Towards Improving Industrial Adoption: The Choice of Programming Languages and Development Environments

Ivonne von Nostitz-Wallwitz*
METOP GmbH
Otto-von-Guericke-University
Magdeburg, Germany
ischroet@ovgu.de

Jacob Krüger
Otto-von-Guericke-University
Harz University of Applied Sciences
Magdeburg & Wernigerode, Germany
jkrueger@ovgu.de

Thomas Leich
Harz University of Applied Sciences
METOP GmbH
Wernigerode & Magdeburg, Germany
tleich@hs-harz.de

ABSTRACT

While promising software engineering approaches are proposed every day, only few are adapted by professional developers. There are many potential reasons for this, such as, problems in identifying helpful approaches, missing tools, or lacking practical relevance. With our current research, we are concerned to improve the knowledge transfer from research to practice. In this paper, we discuss the impact of development environments and programming languages on knowledge transfer – considering that many scientific approaches and tools are interesting for professional developers, but rarely adopted by them. We base our discussion mainly on our personal experiences with industry-academia collaborations. To determine whether these experiences also apply to other developers, we additionally conducted a survey with 89 participants from academia and industry. The first results of our on-going work indicate a gap between the development environments and programming languages that are supported or used by researchers and those that are applied in industry. Based on our results, we describe initial discussions that can help to improve collaborations between industry and research.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; • **Software and its engineering** → *Software notations and tools*;

ACM Reference Format:
Ivonne von Nostitz-Wallwitz, Jacob Krüger, and Thomas Leich. 2018. Towards Improving Industrial Adoption: The Choice of Programming Languages and Development Environments. In *Proceedings of SER&IP'18:IEEE/ACM 5th International Workshop on Software Engineering Research and Industrial Practice (SER&IP'18).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3195546.3195548

1 INTRODUCTION

Software engineering research focuses on many different aspects of software development that can potentially improve the daily work of professional developers, for example, embedding programming context into source code [3], bug fixing assistance [12], and automatic documentation generation [17]. Regularly, new approaches and tools are proposed that facilitate such aspects. Still, while the number of research publications and tools continuously grows [18, 29], few are adopted in practice [7, 25, 26].

One barrier to the practical applicability of research proposals is the fact that they rarely exceed a preliminary evaluation status [22]. Thus, tools and concepts that are developed to improve developers' daily work are hardly applicable in real-world settings, hampering not only knowledge transfer but also the practical evaluation of such approaches. In this context, the question arises how approaches and tools have to be designed and implemented to improve their practical applicability.

To investigate this issue, we report and discuss our personal experiences with the industrial adoption of scientific tools and barriers we faced in corresponding collaborations. In addition, we conducted a survey with 89 participants from research and industry to support our arguments. Here, we especially consider limitations in the participants' free choice regarding programming languages and environments. We remark that these are first findings of a larger body of research we are currently investigating and extending to contribute a more complete view on barriers of knowledge transfer.

Based on our first insights, we propose recommendations to help researchers during the initial phase of developing new approaches and tools to have a stronger impact on industry. For instance, our recommendations can help to decide which programming languages and tools may be more suitable. While some of the results may be known, we are not aware of empirical evidence in this regards and, thus, aim to provide a basis for further research. In particular, we hope to raise researchers' awareness on improving knowledge transfer and cooperation to industry by scoping our software development accordingly.

2 SURVEY CONDUCT

The motivation for our study originated from our personal experiences with industry-academia collaborations and, consequently, these also contribute to our work. Our experiences are based on several years of working in multiple projects and positions. For example, we work as software engineers, project managers, and consultants – supporting industrial, scientific, and collaborative projects with different domains, goals, and sizes [15, 28, 29]. In particular, we obtained our experiences from working at the METOP GmbH, a company focusing on transferring research into practice.

Besides reporting our own experiences, we base the identified barriers and recommendations on a survey. Within this survey, we

*This author previously published as Ivonne Schröter.

SER&IP'18, May 29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5744-9/18/05…$15.00
https://doi.org/10.1145/3195546.3195548

asked the participants to specify their working situation, focusing especially on limitations in the programming languages and development environments they can use. In this section, we report the *purpose*, *conduction*, and *participants* of our survey.

