Architectural Intelligence: a Framework and Application to e-Learning

Jan Martijn E.M. van der Werf, Casper van Schuppen, Sjaak Brinkkemper, Slinger Jansen, Peter Boon, and Gert van der Plas

Department of Information and Computing Science,
Utrecht University
{J.M.E.M.vanderWerf C.vanSchuppen, S.Brinkkemper, S.Jansen, P.Boon, G.A.J.vanderPlas}@uu.nl

Abstract. Architects need insights on the extent to which quality attributes are satisfied in order to adequately evolve software systems. This is especially true for software products, which are delivered to many customers and undergo multiple releases, thereby offering ample opportunities for re-design. Available techniques to validate quality attributes either rely on workshops with stakeholders or are based on design-time software artifacts. Many quality attributes, however, are better assessed at runtime when the software system is in operation. In this paper, we present an approach that enables the systematic processing and interpretation of software operation data to gain architectural knowledge about quality attributes. In addition to introducing this approach—which we call *Architectural Intelligence*—, we present through a case study on an e-Learning environment a formal framework based on process mining that enables the development of second-order information systems for analyzing software operation data to provide architectural intelligence.

1 Introduction

Quality attributes are an important aspect of the design of software architectures, as they determine whether the system guarantees characteristics such as reliability, security, and performance. Checking quality attributes is especially important for software products [5], for failing to meet some qualities may result in the dissatisfaction of multiple customers and may propagate to future product releases.

While it is relatively simple to check if an architecture contains some functionality—e.g., by verifying the existence of specific components in architectural diagrams—, the assessment of quality attributes is more difficult [8] as they are cross-cutting concerns that span across multiple components.

Existing methods include scenario-based approaches such as the Architecture Tradeoff Analysis Method (ATAM) [3], which assesses the adherence of a system architecture with quality attributes by conducting evaluation workshops with stakeholders. The drawback of this solution is that it heavily relies on the knowledge and analysis by experts, and is thus subjective, error-prone and time-consuming.

Other approaches, such as prototyping or simulation [3], obtain more objective results. However, these techniques are mostly performed during the design phase, and

quality attributes are not measured accurately during the operational phase of the software. In line with the work on requirements monitoring [6, 14], we argue for measuring quality attributes during software operation. By doing so, the architect gains valuable insights on how the software actually is being used that can serve to shape the next releases of a software product. For example, frequently used features could be grouped differently to improve overall performance.

Techniques like reverse engineering and software architecture reconstruction focus on reconstructing the design, or even the architecture of a system [10]. Architecture Compliance Checking (ACC) focuses on testing to which degree the realized artifacts conform to the intended software architecture [12]. However, all these techniques mainly focus on the functional aspects of the system, and tend to ignore quality attributes [10]. Some approaches take quality attributes into account, but tend to focus more on reconstructing architectural artifacts than on conformance checking of the quality attributes during the software operation phase.

Dynamic architecture compliance is founded on generally available resources, such as the call stack, process trees and message exchange [10]. However, these sources hardly contain sufficient information to measure the relevant quality attributes of an architecture [20].

In this paper, we present a holistic solution to quality attribute conformance checking by combining the Software Operation Knowledge Framework (SOK) [17]—a method that drives the evolution of software products based on operation knowledge—with insights from process mining [1]—a technique that enables extracting knowledge by analyzing the event logs of information systems.

In previous work, we have shown that process mining has a big potential in architectural analysis. For example, software operation data can be analyzed to validate whether the software adheres to the quality attributes set by the intended architecture [20]. Also, process discovery techniques allow to improve architecture [11], or even generate architecture documentation [19] and give insights in how the software is being used.

After describing the framework (Sec. 2) and results applied on a case study (Sec. 3), we outline our conclusions and present future directions (Sec. 4).

