The Need for an Open Corpus of Software Architecture Descriptions

Jens Knodel
Fraunhofer IESE
Fraunhofer-Platz 1, 67663
Kaiserslautern, Germany
jens.knodel@iese.fraunhofer.de

Jim Buckley
Lero – The Irish Software Research Centre
University of Limerick, Limerick
Ireland
jim.buckley@ul.ie

Sebastian Herold
Karlstad University
651 88 Karlstad
Sweden
sebastian.herold@kau.se

***Abstract*— Software architectures are the conceptual tool to share information about key aspects of a software system and to enable reasoning about the principal, most fundamental, and often most difficult-to-change design decisions of the system. Studies of failed software systems give evidence that architecture drift, erosion or degradation is a prevalent problem in industrial practice. But a recent systematic literature review [9] indicates that research currently investigates compliance checking or inconsistency detection only. To advance research we need an open and grounded corpus of software architecture description – serving as a basis for more sophisticated studies beyond detection only. Such a corpus could enable (1) to evaluate new approaches, (2) to provide means for fixing degradation (when it occurs or a-posteriori), (3) to compare and benchmark approaches and, ultimately, (4) enable longitudinal studies in the field.**

Keywords—software architecture, software architecture description, drift, erosion, degradation, open corpus

I. INTRODUCTION

Software architectures are the conceptual tool to share information about key aspects of a software system [1] and to enable reasoning about the principal, most fundamental, and often most difficult-to-change design decisions of the system. These principle design decisions and their manifestation in source code massively influence the system's ability to meet its business goals: realization of its functional requirements and achievement of its non-functional qualities, the so-called "-ilities". In essence, the software architecture describes how well the software system delivers value to the users of the system, and also how well the development organization responsible for software system can manage its overall lifecycle, covering maintenance, evolution, migration, retirement (of parts and/or technologies) and the like.

Software architectures prescribe the desired decomposition into components, modules and the dependencies among them (intended architecture). Developers then translate the abstract building blocks of the system into source code (realized architecture). In studies of failed software systems evidence has been presented that almost all of the studied system's implementations exhibit significant amount of architecture degradation (sometimes also referred to as architectural drift, architecture erosion, architecture violations, or structural violations) [2-7]. In addition, controlled empirical experiments indicates that changes to software systems affected by architecture degradation can take more than twice as long and result in substantially lesser results, in terms of correctness [8].

Hence, there has been a rich vein of research in the area of architecture degradation. A recent systematic literature review [9] covering 119 papers on software architecture degradation suggests that about 65% of the published, peer-reviewed research articles in this area investigate consistency checking or inconsistency detection only. A much lower percentage of research articles touch upon advanced research topics that go beyond pure detection of architecture degradation: analysis of degradation causes (ca. 13%), fixing of degradation (ca 14%), prevention of degradation (ca. 17%). This situation is surprising and challenging given the results from studies investigating architecture consistency in isolation: Some historical/longitudinal studies give evidence that software architecture degradation sometimes is not removed manually, even when detected [10] .

II. GENERAL RESEARCH CHALLENGES

Various studies [2-12] suggest that software architecture degradation is a prevalent problem, and not only in commercial software development. Research has developed many different techniques to address facets of software architecture degradation. The most intensively investigated aspect is checking for inconsistencies between a specified prescribed architecture and the descriptive architecture, as manifested in the source code of system. The techniques to check for consistency/inconsistency differ in the way a prescriptive architecture is described, the expressiveness of the approach w.r.t. the prescriptive architecture, and the type of implementation, e.g. the programming language, which can be checked.

Empirical evidence in software architecture degradation research is difficult to obtain. Research in the field evaluated by case studies or experiments of commercial systems generally requires access to a system for which not only the system's implementation is available: the prescriptive architecture also needs to be documented or a system expert/architect needs to be available to recover the architecture. In addition, due to the sensitivity of data and results, which potentially could reveal serious architectural degradation, detailed findings and the source code are often subject to non-disclosure agreements. Open source systems, on the other hand, do often not have a documented prescriptive architecture and access to open source project members, who could recover the original intended, primary

design decisions, is often difficult to obtain [11]. This makes it difficult, or even impossible, to perform transparent, high-quality studies. Likewise, different approaches and techniques can hardly be compared or benchmarked, as the systems used for evaluation differ and lack a universally usable description of their prescriptive architectures.