2.1 Purpose

In our industry-academia collaborations, we often face the situation that helpful scientific approaches exist and corresponding tools have been implemented. However, there are several known limitations that prevent the inclusion of such implementations into practice, for example, missing maintenance support, outdated versions, or poor documentation [22]. Still, even if we are able to work with a scientific tool, we often face other limitations, for example, dependencies to legacy software, support only for certain operating systems, or rarely used development environments and programming languages. Especially industry requires tools that are maintained, are adoptable to either multiple or specific environments, and are reliable. Thus, it seems to us that research and practice face a gap in the way they scope the tooling for development projects. Our investigations on this issue are guided by the following two research questions:

- RQ_1 *What are the characteristics of preferred programming languages and environments?*
- RQ_2 *Are professional developers more limited in their choice to use programming languages and environments than researchers?*

We are not aware of empirical evidence connected to such issues, despite some potential gaps being well-known. To obtain initial answers in this paper, we use a survey in which we focus on the participants' freedom when choosing programming languages and environments in their daily work. This freedom defines to which extent they can adopt scientific approaches in their daily work or may have to disregard such approaches, as these do not suit their requirements. In our survey, we compare researchers, developers, and those working in both areas, as we assume that their situations differ. Depending on these differences, researchers have to better scope their approaches to the targeted users' limitations, if they aim to improve the practical impact.

2.2 Questionnaire and Conduct

To answer our research questions, we designed a questionnaire comprising the questions we show in Table 1. We started with participants' meta data, including the number of years they are already programming and their current working position. First, we assessed the diversity among the programming languages and environments. Thus, we asked them which programming languages they are familiar with and which development tools they are working with. Afterwards, we determined the freedom of choice the participants have concerning these two aspects to identify potential limitations especially between the different groups.

The survey was available online and required approximately 10 minutes. We distributed it in organizations and universities to receive feedback from participants with different backgrounds. Additionally, we promoted it on social media platforms in groups related to software engineering to reduce a potential selection bias.

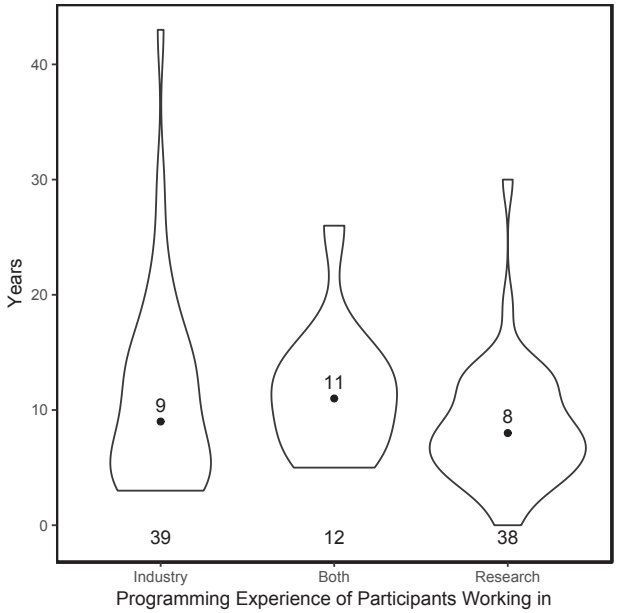


Figure 1: Number of participants and their programming experience in years separated by their stated working position.

2.3 Participants

Overall, 89 developers participated in our survey and we illustrate their current employment area and experience at this point in Figure 1. We see that – at the time of answering our survey – 39 (43.8%) of the participants are employed in industry while 38 (42.7%) are working in research, meaning an almost equal distribution of both groups. In addition, 12 (13.5%) of the participants are working in both areas at the same time.

Considering their experience in programming, we see that the median value of all three groups is around 9 years. The distributions we show in Figure 1 also indicate that we have some more experienced participants, especially in the industrial group. Surprisingly, some participants of the research group state that they have not even one full year of programming experience.

Additionally to their experience, we also find that the distribution of used programming languages and tools is similar to the TIOBE index.[1] Considering their origin, most of our participants are from Germany (59) but approximately a third (30) of them are from around the world, mainly from the United States and India (5 each). Altogether, we argue that our sample consists of participants with different experiences and backgrounds. Thus, the results we derive may not apply to specific domains or areas, but provide a good overview on software development in general.

