Information and Software Technology 61 (2015) 107–123

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Real-Time Reflexion Modelling in architecture reconciliation: A multi case study

Jim Buckley [a], Nour Ali [b], Michael English [a,*], Jacek Rosik [a], Sebastian Herold [a]

[a] The Irish Software Engineering Research Centre (Lero), University of Limerick, Ireland
[b] School of Computing, Engineering and Mathematics, University of Brighton, UK

ARTICLE INFO

Article history:
Received 2 October 2014
Received in revised form 24 January 2015
Accepted 24 January 2015
Available online 2 February 2015

Keywords:
Reflexion Modelling
Software architecture
Architecture consistency
Architecture conformance

ABSTRACT

Context: Reflexion Modelling is considered one of the more successful approaches to architecture reconciliation. Empirical studies strongly suggest that professional developers involved in real-life industrial projects find the information provided by variants of this approach useful and insightful, but the degree to which it resolves architecture conformance issues is still unclear.
Objective: This paper aims to assess the level of architecture conformance achieved by professional architects using Reflexion Modelling, and to determine how the approach could be extended to improve its suitability for this task.
Method: An in vivo, multi-case-study protocol was adopted across five software systems, from four different financial services organizations. Think-aloud, video-tape and interview data from professional architects involved in Reflexion Modelling sessions were analysed qualitatively.
Results: This study showed that (at least) four months after the Reflexion Modelling sessions less than 50% of the architectural violations identified were removed. The majority of participants who did remove violations favoured changes to the architectural model rather than to the code. Participants seemed to work off two specific architectural templates, and interactively explored their architectural model to focus in on the causes of violations, and to assess the ramifications of potential code changes. They expressed a desire for dependency analysis beyond static-source-code analysis and scalable visualizations.
Conclusion: The findings support several interesting usage-in-practice traits, previously hinted at in the literature. These include (1) the iterative analysis of systems through Reflexion models, as a precursor to possible code change or as a focusing mechanism to identify the location of architecture conformance issues, (2) the extension of the approach with respect to dependency analysis of software systems and architectural modelling templates, (3) improved visualization support and (4) the insight that identification of architectural violations in itself does not lead to their removal in the majority of instances.

1. Introduction

Software architecture aims to ensure prioritized non-functional requirements like maintainability and modularity are satisfied through appropriate macro-structuring of software systems [1]. However, often systems grow without an explicitly defined architecture, or have drifted over time from their originally designed architecture [2,3]. In such cases, it is unlikely that the desired non-functional requirements have been delivered.

Early work towards addressing this issue focussed on architecture recovery: deriving the software's architecture from the source code of the existing system, typically from the source code dependencies of these systems [4,5]. However, even though software architects were often allowed to confirm or refute the suggestions of such analyses, they did not drive the process, thus limiting their ability to impose their desired architecture on the system [6]. In addition, these approaches often suffered from the 'garbage-in, garbage-out' phenomenon [7], whereby any architecture derived from analysis of a system without an initial architecture (defined or adhered to) is likely to be flawed.

More recently approaches to retro-fit the intended architecture (as originally or retrospectively defined) onto systems, have been developed to address these issues. This work is referred to in the literature as architecture reconciliation [7] architecture conformance [3] or compliance checking [9]. Several techniques have been proposed in this area, ranging from allowing architects probe

* Corresponding author.

http://dx.doi.org/10.1016/j.infsof.2015.01.011
0950-5849/

the architecture of specific points in the system [10,11] by defining textual rules, to more system-encompassing specifications like Reflexion Modelling (RM). In Reflexion Modelling [12], for example, the architect is initially prompted to explicitly state their ideal (as-intended) architecture for the system, as a simple vertices-and-edges diagram in which vertices represent architectural modules and edges represent the expected/allowed dependencies between these modules. The architect is then asked to map elements of the source code to the vertices in this as-intended architecture.

The approach parses the system to identify dependencies between the source code elements mapped to different vertices in the as-intended architecture, thus allowing corroboration or contradiction of the relationships proposed by the architect with respect to the as-implemented system. It is anticipated that the architect would then update the system to more fully align it with the as-intended architecture. Such an approach allows the architect drive the process from their architectural perspective, and focuses them on the parts of the system where specific architectural violations arise in the implemented system with respect to this perspective [13,14].