2 Architectural Intelligence Via Architecture Mining

One of the main tasks of the software architect is to design the system and to develop a project strategy. By making architectural design decisions, the architect constructs a software architecture. We consider the *software architecture* of a system to be the "set of structures needed to reason about the system, which comprises software elements, relations among them and properties of both" [3]. An important aspect of software architectures is to address the quality attributes that stakeholders expect the system to exhibit. These *quality attributes* are "measurable or testable properties of a system that are used to indicate how well the system satisfies the needs of its stakeholders" [3].

Opposed to the level of software architecture, many software analysis tools and techniques exist on the software level itself to build better software. For example, execution traces can be used to discover failures [13] or to support continuous integration [4].

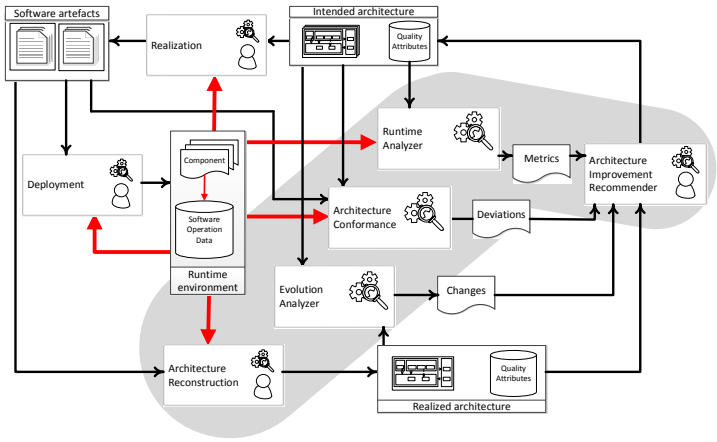


Fig. 1. Architectural Intelligence Framework based on software operation data

This paper advocates the use of runtime *software operation data* [17, 20] to assess the quality of a software architecture. Its analysis has the potential to close the loop between software architecture and the realized software product by monitoring and analyzing the actual realized system. In this way, data analytics techniques can provide architectural intelligence to the software architect. Such techniques aid in making a step forward toward continuous architecting of software systems. This is what we propose with *Architecture Mining*: the collection, analysis, and interpretation of software operation data to foster architecture evaluation and evolution.

The conceptual overview of Architecture Mining is depicted in Fig. 1. It considers an architecture consisting of a set of structures, represented by architectural views, and quality attributes that define quality constraints on the set of structures [3]. The black arrows indicate the dependencies between the different elements, whereas the red arrows indicate which steps can benefit from software operation data. The *intended* architecture is separated from the *realized* architecture due to architectural erosion [7]: the realization of software typically tends to drift away from the intended architecture. A system realization consists of many different artifacts, including source code and documentation, and is eventually deployed in some environment in which the system operates. Once the system is operational, it starts collecting software operation data.

The conceptual overview does not impy an order between the phases, only the dependencies are shown. Following the framework from top to bottom, the *intended architecture* is used to derive the *software artefacts*. The framework does not specify whether e.g. a waterfall method or an agile method is used to realize the software artifacts. The realized software artefacts are *deployed*, which results in an operational software system. The software artifacts are input for *Software Architecture Reconstruction* and *Architecture Conformance Checking*. The former results in an explicit *realized architecture*. Together with the intended architecture, the latter results in a set of *deviations*

from the intended architecture. Based on the software operation data, the *Runtime Analyzer* analyzes to which degree quality attributes specified in the intended architecture are satisfied, and results in a set of *metrics*. Comparing the realized architecture and the intended architecture gives insights in how the realized architecture drifted apart from the intended architecture. This is the task of the *Evolution Analyzer* which results in a set of *changes* with respect to the intended architecture.

The derived actual architecture, the metrics, deviations and changes are input for the *Architecture Improvement Recommender*, which ultimately results in an improved intended architecture.

3 Formalisation and Application in the e-Learning Domain

We applied the Architecture Intelligence framework to a software product called DME [1], an e-Learning application for practicing mathematics and aimed mainly at secondary school students. DME is actively used by teachers and researchers to develop courses, and to analyze didactic behavior [9]. Currently, around 120 schools in the Netherlands actively use the product for their mathematics classes. The product serves in total around 60,000 pupils, students and teachers. The purpose of this study was to evaluate *feasibility* and *effectiveness*.