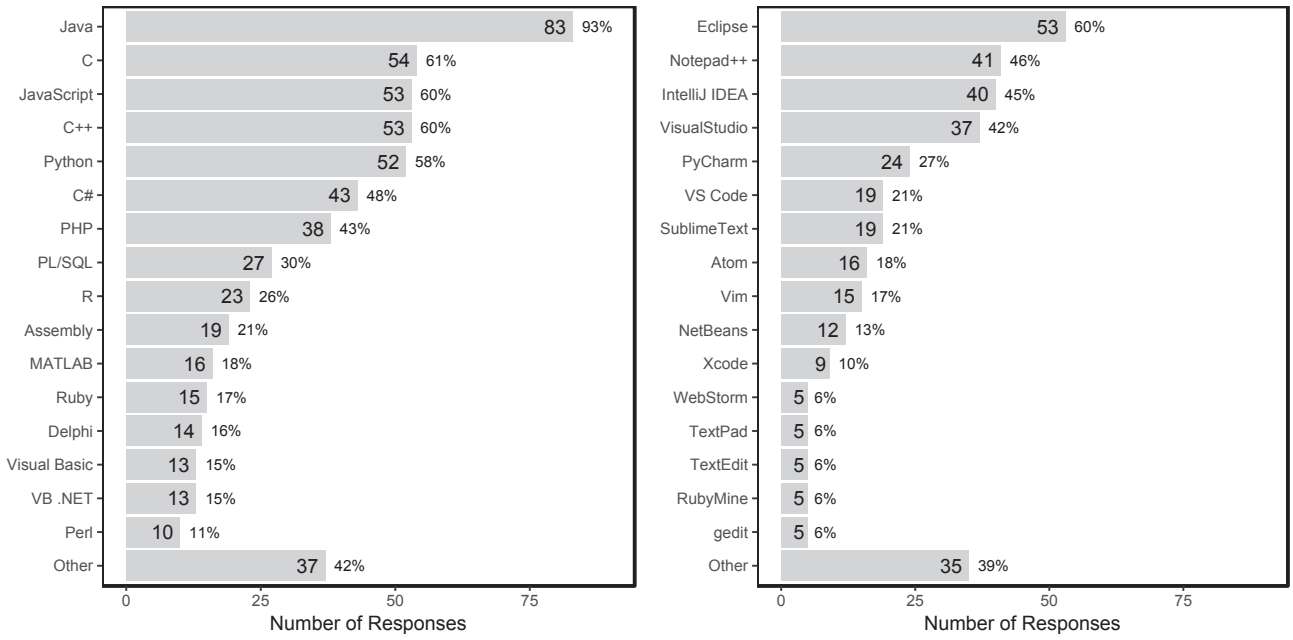
3 RESULTS AND DISCUSSION

Based on our experiences and the results of our survey, we investigate two issues – corresponding to our research questions – on knowledge transfer in this section. Each of these issues comprises a set of problems that can hamper the adoption of tools and

[1] https://www.tiobe.com/tiobe-index//

Table 1: Questions in the survey to assess the participants meta data and to address our research questions.

Nr.	Options	Answers
1	What is your current position?	Multiple choice
2	How long is your programming experience in research?	Free-text
3	How long is your programming experience in industry?	Free-text
4	Which programming languages are you familiar with?	Multiple choice
5	Which development tools are you working with?	Multiple choice
6	What is your current area of activity?	□ Research □ Industry
7	Can you freely choose the programming language you are using in your current area of activity?	○ Yes ○ No ○ Partly
8	Can you freely choose the development tools you are using in your current area of activity?	○ Yes ○ No ○ Partly



(a) Programming languages. **(b) Tools.**

Figure 2: Participants' used (a) programming languages and (b) tools.

approaches into industry. For each issue, we first motivate its importance for knowledge transfer from our experience. Afterwards, we describe the corresponding results of our survey and derive concrete barriers.

3.1 Tool and Language Preferences

In our experience, there exist several issues connected to the programming languages and development environments used to develop new approaches. For example, developers' preferences, domain restrictions, or project requirements are changing over time and for different projects. While researchers are aware of these factors, we still experienced that they sometimes rely on rare languages and tools, as these suit their purpose best. This facilitates their development process, but hampers the practical applicability of research approaches. For instance, during several projects we worked on, we found suitable research tools that would have facilitated our work, but that did not suit the industrial requirements. Consequently, we investigated in our survey, which programming languages and development environments are preferred and discuss the implications considering our first research question. We display a summary of the corresponding survey responses in Figure 2, which displays the distributions of used programming languages and tools.

Dominating Programming Languages and Environments. In our survey, we identify few dominating programming languages and environments, with `Java` and `Eclipse` on top. As we can see in Figure 2, the responses of our participants show that they prefer those languages and environments highly ranked in popularity indexes. For example, considering the September 2017 ranking of the TIOBE index, the first seven programming languages are in almost the same order, with only `JavaScript` and `C#` being slightly higher and lower rated, respectively. The results indicate that object-oriented languages are most commonly used, which is also an observable trend in language indexes. Out of the most familiar ones, only `C` does not support this programming paradigm.

Multi-Language Support. The results indicate that programming environments that can support more than one programming language are preferred. This includes complex integrated development environments (e.g., `Eclipse`, `Visual Studio`) as well as text editors with extended functionality, such as syntax highlighting (e.g., `Notepad++`). Few of the most used tools are specialized in supporting only a single programming language, despite offering plug-in capabilities (e.g., `PyCharm`).