Hence it is not really surprising that the systematic literature review on software architecture degradation mentioned earlier [9] raises the following issues, across the literature:

- In the last five years, ca. 45% of published research did not contain proper empirical evaluation. Most of these papers suggested new techniques or extension to existing ones and validated them by a line of argumentation or synthetic examples.
- Most of the empirical evaluations deal with technical aspects, such as showing applicability or correctness. Very little of this research deals with usability, user perception, or the analysis and assessment of architecture degradation.
- Case studies, as one form of empirical studies, are dominating. This strong focus can be a threat to the generality of results. Wider scoped studies such as controlled experiments and surveys, could improve the validity of results.
- The large majority of case studies investigate degradation at a single point in time, even though the phenomenon has a strong longitudinal aspect: Few historical or longitudinal studies exist.
- Most studies focus on detecting structural architecture degradation only. Other kinds of degradation are underrepresented (e.g., behavioral architecture degradation [12]).

III. CONCLUSION

We expect that an open corpus of software architecture descriptions can drastically reduce the effort of empirical research in the field of software architecture, particularly in software architecture degradation. It will serve as an easily accessible dataset, replacing the often-cumbersome search for systems with available descriptions of prescriptive architecture and/or time-consuming architecture recovery procedures. Our next steps in this direction are:

- Shape a corpus of software architectures descriptions that allows studies in the areas of (1) prevention of architectural violations, and (2) identification, analysis and remediation of pre-existing violations. This shaping would encompass the conceptual, organizational, and technical requirements and constraints of such a repository.
- Populate this open corpus with prescriptive architectures “grounded and validated” by expert knowledge. This means that prescriptive architectures are created or reviewed by an expert, such as the original architect (or a lead developer of that particular system). This corpus could then be used for replication of studies, for benchmarking, and for exploring the usefulness of new and/or existing techniques.
- Exemplify and disseminate the usefulness of the open corpus by authoring an extensive experience report, making the corpus available to the research community and, additionally, publishing relevant findings at conferences in the field of software architecture (e.g., conferences like ICSE, ICSA, ECSA).

Such corpora, or repositories, have been proven successful for other software engineering disciplines, e.g., testing, source code analysis, or project management (e.g., [13]). This is also reflected in the so-called Artifacts Track of many software engineering conferences (e.g., recent editions of ICSME or FSE). Corpora are successful because they enable a common basis for discussing research approaches, tools results, and research results. They allow comparison and benchmarking of results, as they deliver a predefined and structured way for representing results and make it easier to perform replication studies.

REFERENCES

[1] ISO/IEC, ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems. 2007, ISO/IEC/(IEEE). p. 23.

[2] Murphy, G.C., D. Notkin, and K.J. Sullivan, Software Reflexion Models: Bridging the Gap between Design and Implementation. IEEE Trans. Softw. Eng., 2001. 27(4): p. 364-380.

[3] Bourquin, F. and R.K. Keller. High-impact Refactoring Based on Architecture Violations. in 11th European Conference on Software Maintenance and Reengineering (CSMR'07). 2007.

[4] Rosik, J., et al., An industrial case study of architecture conformance, in Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. 2008, ACM: Kaiserslautern, Germany. p. 80-89.

[5] Godfrey, M.W. and E.H.S. Lee, Secrets from the Monster: Extracting Mozilla's Software Architecture, in Intl. Symposium on Constructing software engineering tools (CoSET 2000). 2000.

[6] Brunet, J., et al., Five Years of Software Architecture Checking: A Case Study of Eclipse. IEEE Software, 2015. 32(5): p. 30-36.

[7] Knodel, J., Muthig, D., Naab, M., & Lindvall, M. Static Evaluation of Software Architectures, in Proceedings of the Conference on Software Maintenance and Reengineering (CSMR 2006), 2006.

[8] Knodel, J., Sustainable Structures in Software Implementations by Live Compliance Checking. 2011, Fraunhofer Verlag.

[9] Herold, S., M. Blom, and J. Buckley, Evidence in architecture degradation and consistency checking research: preliminary results from a literature review, in Proccedings of the 10th European Conference on Software Architecture Workshops. 2016, ACM: Copenhagen, Denmark. p. 1-7.

[10] Buckley, J., et al., Real-Time Reflexion Modelling in architecture reconciliation: A multi case study. Information and Software Technology, 2015. 61: p. 107-123.

[11] Ding, W., et al. How Do Open Source Communities Document Software Architecture: An Exploratory Survey. in 2014 19th International Conference on Engineering of Complex Computer Systems. 2014.

[12] Ackermann, C., et al. An Analysis Framework for Inter-system Interaction Behavior. in 2008 19th International Symposium on Software Reliability Engineering (ISSRE). 2008.

[13] Tempero, E., et al. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. in 2010 Asia Pacific Software Engineering Conference. 2010.