3.1 Formalisation of Events

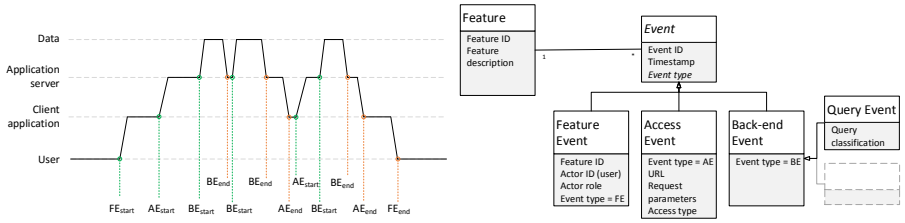
The premise of our formal framework is that software systems can be regarded as large-scale event-based systems: calling a system method, passing a message, and executing functionalities all occur at some time, and in some (partial) order, and are triggered by some source, such as the user, or other system modules. One way to monitor the system operation is by gathering and store these events [20]. Each event is generated by some resource and a time window in which it occurred. Events can contain additional data, such as the function being called, its parameters, etc. We refer the reader interested in the full formalisation to [15].

3.2 Runtime Analysis

A first simple use on operation data is to visualize the execution trace [13]. An example trace within our case study is shown as a UML Timing diagram in Fig. 2(a). This figure reveals useful information, such as which modules are called, and in what order. Each stage generates events, such as the start and end time.

To gather software operation data in a structured way, we build in this case study upon the process mining log framework XES [18], by adding a connection to the software architecture through architectural features. Different conceptual models exist to represent software operation data, such as [2], that focus on storing execution traces. Such models, however, mainly focus on the system execution itself, whereas with architectural intelligence, we would like to relate the operation data to higher abstractions, for which XES is highly suited.

[1] https://app.dwo.nl/site/index_en.html



(a) Feature call (b) Conceptual model

Fig. 2. Event-Based software operation data as feature calls

In our model, depicted in Fig. 2(b), events originate from different sources, and thus may have different properties. The *Event* is the core concept. Each event can be linked to some *Feature*, which represents a dedicated, coherent set of functionality. As per Fig. 2(a), events can be linked to different architectural elements. In this case study, we link the events to architectural elements by creating sub concepts of *Event*. An *Access Event* records the login of an end user, *Back-end Event*s represent events raised by the application server and *Feature event*s are triggered through the actions of end-users. As there might be different back-end systems in use, these can be differentiated as separate specializations of the *Back-end Event*, such as the execution of queries (*Query Event*s).

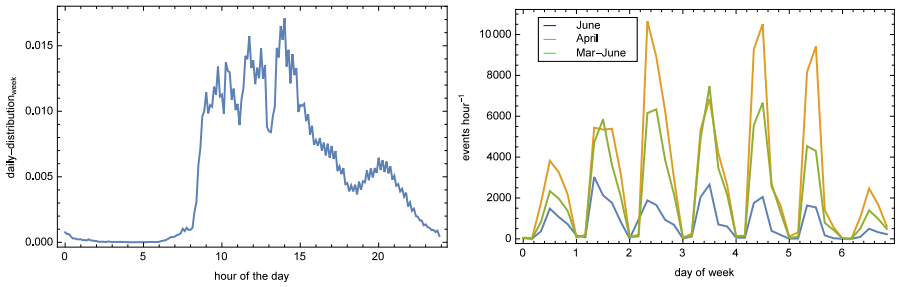
3.3 Design of the Data Collection through GQM

Collecting all possible software operation data is typically infeasible. An operationalized system generates too many events if all function and system calls are being recorded. On the other hand, just recording operation data is not sufficient. As an example, in an event-bus system, just recording which messages are sent over the bus is typically not sufficient: it is not possible to derive when a component has actually read the message. Additionally, the stored data needs to be related to the architectural elements.