Close Connection between Programming Languages and Environments. Unsurprisingly, there seems to be a close connection between the used programming environments and known programming languages. For example, `Eclipse` and `IntelliJ IDEA` are well-known for their `Java` support while `PyCharm` is designed exclusively for `Python`. The programming languages and their corresponding tools are rated similarly. Furthermore, tools are used less often if they support a programming language that is less known.

One Languages for One Domain. The programming languages on top differ primarily in their scope of applications: For example, `JavaScript` is used for client-side applications, `PHP` on servers, `PL/SQL` in databases, and `C` for embedded systems. Thus, it seems not surprising that multiple programming languages for the same domain are rarely on the top spots.

IDEs vs. Editors. It also seems that many developers prefer simple text editors, such as `Notepad++`. This may indicate that they prefer to scope the functionalities of their development tools to a minimum and customize them with plug-ins. For instance, `Eclipse` automatically provides syntax highlighting and method call outlines for different languages. In contrast, `Notepad++` by default only supports syntax highlighting. Unsurprisingly, the results indicate that different developers prefer language-specific environments, multi-language environments, or simple editors.

Discussion. The results substantiate our experiences that the usage of programming languages and environments depends on several factors. While not surprising, it seems obvious that well-established and domain-specific languages are preferred by most developers. Consequently, corresponding environments are also used more often. Still, competition and trends can result in changes and may make a language or tool obsolete at one point. Some examples for this are the absence of languages that do not support the object-oriented paradigm or the rise of `IntelliJ IDEA` that – from our experience – is preferred by many younger developers that dislike `Eclipse`.

For us, the most interesting insight is the preference towards multi-language tools and editors. We also experienced that such tools are more helpful for projects that require multiple languages and are faster to set-up. However, we did not expect that this tendency applies in general. While further studies seem necessary to investigate potential causations for these preferences, some reasons could be simplicity, multi-language support, domain-independence, and also the usage when teaching programming (e.g., Java).

3.2 Differing Restrictions

During our projects, we often experience that researchers prefer and have already used programming languages and tools that do not meet the industrial requirements. In particular, organizations often ask – besides fulfilling domain requirements – for industrially maintained and established tools. This way, they can ensure that the used tools have already a certain quality, have a contractor who fixes bugs, and fulfill necessary quality criteria. As a result of this policy, there apply other, stricter restrictions in the programming languages and environments used in industry compared to research. To investigate this issue and answer our second research questions, we asked our participants if they can freely pick these parts of their daily work. We display the corresponding results in Figure 3.

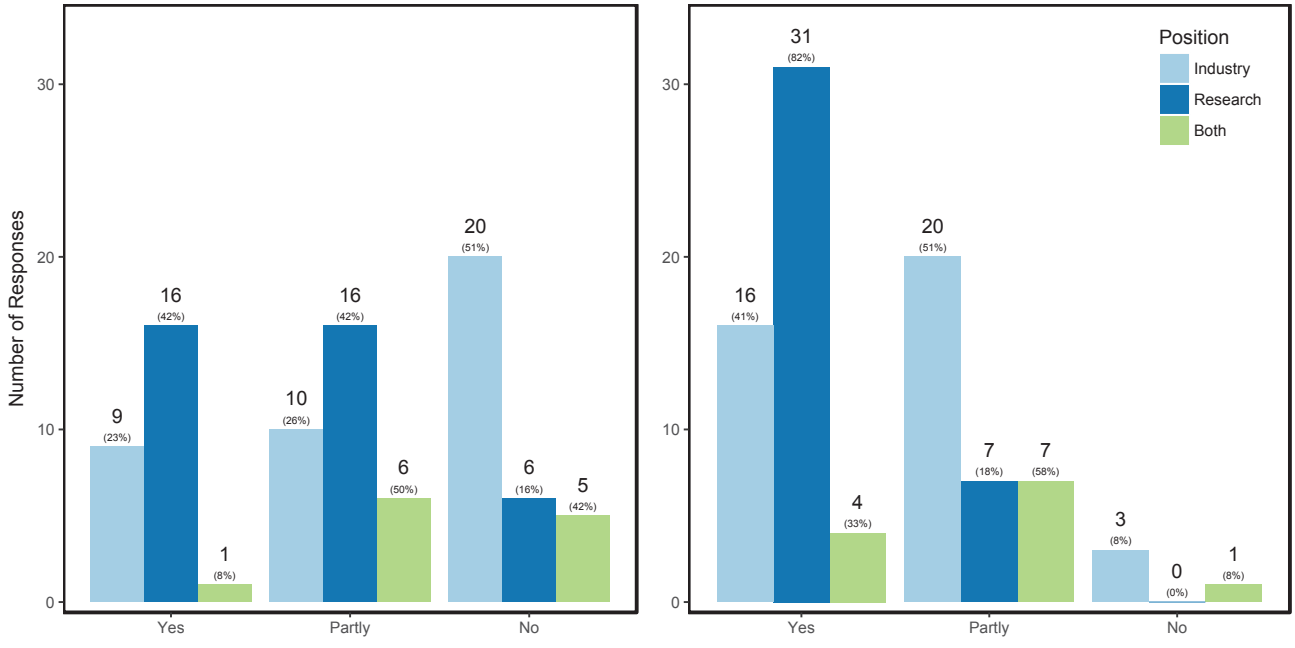
Language Restrictions. As we can see in Figure 3, developers working in industry are mostly not or only partly allowed to freely choose their programming language, with 20 (51%) and 10 (26%) responses, respectively. In contrast, developers in research face fewer limitations: 16 (42%) can freely and 16 (42%) can partly choose the used programming language. While the group of participants working in both areas is rather small, their responses align better to these from industry.

