Poster: Making Well-Informed Software Design Decisions

Arman Shahbazian, Youn Kyu Lee, Yuriy Brun, Nenad Medvidovic

University of Southern California
Los Angeles, CA, USA
{armansha, younkyul, neno}@usc.edu

University of Massachusetts Amherst
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Design decisions software architects make directly impact system quality. Real-world systems involve a large number of such decisions, and each decision is typically influenced by others and involves trade-offs in system properties. This paper poses the problem of making complex, interacting design decision relatively early in the project's lifecycle and outlines a search-based and simulation-based approach for helping architects make these decisions and understand their effects.

CCS CONCEPTS

• **Software and its engineering → Designing software**;

KEYWORDS

Model-driven engineering; software architecture

ACM Reference Format:
Arman Shahbazian, Youn Kyu Lee, Yuriy Brun, and Nenad Medvidovic. 2018. Poster: Making Well-Informed Software Design Decisions. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27–June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194961

1 MAKING ARCHITECTURAL DESIGN DECISIONS

Fairly early in a software system's life cycle, software architects make a set of critical design decisions that form the system's architecture. Some of these decisions are made fresh, while others are borrowed from previous versions of the system or other existing similar systems. Some decisions are influenced by established architectural styles and patterns, some by selected implementation frameworks and libraries, and some by the architects' expertise and prior experiences. To name just a few aspects of systems affected, architectural design decisions address system structure, system behavior, component interaction, system deployment, system evolution, and non-functional properties [28]. It has been long accepted that the number grows quickly with the complexity of the system [5]. As an example, designing Apache Hadoop required well over one hundred design decisions [**? ?**]. And the large number of decisions is only part of quantifying the difficulty of system design, as many decisions involve intertwined factors and force trade-offs in system properties that must be considered [4, 24].

A software architecture includes many variation points that can take on one of a set of possible alternatives, e.g., employing either an encrypted or plain-text data storage, or using either a relational database, a document database, or a key-value store. A design decision is the selection of one of these alternatives. Ideally, when making a decision, architects carefully assess each alternative and how it satisfies or affects each of the system's requirements; however, this is frequently not done in practice [6]. The Healthcare.gov portal is a recent example of ineffective design-decision impact assessment, leading to serious technical problems at launch [17] and a development cost of US$1.5B, despite original estimates of ~US$100M [16]. For example, the portal's downtimes of up to 60% were caused by flawed architectural and deployment design decisions. The system was deployed using a single-node NoSQL database that also stored federal government employee information, rather than using a distributed database configuration. This decision alone caused half of the system outages [30].

As long as the design decisions are even partially independent, the space of possible systems resulting from all the selections of concrete alternatives grows exponentially in the number of decisions. Manually comparing these potential systems is infeasible for most systems, and the community has recognized the need for tools to support architects in evaluating these decisions [12].

2 STATE-OF-THE-ART DECISION SUPPORT

To make effective design decisions, architects need to understand the effects of the decisions on the final system. Thus, it is helpful to be able to objectively assess these potential final systems before building the systems and even before making all the decisions [24]. The existing approaches to assess such systems rely on static or dynamic analysis of system models. Static analysis techniques tend to require architects to develop complex mathematical models, which imposes steep learning curves and significant modeling effort and limits on the resulting system's scalability [3, 11, 21]. Depending on the mathematical models they rely on, these techniques are confined to specific kinds of software system models, or are heavily dependent on error-prone and sometimes inaccessible expert inputs [10].

Dynamic analysis techniques — architectural model *simulations* [2, 9, 15, 18] — come with shortcomings of their own (e.g., false negatives, longer execution times), but are more capable of capturing the randomness reflective of reality [14] and are more amenable to constructing models that are tailored to the task at hand. Despite

ICSE '18 Companion, May 27–June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194961

notable efforts [7, 29, 31], simulations of software architectural models have not been as widely employed as traditional static analyses [1] because creating simulatable system designs is difficult [9], running simulations on complex models is time consuming and requires explicitly addressing scalability issues [23], trade-offs in system properties caused by design decisions complicate quantitative assessment [22], and analysis of system behavior may rely on massive datasets [8, 25].

3 SIMULATION-BASED SEARCH

One possible way to address the shortcomings of prior approaches is to use a search-based strategy together with architectural-model-driven discrete-event simulation to evaluate the potential systems corresponding to the model's design decisions. Such an approach can help architects make design decisions by providing concrete simulation-based evidence on the effects each decision (and their combination) can have on the final system and its specific properties and requirements.

The challenges of such an approach include (1) enabling architects to effectively specify simulatable system-design models that precisely capture the design decisions, their alternatives, and their interactions, as well as system properties of interest, and (2) scalably executing the potentially many concrete instantiations of the models with each design decision confined to a specific alternative to evaluate the decisions' impact on system properties.

