Software Engineering Research and Industry: A Symbiotic Relationship to Foster Impact

Victor Basili[1], Lionel Briand[2], Domenico Bianculli[2], Shiva Nejati[2], Fabrizio Pastore[2], Mehrdad Sabetzadeh[2]

1 University of Maryland, College Park, USA

2 SnT Centre, University of Luxembourg, Luxembourg

Software engineering is not only an increasingly challenging endeavor that goes beyond the intellectual capabilities of any single individual engineer, but is also an intensely human one. Tools and methods to develop software are employed by engineers of varied backgrounds within a large variety of organizations and application domains. As a result, the variation in challenges and practices in system requirements, architecture, and quality assurance is staggering. Human, domain and organizational factors define the context within which software engineering methodologies and technologies are to be applied and therefore the context that research needs to account for, if it is to be impactful. This paper provides an assessment of the current challenges faced by software engineering research in achieving its potential, a description of the root causes of such challenges, and a proposal for the field to move forward and become more impactful through collaborative research and innovation between public research and industry.

The relevance of academic research

In a previous column article published in IEEE Software[5], it was argued that there is a large disconnect between research and practice, and that as a result, software engineering research has had less impact than it could have. Though there has been a number of occurrences where software engineering research has had high impact on industrial practice (e.g., the "design-by-contract"[8] programming methodology, the REST[6] architectural style, the Microsoft SLAM/SDV project[2] for the automated verification of device drivers), its full impact potential has not materialized yet. One of the most prominent reasons, as we argue, is that most of the research is insufficiently grounded in real development contexts. This often results in problems that do not match real needs, and solutions that are not applicable or scalable, even with additional engineering effort. We believe that this is a missed opportunity for our field to become more relevant, better funded, and more attractive to young researchers.

Extending the above column article, we argue that for software engineering research to increase its impact and steer our community toward a more successful future, it must evolve. Specifically, we see the need to foster *context-driven research*. By that, we mean research focused on problems driven by concrete needs in specific domains and development projects. Currently, only a small proportion of the publications at top venues stem from such research.

The importance of context in software engineering research

The main motivation for the proposed evolution is that software engineering solutions'

applicability and scalability depend largely on contextual factors, whether human (such as engineers' background), organizational (such as cost and time constraints), or domain-related (such as the level of criticality and compliance with standards). For example, testing and verification techniques typically take a specific set of inputs—such as a model or source code—and make assumptions about those inputs' form and content—such as a model's level of precision or the constructs in source code. Whether such assumptions hold often depends on contextual factors. These techniques' outputs usually are assumed to be useful to support a given development task; whether this is true also depends largely on contextual factors. For example, many software engineering research papers represent cyber-physical systems (CPSs) using purely Boolean abstractions, excluding the notion of time and numerical data communicated between software and physical devices. This view, while possibly valid for a certain class of CPSs and for some specific analytical purposes, is simplistic in many other cases. This leads to overly coarse models that enable only shallow analysis of CPSs. Such unwarranted simplifications and generalizations are often due to community biases that assume any computation is rooted in logic. Our experience is that context-driven research can help correct such biases and have more impact on software development.

Research without context tends to aim at creating clean solutions, by simplifying assumptions to work out the fundamentals, which can then be tailored to more realistic contexts. However, solutions developed that way are rarely adaptable to any particular context. Usually, it is infeasible to modify them to relax assumptions about inputs, scale, people's background and capabilities, or any other relevant aspect. Too often the practitioners see research as a risk they are unwilling to take or a problem in the adoption to their context.

The focus here is on doing research that can be demonstrated to be effective in a particular context, based on realistic assumptions because it is grounded in reality. For example, observation of current engineering practices in the CPS domain reveals that most analyses rely on continuous-time-signal representations of CPS behavior[7]. This triggers the need for challenging research to devise proper modeling abstractions and analysis techniques for CPSs that account for their continuous and physics-based aspects.

Nevertheless, someone might argue that context-driven research leads to results that don't grasp the big picture and to one-off solutions that are of limited value outside their immediate context. While context-driven research provides no general recipe to tackle either of the above problems, based on many years' experience in research and engineering practice and dozens of collaborative industrial projects, our position is that a sustained, long-term progression of context-driven research provides a natural antidote to the limitations posed by this type of research. In particular, we observe that although contextual factors strongly drive the applicability of techniques and methodologies, software development organizations tend to be representative of certain application domains and develop types of systems that are also common in other organizations. For example, the development of controllers in embedded systems tends to present similar challenges and relies on similar technologies across companies and domains. This implies that context-driven research results tend to be relevant outside the immediate context in which they were obtained, usually in some type of industry or domain. Consequently, with many contexts seen, patterns begin to emerge, leaving the researchers well-positioned to tell apart what aspects of a problem or a solution generalize and what aspects don't.