Programming Tool Restrictions. For the used tools, the outcome of our survey varies heavily. Almost none of our participants is always forced to use a specific tool set, with only 3 (8%) and 1 (8%) of the participants employed in industry or both areas stating no free choice at all, respectively. In the industry group, 16 (41%) can freely choose their tools and 20 (51%) are only partly limited. For the research area, even up to 31 (82%) can completely freely select the used tools. Again, the responses of the mixed group align better to these of the industrial developers.

Discussion. Industrial developers facing language restrictions seems not surprising. They may work for a longer time on existing systems and applications, focus on a specific (language-dependent) domain, and have to use established languages to increase availability [19]. For example, `Javascript` and `PHP` are used more frequently for web development while `C` is more suitable for embedded systems. As a result, they have to rely on specific programming languages and cannot freely switch between them. Still, such restrictions potentially depend on project specifics and a developers' position, wherefore they do not exist in every case. Thus, industrial developers can partly select the programming languages they use.

In contrast to industrial developers, researchers are often working on smaller projects matching their current problem. Approaches are often developed from scratch and solely for evaluation purpose [2, 16]. Thus, researchers face restrictions mainly when they reuse another tool or work in a specific domain. However, this seems not to apply for researchers working in industry-academia collaborations, as their responses align better to those of the industrial developers. This also suites our experience that some researchers who never worked with industry do not care about the practical applicability of their research. While practicability may not be the main purpose of researchers, it is still beneficial and indicates a missing awareness of real-world scenarios.

Considering tools, the results indicate that most developers in both areas can choose those more freely. This can be explained, as many tools provide similar to identical functionalities, are freely available, and are established. Thus, restrictions may derive from project recommendations, specific extensions being only available



(a) Programming languages. **(b) Tools.**

Figure 3: Participants' freedom to choose (a) the programming languages and (b) the tools they are working with.

in some tools, and customer requirements – but seem to mainly focus on a working product. As a result, industrial and collaborating developers face partial restrictions more often.

4 IMPLICATIONS

Considering the previously described results, we derive preliminary answers to our research questions in this section. Afterwards, we derive recommendations that can help to improve the practical adoption of research approaches. While these recommendations may be intuitive for researchers often collaborating with industry, we experienced that others are completely unaware of them. Again, we emphasize that these are initial results of on-going work and will be refined in more extensive investigations.

4.1 Research Questions

Our first research question is concerned with the characteristics of the preferred programming languages and environments. While this is a preliminary assessment, we still find that most preferred programming languages are domain specific and support object-orientation. Considering the tools, we find that they often support multiple languages, can be extended, or are simplified text editors. An unsurprising outcome of these observations is that languages and environments seem to align.

The second research question addresses the developers' free choice to select the programming languages and tools they use. From our point of view, the results are less surprising and indicate that industrial developers often need to use specific ones. In the research community however, there seems to be a difference between those that collaborate with industry and those that do not. While this is reasonable, we find it interesting to further investigate how this free choice and experiences with collaborations may influence later projects of these scientists: Do researchers focus more on practical applicability after they worked in industrial collaborations or does it have no impact?