While simulating all of the potential systems that result from concrete instantiations of the decisions is feasible for real-world systems, heuristic-based search has been highly successful in relatively efficiently approximating optimal solutions [13]. Even exploring a subset of the overall search space is likely to be helpful and enable architects to make better informed decisions. Existing optimization techniques have been successfully applied to similar problems, e.g., computing the effects of possible deployment architecture on system quality of service [19]. Further, modern cloud computing enables executing thousands of system simulations in parallel, and recent advances data processing and analysis [27] can help create specialized techniques to increase the efficiency of the required analyses.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grants no. CCF-1564162 and CCF-1618231, and by Huawei Technologies Co., Ltd.

REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering (TSE)*, 39(5):658–683, 2013.
[2] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.
[3] B. Boone, S. Van Hoecke, G. Van Seghbroeck, N. Joncheere, V. Jonckers, F. De Turck, C. Develder, and B. Dhoedt. Salsa: Qos-aware load balancing for autonomous service brokering. *Journal of Systems and Software*, 83(3):446–456, 2010.
[4] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina. Goal-centric traceability for managing non-functional requirements. In *International Conference on Software Engineering (ICSE)*, pages 362–371. ACM, 2005.
[5] P. C. Clements. *Software architecture in practice*. PhD thesis, Software Engineering Institute, 2002.
[6] M. D'Ambros, A. Bacchelli, and M. Lanza. On the impact of design flaws on software defects. In *QSIC 2010 (10th International Conference on Quality Software)*, pages 23–31. IEEE, 2010.
[7] P. de Oliveira Castro, S. Louise, and D. Barthou. Reducing memory requirements of stream programs by graph transformations. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 171–180. IEEE, 2010.
[8] G. Edwards. *Automated synthesis of domain-specific model interpreters*. PhD thesis, University of Southern California, 2010.
[9] G. Edwards, Y. Brun, and N. Medvidovic. Automated analysis and code generation for domain-specific models. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 161–170. IEEE, 2012.
[10] N. Esfahani, S. Malek, and K. Razavi. GuideArch: Guiding the exploration of architectural solution space under uncertainty. In *International Conference on Software Engineering (ICSE)*, pages 43–52, 2013.
[11] S. S. Gokhale. Software application design based on architecture, reliability and cost. In *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*, volume 2, pages 1098–1103. IEEE, 2004.
[12] L. Grunske, P. Lindsay, E. Bondarev, Y. Papadopoulos, and D. Parker. An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In *Architecting dependable systems IV*, pages 188–209. Springer, 2007.
[13] M. Harman. The current state and future of search based software engineering. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 342–357, 2007.
[14] W. D. Kelton and A. M. Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.
[15] M. Langhammer, A. Shahbazian, N. Medvidovic, and R. H. Reussner. Automated extraction of rich software models from limited system information. In *IEEE/IFIP Working Conference on Software Architecture (WICSA)*, pages 99–108, April 2016.
[16] D. R. Levinson. An overview of 60 contracts that contributed to the development and operation of the federal marketplace, oei-03-14-00231. http://oig.hhs.gov/oei/reports/oei-03-14-00231.pdf, August 2014.
[17] F. Luke Chung. Healthcare.gov is a technological disaster. http://goo.gl/8B1fcN, 2013.
[18] J. Ma, F. Le, A. Russo, and J. Lobo. Declarative framework for specification, simulation and analysis of distributed applications. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1489–1502, 2016.
[19] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *IEEE Transactions on Software Engineering (TSE)*, 38(1):73–100, 2012.
[20] M. Marseguerra, E. Zio, and L. Podofillini. Genetic algorithms and monte carlo simulation for the optimization of system design and operation. In *Computational Intelligence in Reliability Engineering*, pages 101–150. Springer, 2007.
[21] J. D. McGregor, F. Bachmann, L. Bass, P. Bianco, and M. Klein. Using arche in the classroom: One experience. Technical report, DTIC Document, 2007.
[22] G. Me, C. Calero, and P. Lago. Architectural patterns and quality attributes interaction. In *IEEE Workshop on Qualitative Reasoning about Software Architectures (QRASA)*. IEEE, 2016.
[23] C. Poloni and V. Pediroda. Ga coupled with computationally expensive simulations: tools to improve efficiency. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 267–288, 1997.
[24] P. Potena. Composition and tradeoff of non-functional attributes in software systems: research directions. In *Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 583–586. ACM, 2007.
[25] A. Shahbazian, G. Edwards, and N. Medvidovic. An end-to-end domain specific modeling and analysis platform. In *International Conference on Software Engineering (ICSE)*. IEEE, 2016.
[26] A. Shahbazian, Y. K. Lee, D. Le, and N. Medvidovic. Uncovering architectural design decisions. *CoRR*, abs/1704.04798, 2017.
[27] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31(1):1–31, January 2017.
[28] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.
[29] N. Trčka, M. Hendriks, T. Basten, M. Geilen, and L. Somers. Integrated model-driven design-space exploration for embedded systems. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 339–346. IEEE, 2011.
[30] US Department of Health and Human Services. Healthcare.gov progress and performance report. http://goo.gl/XJRC7Q, 2013.
[31] A. Vescan. A metrics-based evolutionary approach for the component selection problem. In *Computer Modelling and Simulation, 2009. UKSIM'09. 11th International Conference on*, pages 83–88. IEEE, 2009.