER-FILED BUGS IN ECLIPSE WITH ASSOCIATED WARNING IN FINDBUGS

| Bug Id | Problem Description | FindBugs Warning |
|---|---|---|
| 436138 | NPE when select SWT.MOZILLA style in control example | Method call passes null for nonnull parameterThis method call passes a null value for a nonnull method parameter. Either the parameter is annotated as a parameter that should always be nonnull, or analysis has shown that it will always be dereferenced. Bug kind and pattern: NP - NP_NULL_PARAM_DEREF |
| 414508 | NPE trying to launch Eclipse App | Load of known null value. The variable referenced at this point is known to be null due to an earlier check against null. Although this is valid, it might be a mistake (perhaps you intended to refer to a different variable, or perhaps the earlier check to see if the variable is null should have been a check to see if it was nonnull). Bug kind and pattern: NP - NP_LOAD_OF_KNOWN_NULL_VALUE |
| 433526 | browser.getText() is throwing exception after Internet Explorer 11 install | Possible null pointer dereference There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed; deciding that is beyond the ability of FindBugs.<br>Bug kind and pattern: NP - NP_NULL_ON_SOME_PATH |
| 459025 | Can't right-click items in manifest editor's extensions tab on OSX | Non-virtual method call passes null for nonnull parameter A possibly-null value is passed to a nonnull method parameter. Either the parameter is annotated as a parameter that should always be nonnull, or analysis has shown that it will always be dereferenced.<br>Bug kind and pattern: NP - NP_NULL_PARAM_DEREF_NONVIRTUAL |
| 427421 | NumberFormatException in periodic Workspace Save Job | Boxing/unboxing to parse a primitive A boxed primitive is created from a String, just to extract the unboxed primitive value. It is more efficient to just call the static parseXXX method.<br>Bug kind and pattern: Bx - DM_BOXED_PRIMITIVE_FOR_PARSING |
| 428890 | Search view only shows default page (NPE in PageBookView.showPageRec) | Possible null pointer dereference There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed; deciding that is beyond the ability of FindBugs.<br>Bug kind and pattern: NP - NP_NULL_ON_SOME_PATH |
| 426485 | [EditorMgmt][Split editor] Each split causes editors to be leaked | Possible null pointer dereference There is a branch of statement that, if executed, guarantees that a null value will be dereferenced, which would generate a NullPointerException when the code is executed. Of course, the problem might be that the branch or statement is infeasible and that the null pointer exception can't ever be executed; deciding that is beyond the ability of FindBugs.<br>Bug kind and pattern: NP - NP_NULL_ON_SOME_PATH |

in generating warnings relating to real-world bugs reported by users of the Eclipse IDE.

Our results indicate that FindBugs is capable of identifying bugs that will manifest themselves as bugs reported by users. Since real-world evidence would influence the decision to adopt a tool as indicated in the survey results, FindBugs needs to improve the percentage of such warnings to make the evidence convincing. Currently, FindBugs does not have the intelligence to track the bug among the list of false positives that can manifest. Our research provides improvement directions to FindBugs in identifying the important bugs from the warning base by analysing the historical data and user requirement. FindBugs results can be improved by introducing a new 'user-impact' category to classify the warnings which will potentially have an impact in the client environment and hence, need immediate attention. For this, sufficient information from industrial practice needs to be applied into testing the research tool.

The model appears simple but it is challenging for researchers to identify a scenario and approach users and/or data to demonstrate the effectiveness of their tool or methods. It might be difficult for them to find the user-filed data always. Also, once they have the data to demonstrate the effectiveness, they need a mechanism to propagate it to industry. Also, the concern about the relevance of research suggests that researchers need to think about the relevance of the problem they are trying to address. These factors indicate that universities and industry need to start collaborating at an early stage and consider co-developing whenever possible and feasible. This can pave a good start towards improving the relevance and adoption of research outcomes in general, and bridging the gap between industry and academia.