4.2 Recommendations

Focus on Popular Programming Languages and Tools. If there is no concrete project to which an approach has to be scoped, we recommend to rely on more popular programming languages and environments, considering, for example, the TIOBE index. As developers cannot freely choose the programming languages they are working with, researchers should focus on the ones that are frequently used in their specific domain or are generally used. This does not only support knowledge transfer, but also facilitates extending and maintaining tools. In contrast, if the approach is developed to support researchers, language restrictions are less important – but we still recommend to rely on well-known languages and tools. Otherwise, our experiences are that others may have problems and may not reuse an approach.

Prefer Pluggable over Stand-Alone Tools. Researching on programming tools can improve practical applicability, as developers can more freely select which tools they use. Nevertheless, most of the time developers use only their preferred programming environment. Thus, the development of plug-ins for frequently used programming environments or importable components (e.g., as jar files) should be preferred over the development of completely new tools. However, it may be problematic to identify which environment should be supported. For some languages, such as `Java`, several different environments and editors, for example, `Eclipse`, `IntelliJ IDEA`, and `Notepad++`, exist. The question arises, whether

a single (i.e., plug-in) or multiple (i.e., a component) of these tools should be supported to be applicable by most developers. This also includes considering the purpose of the approach: Shall it only be used by developers or integrated into other tools?

Consider Tools' Functionalities. Considering evaluations of new approaches and tools, we remark that programming environments often contain partly complex functionalities and can often be freely chosen. This must be taken into account while scoping approaches and their evaluations. For example, if different programming languages are used, such as, in static and dynamic type systems, it is not always possible to use the same programming environment with the same functionalities. This can influence, for instance, performance measurements while solving programming or comprehension tasks. Still, if in such cases only text editors with limited functionality are used, the results of an evaluation may be not valid in practice. Tools with richer functionality may decrease possible disadvantages of one approach but are commonly used in practice and developers are familiar with them.

5 THREATS TO VALIDITY

In this section, we discuss potential threats to the validity of our work. Here, we rely on common distinctions [6, 24, 34] for empirical studies in software engineering. Consequently, we separate *construct*, *internal*, and *external* validity as well as *reliability*.

5.1 Construct Validity

The construct validity of our study is potentially threatened by the questions and terms we used. Our participants may have misunderstood something, for example, due to language barriers. Still, the terminology should be easy to understand for experienced developers and in an according question, none of the participants reported problems regarding the questions. Overall, we argue that there is no threat to the construction validity of our questionnaire.

5.2 Internal Validity

As for the question regarding knowledge and usage of programming languages and tools, we provided a predefined set of selectable options. Further languages and tools could be added by the participants. Thus, the answers may be biased, as participants may not remember all programming languages and tools they are familiar with or they are using. If they were not included in our options, the participants may have missed these. While this threat can hardly be completely overcome, we derived the options for languages and tools from popularity ratings, such as the TIOBE index. For this reason, we argue that we cover the most important of them, which reduces the potential bias.

5.3 External Validity

There is a potential selection bias, as we received more answers from developers we could contact directly. As a result, most of the participants are from Germany. Still, we received several additional answers from around the world. Thus, we argue that the results are valid and the identified issues are important to address.

Another threat to the external validity are the questions we used. For example, we did not precisely ask for the reasons why participants cannot freely choose the used tools and languages. While there may be additional reasons for this, we argued about the differences based on the results and our experiences. Furthermore, the different reasons for limitations in using languages and tools do not directly affect the derived recommendations. Thus, we argue that our recommendations are valid and reasonable, but can be refined in further studies that include additional questions.

5.4 Reliability

Overall, there are potential threats to the reliability of our work: The results and outcome of the survey may differ depending on the considered participants. Still, any researcher can replicate this study and will face the same threats. Consolidating and validating our findings with replication studies is important and can help to reduce this threat to reliability.

As we also mentioned before, our experiences do not represent those of all researchers. Consequently, in several collaborations – especially with researchers commonly performing collaborative projects – such experiences may differ. However, we argue that some researchers and, in particular, those that cooperate with industry for the first time, are less aware of the described issues. We also addressed this threat by conducting a survey that illustrates some of our experiences being apparently common.

6 RELATED WORK

There is considerable work related to knowledge and technology transfer issues from research into industry [9–11, 25, 35]. Different reasons have been determined, for example, lacks of credibility, missing awareness and discrepancies between research topics and those with practical relevance, as well as difficulties to survey, understand, and use research [11, 25, 26, 29, 31, 32]. These works are related to ours as they also investigate potential barriers of knowledge transfer. However, we complement such works by focusing specifically on discrepancies of languages and tools used in research and industry. We are not aware of studies addressing these issues.