Simply put, we believe that in such a practical field as software engineering, the big picture and widely-applicable solutions are more likely to be borne out of bottom-up research and a succession of case studies than top-down research, where critical contextual details with a make-or-break

impact on the applicability of the solutions may be abstracted away in the name of generalizability. For example, regarding quality assurance for natural-language requirements, numerous assumptions can directly affect a technique's applicability, such as the availability of a domain glossary, the use of strict templates, or a certain amount of background knowledge[1].

An additional benefit of context-driven research is that immersion in a domain and context helps identify problems which might be overlooked but which are nevertheless important. For example, while working on the verification of systems with strong legal and compliance requirements, e.g., a tax system, we observed a serious communication gap between software engineers and legal experts. The former usually have a superficial understanding of the law; the latter can't analyze and understand source code and complex formal notations. So, it is important in such contexts to devise a domain-specific modeling methodology that both types of stakeholders can understand.

Collaborative research and empirically-driven improvement in the software industry

Collaborative research between industry and academia is a requirement for context-driven research with significant impact. Research in software engineering, similar to other scientific disciplines, e.g., physics, medicine, manufacturing, requires the cycle of model building, experimentation and learning. The study of software engineering is a laboratory science where the researcher's role is to understand the nature of the processes, products, and their relationship in the context of the system and organization. The organization's role is to build the best systems they can, using the knowledge available. More than other disciplines, these roles are symbiotic. The researcher needs laboratories to observe and manipulate the variables; these laboratories exist only where software is being built. The organization needs to better understand how to build the best possible systems and the research can provide the novel solutions to help. In other words, both research and practice can benefit from collaborative research, the former in terms of enhancing our general understanding and body of knowledge and the latter as a way to drive innovation within software development organizations. This section addresses a specific strategy, which relies on empirical studies and analysis, to combine collaborative research and quality improvement in organizations.

To achieve synergy in such collaborations, researchers must work on high-priority, well-defined industrial problems and relate those problems to the general, scientific body of knowledge. The first challenge researchers face -- which is usually much more difficult than one could a priori expect -- is to precisely identify and prioritize problems in the collaborating organization. Once a clear problem is targeted, the gap between the state-of-the-art and the requirements for its solution must be assessed. In developing a solution it is important to clearly define working assumptions in order to achieve applicability and scalability in context. Once tentative solutions are developed, they must be evaluated in a realistic fashion, reusing artifacts from past development projects or on pilot projects. In the latter case, it is important for research activities not to interfere with the project outcomes and avoid being on the critical path. Usually, a first evaluation is performed in an artificial setting, reusing past representative artifacts, in order to get an initial assessment of a technique or methodology that is an essential component of the solution. If the results turn out to be promising, then a more realistic and expensive evaluation ensues in the context of pilot projects.

Empirical software engineering is an active field of research[4] and an indispensable element of context-driven research. There are many different types of empirical methods we can rely on, ranging from experiments to case studies and surveys, which provide complementary advantages and drawbacks fitting various objectives and contexts. Their role is to enable the rigorous and realistic evaluation of research solutions and the identification of their limitations as a function of

contextual factors. In industrial contexts, data collection and empirical analysis can provide a continuous and objective mechanism for identifying pain points, ineffective practices and assessing the impact of technology or process changes on development quality, timeliness, and productivity. In other words, empirical studies are not only at the heart of software engineering research but are also a mechanism for organizations to achieve continuous quality improvement. There is therefore an opportunity for establishing synergistic relationships between academic research and industry from which both can benefit. One important question, as we argue below, is how to organize and implement such a collaboration to ensure mutual benefits.

If we are to do context-driven research and quality improvement, we should learn from development projects and integrate what we have learned to develop richer solutions that take into account relevant context variables. One approach to achieve this was the Software Engineering Laboratory (SEL) model[3], which was implemented over 25 years in the Flight Dynamics Division at the NASA Goddard Space Flight Center (GSFC) as a consortium also involving the University of Maryland and Computer Sciences Corporation. In the SEL model, the organization is divided into two parts, the Project Organization (PO) in charge of software development, and the Experience Factory (EF) in charge of collecting and analyzing data from projects, packaging results for reuse, deriving recommendations for future projects, and in general, providing access to an experience base (EB) for enabling organizational learning and improvement. Practitioners at all levels thus receive guidance and training in selecting strategies to apply based upon what has worked for their particular situation and context. They are also provided with feedback on their work to be able to make changes in real time. Researchers both inside and outside their organization work on problems to improve their practices. As an example, over a period of nine years at the NASA GSFC and based upon the analysis of software development quality and cost data, the organization was able to successively improve the defect development rates over two four-year periods, first by 75% and then by 37%, and reduce the cost of developing those systems, first by 55% and then 42%.