Several implementations of RM variants have been empirically evaluated through case studies [15–18], largely achieving positive feedback from industrial practitioners. Specifically, software architects have shown great enthusiasm for the information which it provides [13,19] and have even proactively sought such analysis again [20]. However, most of these studies focus on the ability of RM to identify inconsistencies [21,22] but do not concentrate on how the capabilities and insights provided by RM approaches are utilized by software practitioners and if the violations are subsequently removed.

This research assesses how the capabilities and insights provided by *Real-Time Reflexion Modelling* (RT-RM) approaches are utilized by practitioners. Here RT-RM [18] refers to a variant of RM where new architectural violations are presented to the architect, as new mappings are made between architectural modules and source code elements.

The research assesses how RT-RM is leveraged, determines if RT-RM results in the removal of architectural violations and also identifies the ways in which the participants would like to see RT-RM adapted/evolved in the future. It addresses these questions through a usage-analysis of a Just-In-Time Tool for Architecture Consistency (JITTAC) [34], which embodies a RT-RM approach. The empirical analysis is performed over five in vivo case studies in four different commercial organizations.

This paper is a substantial extension of the work presented by several of the authors in [18]. That paper reported on the first three case studies, providing a characterization of the modelling practices of the participants involved, their mapping (source code to RM vertices) preferences, and a characterization of the violations identified. In comparison this paper:

- Incorporates a larger data set, including two more case-studies, and retrospective interviews with the participants from all five case studies. This provided us with a larger, more representative data-set on which to base our findings and with the ability to explicitly assess the outcome of the RT-RM intervention longitudinally.
- Provides an expanded analysis of the data-set, resulting in several new findings. Specifically, while it does re-assess the modelling, mapping and architectural violations issues expounded upon in [18], with respect to the enlarged data set, it also includes findings on iterative analysis of systems through Reflexion models, findings on the outcome of RM in terms of the systems' architectural violations and an extended set of requirements for RM going forward.
- Presents an extended review of the related literature in this area (Section 2) which discusses the alternative approaches adopted and empirical work carried out in this and closely related areas.

The paper is structured as follows. Section 2 discusses the current state of the art with respect to various architecture conformance approaches. Section 3 focuses on one particularly successful approach: RM, discussing the approach, its empirical evaluations to date and how RT-RM builds on the approach. Section 4 describes the five case studies that make up the empirical component of this paper. Sections 5 and 6 present and discuss the findings across these case studies, with Section 7 examining the threats to validity. Finally, Section 8 concludes the paper.

2. State of the art

When a software system's implementation diverges from its designed architecture, it is referred to as *architectural drift* or *architectural erosion* [8,2]. This usually happens during software evolution, when the software undergoes changes as a result of bug fixes and updates, but may also happen during initial implementation of the system [23]. Architectural drift may result in the goals associated with the system's as-intended architecture being lost [20,21], often with serious consequences [3,24,25].

Architecture conformance aims to address architectural drift. It has been defined as a process of ensuring continued conformance of a subject system's implementation to its architectural design documentation and goals [26]. Here, design documentation is defined as any artefact created during the system's design (sometimes, even after the code is written) that documents the system's architecture. There have been many approaches suggested to increase architectural conformance, and these can be classified into several schools [8]. This section, reviews several of these approaches.

Tvedt Tesoriero et al. [16] divide architectural evaluation work into two main areas: pre-implementation architecture evaluation and implementation-oriented architecture conformance. In their classification, pre-implementation architectural evaluation involves the analysis of a proposed architecture to check whether it will fulfil the optimum number of the system's desired requirements. These approaches are used by architects during initial design and provisioning stages, before the actual implementation starts.

In contrast implementation-oriented architecture conformance approaches assess whether the implemented architecture of the system matches the intended architecture of the system [16,22,26]. Specifically, whereas architectural evaluation assesses the quality of the *proposed* architectural design, architectural conformance assesses whether the *implemented* architecture is consistent with the proposed architecture's specification, the goals of the proposed architecture, or both. Implementation-oriented conformance approaches can be split into two categories [27]:

- *Conformance by construction*: These approaches strive to achieve conformance through automated or semi-automated generation of artefacts, composing the system from the architectural descriptions. Several, established approaches implementing conformance by construction exist already. For example, approaches such as: generative programming [28], round-trip engineering [29], and model driven development [30,31] are being used in commercial software development. However, it is more difficult to apply these approaches retrospectively on existing systems: while model transformations have the ability to map from the implementation back to architectural models, these transformations are predefined and usually reflect well