To overcome the aforementioned issues, technology transfer models have been proposed. These, for instance, demand close cooperations between research and industry [10], because research results cannot be transferred directly. In addition, scientific publications are not designed to be a communication channel between research and industry [27, 33]. Technology and knowledge transfer research is related to our work, as we address the same issues. Still, we focus on another point of view: How must approaches and tools be implemented to improve their applicability in the daily work of industrial developers? Thus, existing models can be extended and refined to consider the results of our work.

There are several works concerning the interests of professional developers regarding, for example, programming tools, concepts, and algorithms [1, 4, 5, 14]. Other works analyze topics researchers focus on, including the identification of gaps between research and industry [5, 8, 11, 30]. The results are not only interesting for developers aiming to use the investigated techniques, but also for researchers that aim to propose new approaches and tools that can be used in industry. As we investigate the diversity among used programming languages and tools as well as the freedom of choice in this regard, we complement such works. In particular, works exploring why specific approaches are not adopted in practice

are not only related to ours but the results may be compared to consolidate the findings.

Other works investigate how developers use development tools or perform specific tasks [13, 20, 21, 23]. Such works improve the understanding of how tools can be extended to facilitate the practical applicability of new approaches. Thus, we complement these investigations by identifying which languages and tools are popular, and to which extent developers can freely choose them. Combining the results of both areas supports researchers while scoping concrete tools extensions.

7 CONCLUSION

The number of research projects focusing on approaches and tools to improve programming is continuously growing. This growth can hamper the decision-making process of professional developers to adopt research approaches. Still, this is not the only reason for limited knowledge and technology transfer between research and industry. In our on-going work, we addressed this issue by investigating aspects that influence the probability that a new approach or tool can be used in practice. To this end, we relied on our experiences with collaborative projects and conducted an empirical study to support these. Overall, we identify the diversity among used programming languages and environments, as well as the freedom of choice in this regard. In particular, we separated between professional developers working in research, industry, and both to identify differences between these groups.

We identified a set of dominating programming languages and environments that most developers use or are familiar with. Less surprising, we find a close connection between languages and environments. Furthermore, professional developers are more likely to be limited in their freedom to choose programming languages. In contrast, tools are much rarer fixed among all groups.

Overall, we derive the following recommendations to improve the potential of approaches and tools being used in industry:

- Researchers should focus on domain specific and popular programming languages and tools.
- Developing pluggable tools and extensions should be preferred over developing new stand-alone tools.
- As complex development environments seem to be preferred by practitioners, researchers have to consider their functionalities to evaluate practical applicability.

In future works, we aim to refine our survey by consolidating existing works on barriers that hamper knowledge transfer towards practice. Furthermore, we will refine and extend our survey to identify concrete reasons why approaches and tools are not adopted. Additional future work includes the definition of guidelines that summarize best practices to improve knowledge transfer.

ACKNOWLEDGMENTS

This research is supported by DFG grant LE 3382/2-1 and Volkswagen Financial Services AG.

REFERENCES

[1] Chris Barry and Michael Lang. 2001. A Survey of Multimedia and Web Development Techniques and Methodology Usage. *IEEE MultiMedia* 8, 2 (2001), 52–60.

[2] Andrew Bragdon, Robert C. Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola Jr. 2010. Code Bubbles: A Working Set-Based Interface for Code Understanding and Maintenance. In *International Conference on Human Factors in Computing Systems (CHI)*. ACM, 2503–2512.

[3] Alexander Breckel and Matthias Tichy. 2016. Inline: Now You're Coding with Portals. In *International Conference on Program Comprehension (ICPC)*. IEEE, 1–3.

[4] John Businge, Alexander Serebrenik, and Mark van den Brand. 2010. An Empirical Study of the Evolution of Eclipse Third-Party Plug-Ins. In *Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (EVOL/IWPSE)*. ACM, 63–72.

[5] Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. 2011. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. In *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 337–346.

[6] Thomas D Cook and Donald Thomas Campbell. 1979. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin.

[7] James R. Cordy. 2003. Comprehending Reality-Practical Barriers to Industrial Adoption of Software Maintenance Automation. In *International Workshop on Program Comprehension (IWPC)*. IEEE, 196–205.

[8] Dan Craigen, Susan Gerhart, and Ted Ralston. 1993. An International Survey of Industrial Applications of Formal Methods. In *Annual Z User Meeting*. Springer, 1–5.

[9] Philipp Diebold and Antonio Vetro. 2014. Bridging the Gap: SE Technology Transfer into Practice: Study Design and Preliminary Results. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 52:1–52:4.

[10] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. 2006. A Model for Technology Transfer in Practice. *IEEE Software* 23, 6 (2006), 88–95.