Thus, the architect should think already during system design how quality attributes can be measured and which data needs to be collected to evaluate the metric. For greenfield designs, the architect can add an operation-data view to the architecture documentation, and design the operation-data collection from scratch. However, for evolving existing applications, this is more difficult, and the available data sources—such as performance monitors or even sources of functional data—need to be analyzed and integrated in the Architectural Intelligence Framework.

To assist the architect in deciding what data to record, we use the Goal-Question-Metric (GQM) method [16]. The architect defines a set of goals that the system should fulfill and for which insights are needed. Based on these goals, the architect collects the design decisions for that goal, the perspectives of the relevant stakeholders, the quality attributes that are related to this goal, and which questions and metrics the architect wants to monitor. Additionally, the architect should add a section on how data is generated and collected.

In our e-Learning environment, the architects aimed to answer the question “When is the application used for what purposes?”. Based on the metrics from the GQM tem-



(a) Time of the day (b) Over the week (starting at Sunday)

Fig. 4. Distribution of mutations and queries

plates, we gathered data to compute those metrics. The data gathering was done using the SOK acquisition tool JAMon [2] which provided data per query (22 MB) and per time period aggregated access data (17 MB). Each of the queries was subsequently classified and related to a specific feature. Additionally, the database mutations (inserts, updates) were examined based on extracted event data from daily backups of the database (MySQL binlog files, total of 35 GB). JAMon data was gathered for two weeks, and query mutation data for 6 months.

Early observations about the events over time (hour) and the load over the day show that it is convenient to split the analysis into week and weekend days. Analysis of the data shows that most usage originates from students doing exercises. Many features, as well as the overall view, show a specific usage daily distribution, shown in Fig. 4. Additionally, we looked at the fraction of mutations and queries, which seems to be rather constant over time. This gives the advantage that less data collection is required to sufficiently fill the metrics to answer the questions of the architects.

3.4 Evaluation with the Architect

Based on the metrics on the software operations, the software architecture was evaluated with the lead architect of DME. The study provided valuable insights, including answers to the goals and the initial concerns of the architect. During the study, a key challenge was to keep up with the architectural changes that occurred quickly. A number of important development changes were ongoing, including a refactoring of the database model and the exploration for the on-going adaptation for deployment in a cloud environment. It is essential to document these evolving aspects of the architecture, and moreover to analyze pending or optional architectural changes and base our recommendations on them instead of analyzing a static architectural snapshot.

We found that the definition of the questions from the architect, getting the required data, and doing the right analysis are a process by themselves and require a number of iterations in order to become sufficiently accurate. We have only started shaping this process, and the use of the GQM template was positive; however, more research is necessary to create a reliable, general method to support these activities.

[2] http://jamonapi.sourceforge.net/

Threats to validity First, the case study was conducted for a relatively short time frame over which the data was available and the model was stable, an essential aspect required to use this framework. To fill all the defined metrics as intended, we should have collected more data or employed a complete dataset. Determining the logging required to acquire the relevant software operation data is therefore essential for using this framework. Second, we did conduct a single case study; thus, the generalizability of the results is low. Third, we used convenience sampling to choose the application; nevertheless, DME is a large and real-world system that goes way beyond being a toy example. Fourth, the interviews with the lead architect were conducted through informally; in further researches, semi-structured interviews should be preferred to obtain more reliable knowledge.

4 Conclusion and Future Work

In this paper, we advocate the use of software operation data to provide useful information about the extent to which a software architecture conforms to the desired quality attributes. We have proposed Architecture Mining that synergistically combines existing fields like reverse engineering, software architecture reconstruction and architecture compliance checking with large scale data analysis techniques like process mining. These techniques add the capability to analyze event-based software operation data to discover dynamic architectural views to perform conformance checking about the quality attributes. The proposed approach emphasizes the central role of the software architecture in providing useful insights to software designers, especially to architects.