### A. Limitations and Threats to Validity

Our experiments were limited by the small number of Eclipse bugs reported with a documented stack-trace. It is important to note that the ability to analyse only 90 of the 1185 bugs considered reflects a limitation of our approach and does not reflect a limitation of FindBugs. There are some limitations to the validity of our experiments:

1) FindBugs does not always point to the exact line number referred to in the stack trace. It might be possible that the source of error could be different from the warnings provided by FindBugs.
2) While it is likely that the FindBugs warnings listed in Table I are the actual cause of the listed real-world bugs reported by Eclipse users, we cannot be certain of this.
3) As there is lack of literature detailing the use of static analysis tools like FindBugs by the Eclipse development team, a comparison study was not feasible.

4) This test highlights the relevance of static analysis tools, though the effectiveness in real-world projects, particularly large scale projects, cannot be confirmed as the sample size of our survey is small. However, by considering the sample sizes of 20 and 18 used in related studies [16], [17], we decided to proceed with our sample size of 27 participants.
5) The conclusion might not be generalised as the results are specific to the static analysis context. The proposed model needs to be validated regarding the tools in other phases of software development.

## VI. Related Work

Adoption of software engineering research outcomes in industry practice has been a concern [1], [18]. There have been ongoing efforts to improve the adoption of research outcomes since the 1970's. However, the efforts mainly focused on approaches to increase university-industry collaboration for improving adoption through technology transfer. These efforts include policy changes leading to establishment of TTOs in universities and proposing models for effective technology transfer [8], [19]. However, TTOs generally focus on building collaborative relationships between researchers and industry [20] rather than the readiness of research outcomes for adoption in industry. In this paper we demonstrated how some simple experiments can analyse the effectiveness of software engineering research tools in practice. Specifically we analysed the effectiveness of a static analysis tool.

Various experiments have been conducted to demonstrate the effectiveness of the FindBugs static analysis tool by showing that it is able to detect the specific problems it has been designed to detect [15], [21], [22]. However, our experiment has been conducted in unconstrained environment that involves real-world scenario that has impacted clients. Ruthruff et al. [23] involved developers in determining which reports were useful and which ones were not. This information was used to filter FindBugs warnings so that only those that developers found useful were reported. We retrace the user-filed bug to the warning generated by the static analysis tool. This can pave way to create an intelligent mechanism to prioritise bugs based on user-impact.

Al Bessey et al. [24] identify several factors that impacted the industry adoption of their static analysis tool Coverity [25]. They include trust between the researchers and industry users, the number of false positives, and the capability of the tool to provide warnings relating to problems which have had a significant impact on its users. Our work also confirms that tool's capability is important. It also identifies that licensing, IDE integration and ease of use are significant factors.

Johnson et al. [16] investigated the reasons behind the low adoption rate of static-analysis tools despite their proven benefits in finding bugs. Their investigations confirmed that large numbers of false positives is a major factor in low adoption rates. We note that their findings were based on the survey of developers who had all used static analysis tools. This meant that authors were not able to comment on whether low-adoption rates were due to a lack of awareness of static analysis tools among developers.

None of the work described above analyses the effectiveness of FindBugs in identifying problems that manifest themselves as real-world bugs reported by users. The experiments described in this paper analyse the connection between warnings generated by the FindBugs static analysis tool and field defects reported by Eclipse users on Bugzilla bringing in client into the perspective. The experiments test the applicability of our proposed model in the static analysis context.

## VII. Conclusion and Future Work

We have proposed a model to contribute towards improving the adoption of research tools by industry by demonstrating the effectiveness of the tool in real world scenario. We have presented a mechanism which involves a research tool as a medium of building tangible evidence. A survey of software developers supports our hypothesis that such tangible evidence of effectiveness of a tool can have a positive influence on real-world decisions to adopt static analysis tools.

Further experiment for testing the applicability of the model in the static analysis context was conducted. In this experiment, by establishing a connection between user-reported bugs and warnings generated by the FindBugs static analysis tool, we have demonstrated the ability of static analysis tools to eliminate some defects before software is deployed. However, the evidence needs to be stronger regarding the number of such connections in order to be more convincing and improving the industrial adoption of the tool.

Future research would present more detailed analysis of the complete list of the bugs found in Section IV-F2, which will provide us precise data about the effectiveness of the tool according to our approach. Our approach also presents a scenario where industry and university researchers can work together to create more useful tools. We plan to discuss these results with the FindBugs development team to explore the possibility of strengthening the evidence and devising a new classification *user-impact* to indicate the warnings that would manifest in client-environment.