The EF encompasses context-driven research and builds the EB for the organization which consists of models that have been applied and evaluated. By "*models*" we refer to abstractions such as lessons learned, heuristics, patterns, procedures, quantitative and qualitative prediction models, conceptual models, and decision support frameworks with its influencing variables and recommended context for use. The EB plays the role of a Decision Support System for the PO. The EF starts by offering the PO its potential solutions based upon what has been learned to date and tracks the application process, offering improvements whenever possible.

For a given project, the PO identifies the problems to be addressed and, based upon the context variables characterizing the project, selects a solution recommended by the EF and tailors it, to the degree possible, for the problem at hand. If there are no relevant solutions, the EF selects and tailors a solution from other sources, e.g., the research literature or commercial market, or develops it from scratch. When applying the proposed solution, data is collected and analyzed to evaluate the effectiveness of the solution. A measurement program is in place to ensure the consistent collection of data across projects, in a way that is tailored to the goals of the organization[3]. The EF analyzes such data in real time and can provide feedback to the PO, who uses that feedback to continue to evolve and improve their solution. After the project is completed, a post-mortem analysis is carried out with the help of the EF and their solution is then made available to the EB along with analysis results and the context variables discovered as important.

Another way to think of these activities is to see them as a context-based research and innovation engine that allows us to define and improve methods and techniques by letting us:

1. Formulate hypotheses regarding potential software development solutions, e.g., a test strategy being better than another one.

2. Collect data (e.g., using the GQM methodology[3]) from empirical studies, e.g., case studies where fault detection rates and test effort are collected.

3. Test our hypotheses, e.g., whether the new test strategy has more fault-revealing power.

4. Develop models based on data analysis and empirical results, e.g., under certain assumptions, such as the availability of source code and a low execution cost, the new test strategy can be expected to provide a certain degree of benefits.

5. Package and integrate those models into the experience base, so they can easily found and reused by the PO and evolved.

Because context variables are many and they have such a significant impact on the effectiveness of solutions, building a software engineering body of knowledge, even when focused on a narrow problem area, is too big of a task to be performed by one organization and research team. It requires multidisciplinary and collaborating teams who can provide various forms of complementary expertises and work across organizations differing with respect to multiple context factors. One important objective is to enable the replication of empirical studies[4], which is critical to make quick progress possible and obtain results with stronger validity. The ISERN network (http://isern.iese.de/Portal/) was established to promote such collaborations.

Limitations

Context-driven research is not without challenges. There must be a mutual understanding about the needs of both parties. The typical pace of research is very different from what can be expected in industrial projects. The industrial partner(s) must be capable of receiving research results and package them into usable solutions. Results cannot readily and widely generalize and must be interpreted in context. The EF model also comes with challenges. It takes time to build the organizational structure and create the required mindset. It needs resources up front and a champion in the organization. It requires close collaboration between researchers and developers. From a publication point of view, there can be problems with proprietary information, that need to be carefully handled.

Future outlook

Over the last 50 years, software engineering has evolved from being a branch of computer science to an engineering field in its own right. Related to this is the increasing realization among researchers that software engineering is ubiquitous across many application domains and needs to account for their characteristics to be relevant, thus leading to more interdisciplinary research. Context-driven research is a natural strategy to make such endeavors possible. Empirical methods, which are essential to relate contextual factors and the performance of software engineering solutions, are now widely and commonly used in research.

To further increase relevance in software engineering research, and therefore ensure its successful future, several things must however change in our research community. We all have, in different roles, a responsibility to improve the current state of affairs.

First, top journals and conferences must acknowledge that context-driven research is needed,

valuable, and challenging. Such research focuses on practicality and scalability in realistic conditions, and therefore brings essential contributions for an engineering field. Mathematically demonstrating the correctness and completeness of practical, scalable solutions is rarely possible, similarly to proving the correctness of an industrial-strength software system. Indeed, many SE solutions rely on heuristics which may not be provable mathematically but yet evidence of their soundness and usefulness can be provided by empirical means.

Furthermore, funding agencies must help promote and reward collaborative research between public research and industry, thus enabling context-driven research. Such collaborations must be encouraged as researchers need industry to identify and characterize problems, develop innovative and practical solutions, and evaluate them in realistic contexts. Given the many challenges that context-driven research entails, it is unrealistic to believe that this type of research will expand if financing it remains difficult. As a result, a number of countries have already put in place innovative programs, an example being the long-running Collaborative Research and Development grants in Canada. Related to this, hiring and promotion committees must emphasize and reward evidence of impact on engineering practice.