To validate the framework, we applied it on a case study on an real-world, large-scale e-Learning environment and evaluated the results with the architects, who perceived the gained insights and recommendations on flexibility and adaptability as effective. The case study revealed that defining and obtaining the right software operation data is a process in itself, and therefore we have proposed to use the GQM method to define the appropriate data sources for software operation data. This use of GQM was perceived positively by the architects. However, more research is required to create a reliable, general method to support this process.

We found that, in order to better manage log data and for the continuous acquisition of software operation data, the logging mechanisms should be an integral part of the design of a system. Preferably, the acquired data should be easily adaptable to the specific questions and goals of the architect during the software life cycle.

Initial research shows that analyzing software operation data provides useful insights on software usage from an architectural perspective. Although our baseline consists of solid techniques from different fields, further research is required to define a more comprehensive method that supports architectural intelligence. We argue architectural intelligence to be crucial for closing the gap between the design and operation of software, and a building block of continuous architecting.

References

1. W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, Berlin, 2011.

2. L. Alawneh, A. Hamou-Lhadj, and J. Hassine. Towards a common metamodel for traces of high performance computing systems to enable software analysis tasks. In *SANER 2015*, pages 111–120. IEEE, 2015.
3. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Series in Software Engineering. Addison Wesley, Reading, MA, USA, 2012.
4. M. Brandtner, E. Giger, and H. C. Gall. Supporting continuous integration by mashing-up software quality information. In *CSMR-WCRE 2014*, pages 184–193. IEEE, 2014.
5. Sjaak Brinkkemper and Xu Lai. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, 2007.
6. Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Adaptive socio-technical systems: a requirements-driven approach. *Requirements Engineering*, 18(1):1–24, 2013.
7. Lakshitha de Silva and Dharini Balasubramaniam. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, 2012.
8. L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.
9. P. Drijvers, M. Doorman, P. Boon, H. Reed, and K. Gravemeijer. The teacher and the tool: instrumental orchestrations in the technology-rich mathematics classroom. *Educational Studies in mathematics*, 75(2):213–234, 2010.
10. S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. *Software Engineering, IEEE Transactions on*, 35(4):573–591, 2009.
11. S. Klock, J. M. E. M. van der Werf, J. P. Guelen, and S. Jansen. Workload-based clustering of coherent feature sets in microservice architectures. In *ICSA 2017*. IEEE, 2017.
12. J. Knodel and D. Popescu. A comparison of static architecture compliance checking approaches. In *WICSA 2007*, pages 12–12. IEEE, 2007.
13. M. Perscheid, T. Felgentreff, and R. Hirschfeld. Follow the path: Debugging state anomalies along execution histories. In *CSMR-WCRE 2014*, pages 124–133. IEEE, 2014.
14. W. N Robinson. A requirements monitoring framework for enterprise systems. *Requirements engineering*, 11(1):17–41, 2006.
15. C. van Schuppen. Quality attribute tradeoff in learning infrastructure scaling. Utrecht University, 2015.
16. R. van Solingen, V. Basili, G. Caldiera, and D. H. Rombach. Goal question metric (GQM) approach. *Encyclopedia of Software Engineering*, 2002.
17. H. van der Schuur, S. Jansen, and S. Brinkkemper. Reducing maintenance effort through software operation knowledge: An eclectic empirical evaluation. In *CSMR 2011*, pages 201–210. IEEE, 2011.
18. H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. XES, XESame, and ProM 6. In *Information System Evolution*, volume 72, pages 60–75. Springer, Berlin, 2011.
19. J. M. E. M. van der Werf and E. Kaats. Discovery of functional architectures from event logs. In *PNSE 2015*, volume 1372 of *CEUR-WS*, pages 227–243. CEUR-WS.org, 2015.
20. J. M. E. M van der. Werf and H. M. W. Verbeek. Online compliance monitoring of service landscapes. In *BPM Workshops, Revised Papers*, volume 202 of *LNBIP*, pages 89–95. Springer, Berlin, 2015.