Finally, we would like to adapt this approach to explore the effectiveness of research tools involved in other phases of the software development life-cycle.

## References

[1] D. Rombach and F. Seelisch, "Balancing agility and formalism in software engineering," B. Meyer, J. R. Nawrocki, and B. Walter, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Formalisms in Software Engineering: Myths Versus Empirical Facts, pp. 13–25.
[2] A. Endres and H. D. Rombach, *A handbook of software and systems engineering: empirical observations, laws and theories*. Pearson Education, 2003.
[3] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, vol. 16, no. 3, pp. 365–395, 2011.
[4] University of Maryland, "Findbugs," viewed May 2015, http://findbugs.sourceforge.net, 2015.
[5] The Eclipse Foundation, "Eclipse," viewed May 2015, http://www.eclipse.org, 2015.

[6] R. Grimaldi, M. Kenney, D. S. Siegel, and M. Wright, "30 years after bayh–dole: Reassessing academic entrepreneurship," *Research Policy*, vol. 40, no. 8, pp. 1045–1057, 2011.

[7] W. H. Schacht, "Patent ownership and federal research and development (R&D): A discussion on the Bayh-Dole act and the Stevenson-Wydler act." Congressional Research Service, Library of Congress, 2000.

[8] D. S. Siegel, D. A. Waldman, L. E. Atwater, and A. N. Link, "Commercial knowledge transfers from universities to firms: improving the effectiveness of university–industry collaboration," *The Journal of High Technology Management Research*, vol. 14, no. 1, pp. 111–133, 2003.

[9] J. G. Thursby, R. Jensen, and M. C. Thursby, "Objectives, characteristics and outcomes of university licensing: A survey of major us universities," *The Journal of Technology Transfer*, vol. 26, no. 1-2, pp. 59–72, 2001.

[10] D. S. Siegel, D. A. Waldman, L. E. Atwater, and A. N. Link, "Toward a model of the effective transfer of scientific knowledge from academicians to practitioners: qualitative evidence from the commercialization of university technologies," *Journal of Engineering and Technology Management*, vol. 21, no. 1, pp. 115–142, 2004.

[11] R. L. Glass, "The relationship between theory and practice in software engineering," *Communications of the ACM*, vol. 39, no. 11, pp. 11–13, 1996.

[12] The Australian National University, "Anu polling online," viewed July 2015, ¡https://anubis.anu.edu.au/apollo/¿, 2015.

[13] Creative Commons License, "bugzilla," viewed May 2015, https://www.bugzilla.org, 2015.

[14] A. K. Tripathi and A. Gupta, "A controlled experiment to evaluate the effectiveness and the efficiency of four static program analysis tools for java programs," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 23.

[15] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM Sigplan Notices*, vol. 39, no. 12, pp. 92–106, 2004.

[16] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 672–681.

[17] L. Layman, L. Williams, and R. S. Amant, "Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*. IEEE, 2007, pp. 176–185.

[18] S. Beecham, P. OLeary, I. Richardson, S. Baker, and J. Noll, "Who are we doing global software engineering research for?" in *2013 IEEE 8th International Conference on Global Software Engineering*. IEEE, 2013, pp. 41–50.

[19] S. L. Pfleeger, "Understanding and improving technology transfer in software engineering," *Journal of Systems and Software*, vol. 47, no. 2, pp. 111–124, 1999.

[20] D. S. Siegel, R. Veugelers, and M. Wright, "Technology transfer offices and commercialization of university intellectual property: performance and policy implications," *Oxford Review of Economic Policy*, vol. 23, no. 4, pp. 640–660, 2007.

[21] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Using findbugs on production software," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 2007, pp. 805–806.

[22] ——, "Evaluating static analysis defect warnings on production software," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2007, pp. 1–8.

[23] J. R. Ruthruff, J. Penix, J. D. Morgenthaler, S. Elbaum, and G. Rothermel, "Predicting accurate and actionable static analysis warnings: an experimental approach," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 341–350.

[24] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[25] Synposys Inc., "Coverity," viewed May 2015, http://www.coverity.com, 2015.