Industry increasingly realizes that it can better support innovation and quality improvement when collaborating with researchers. Collaborative research can indeed be extremely beneficial if it is organized to be synergistic and fulfill the objectives of all parties. Many companies, especially the ones who cannot afford to run their own R&D facilities, are well aware that addressing their modern-day software engineering problems requires targeted collaborations with specialized researchers. While increasingly invested in software development, companies often have their core competence areas defined around business domains and systems, rather than software per se. Consequently, most companies do not have the necessary resources and know-how to develop effective solutions to their software engineering problems. This makes academia-industry collaborations a win-win, and presents a unique opportunity for increasing the impact of software engineering research, as long as researchers lend a keen ear to industrial needs and develop a better appreciation of the special nuances introduced by context.

Acknowledgements

This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), from the Luxembourg National Research Fund (FNR) under grant agreement FNR/P10/03.

References

1. C. Arora, M. Sabetzadeh, L. C. Briand, F. Zimmer, "Automated Checking of Conformance to Requirements Templates Using Natural Language Processing", IEEE Trans. Software Eng.41(10): 944-968 (2015)
2. T. Ball, V. Levin, S. K. Rajamani, "A decade of software model checking with SLAM", Commun. ACM 54(7): 68-76 (2011).
3. V. R. Basili, "Learning through Applications: The Maturing of the QIP in the SEL", in Making Software, Andy Oram and Greg Wilson, eds., O'Reilly, 2011.
4. V. R. Basili, "A Personal Perspective on the Evolution of Empirical Software Engineering", in Perspectives on the Future of Software Engineering, Jurgen Munch, Klaus Schmid (Eds.), Springer, pp.255 - 274, 2013.
5. L. C. Briand, D. Bianculli, S. Nejati, F. Pastore, M. Sabetzadeh, "The Case for Context-Driven Software Engineering Research - Generalizability Is Overrated", IEEE Software 34(5): 72-75 (2017)
6. R. T. Fielding, R. N. Taylor, J. R. Erenkrantz, M. M. Gorlick, J. Whitehead, R. Khare, P. Oreizy, "Reflections on the REST architectural style and "principled design of the modern web architecture" (impact paper award)", ESEC/SIGSOFT FSE 2017: 4-14

7. R. Matinnejad, S. Nejati, L. C. Briand, T. Bruckmann, “Automated Test Suite Generation for Time-Continuous Simulink Models”, ICSE 2016: 595-606

8. B. Meyer, “Applying 'design by contract”, Computer 25(10): 40-51, Oct. 1992.

Victor Basil is Professor Emeritus of Computer Science at the University of Maryland. He holds a PhD degree in Computer Science from the University of Texas and two honorary degrees. He was a founding director of the Software Engineering Laboratory at NASA/GSFC and the Fraunhofer Center-Maryland. He has worked on measuring, evaluating, and improving software development processes and products. He is a fellow of the ACM and IEEE. Contact him at basili@cs.umd.edu.

Lionel Briand is Vice-director, professor and FNR PEARL chair at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. Lionel was elevated to the grade of IEEE Fellow for his work on testing of object-oriented systems. He was also granted the IEEE Computer Society Harlan Mills award and the IEEE Reliability Society Engineer-of-the-year award for his work on model-based verification and testing. His research interests include: model-driven development, testing and verification, search-based software engineering, and empirical software engineering. Contact him at lionel.briand@uni.lu.

Domenico Bianculli is a research scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. He holds a PhD degree from Università della Svizzera italiana, Switzerland. His research interests include: run-time verification, specification languages, modeling and enforcing of access control policies, program analysis for security, and incremental verification. Contact him at domenico.bianculli@uni.lu

Shiva Nejati is a research scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. She holds a PhD degree from the University of Toronto, Canada. Her main research area is software engineering, with specific interests in model-based development, analysis of real time embedded systems, search-based software engineering, behavior analysis, and specification and design methods. Contact her at shiva.nejati@uni.lu.

Fabrizio Pastore is a research scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. He holds a PhD degree from University of Milano Bicocca. His research interests cover different software engineering topics like automated software testing, program analysis, self-healing systems and crowdsourcing. Contact him at fabrizio.pastore@uni.lu.

Mehrdad Sabetzadeh is a senior research scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg. He holds a PhD degree from the University of Toronto, Canada. His research interests are focused on model-based software engineering with a particular emphasis on model-based verification and validation of business- and mission-critical applications. Contact him at mehrdad.sabetzadeh@uni.lu.

Tweets

- Impactful software engineering research needs to account for context: human, domain and organizational factors
- Collaborative research between academia and industry is essential to enable impactful software engineering research
- Context-driven SE research focuses on practicality and scalability and is essential for an engineering field