[11] Vladimir Ivanov, Alan Rogers, Giancarlo Succi, Jooyong Yi, and Vasilii Zorin. 2017. What Do Software Engineers Care About? Gaps Between Research and Practice. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 890–895.

[12] Dennis Jeffrey, Min Feng, Neelam Gupta, and Rajiv Gupta. 2009. BugFix: A Learning-Based Tool to Assist Developers in Fixing Bugs. In *International Conference on Program Comprehension (ICPC)*. IEEE, 70–79.

[13] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32, 12 (2006), 971–987.

[14] Jacob Krüger, Niklas Corr, Ivonne Schröter, and Thomas Leich. 2017. Digging into the Eclipse Marketplace. In *International Conference on Open Source Systems: Towards Robust Practices (OSS)*. Springer, 60–65.

[15] Jacob Krüger, Stephan Dassow, Karl-Albert Bebber, and Thomas Leich. 2017. Daedalus or Icarus? Experiences on Follow-the-Sun. In *International Conference on Global Software Engineering (ICGSE)*. IEEE, 31–35.

[16] Aniket Kulkarni and Ravindra Metta. 2014. A Code Obfuscation Framework Using Code Clones. In *International Conference on Program Comprehension (ICPC)*. ACM, 295–299.

[17] Paul W. McBurney and Collin McMillan. 2014. Automatic Documentation Generation via Source Code Summarization of Method Context. In *International Conference on Program Comprehension (ICPC)*. ACM, 279–290.

[18] Matthew R McGrail, Claire M Rickard, and Rebecca Jones. 2006. Publish or Perish: A Systematic Review of Interventions to Increase Academic Publication Rates. *Higher Education Research & Development* 25, 1 (2006), 19–35.

[19] Leo A. Meyerovich and Ariel S. Rabkin. 2013. Empirical Analysis of Programming Language Adoption. In *International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*. ACM, 1–18.

[20] Gail C. Murphy, Mik Kersten, and Leah Findlater. 2006. How are Java Software Developers Using the Elipse IDE? *IEEE Software* 23, 4 (2006), 76–83.

[21] Emerson Murphy-Hill and Andrew P Black. 2008. Refactoring Tools: Fitness for Purpose. *IEEE Software* 25, 5 (2008).

[22] Donald A. Norman. 2010. The Research-Practice Gap: The Need for Translational Developers. *Interactions* 17, 4 (2010), 9–12.

[23] Semih Okur and Danny Dig. 2012. How Do Developers Use Parallel Libraries?. In *International Symposium on the Foundations of Software Engineering (FSE)*. ACM, 54:1–54:11.

[24] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. 2000. Empirical Studies of Software Engineering: A Roadmap. In *International Conference on Software Engineering (ICSE)*. ACM, 345–355.

[25] Shari Lawrence Pfleeger. 1999. Understanding and Improving Technology Transfer in Software Engineering. *Journal of Systems and Software* 47, 2-3 (1999), 111–124.

[26] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How Do Professional Developers Comprehend Software?. In *International Conference on Software Engineering (ICSE)*. IEEE, 255–265.

[27] Per Runeson. 2012. It Takes Two to Tango - An Experience Report on Industry - Academia Collaboration. In *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 872–877.
[28] Martin Schäler, Thomas Leich, Marko Rosenmüller, and Gunter Saake. 2012. Building Information System Variants with Tailored Database Schemas Using Features. In *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 597–612.
[29] Ivonne Schröter, Jacob Krüger, Philipp Ludwig, Marcus Thiel, Andreas Nürnberger, and Thomas Leich. 2017. Identifying Innovative Documents: Quo Vadis?. In *International Conference on Enterprise Information Systems (ICEIS)*, Vol. 1. ScitePress, 653–658.
[30] Ivonne Schröter, Jacob Krüger, Janet Siegmund, and Thomas Leich. 2017. Comprehending Studies on Program Comprehension. In *International Conference on Program Comprehension (ICPC)*. IEEE, 308–311.
[31] Mary Shaw. 2002. What Makes Good Research in Software Engineering? *International Journal on Software Tools for Technology Transfer* 4, 1 (2002), 1–7.
[32] Margaret-Anne Storey. 2006. Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future. *Software Quality Journal* 14, 3 (2006), 187–208.
[33] Claes Wohlin. 2013. Empirical Software Engineering Research with Industry: Top 10 Challenges. In *International Workshop on Conducting Empirical Studies in Industry (CESI)*. IEEE, 43–46.
[34] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer.
[35] Marvin V. Zelkowitz, Dolores R. Wallace, and D. Binkley. 1998. Culture Conflicts in Software Engineering Technology Transfer. In *NASA Goddard Software Engineering Workshop*. 1–17.