

Towards Independent Software Architecture Review*

Antony Tang, Fei-Ching Kuo, and Man F. Lau

Faculty of Information and Communication Technologies
Swinburne University of Technology, Australia
{atang,dkuo,elau}@ict.swin.edu.au

Abstract. Many software architecture evaluation methods, proposed by the research community, have a common problem of engaging the same architects to perform architecture design and evaluation. This violates the independence of quality assurance and hence may lead to biased evaluation, thereby resulting in inferior architectural design. In this paper, we analyze current approaches and issues to software architecture quality assurance. We propose seven conditions for architectural design quality assurance and discuss existing challenges towards independent software architecture design review.

1 Introduction

Software architecture has been studied for over two decades. There are various standards such as IEEE 1471-2000 [1] which define software architecture, and research articles such as [2] and [3] which discuss different aspects of software architecture. In the industry, most sizeable software development organizations have a software architect role, and some of them employ architectural frameworks such as Zachman framework, TOGAF or RM-ODP to represent architectural information.

The IEEE 1471-2000 standards defines architecture as the *fundamental organization of a system* [1] and Perry and Wolf define it as a model of *elements, forms and rationale* [2]. Such definitions are generic and subject to interpretations. As a result, different architects may have different interpretations of the scope and the level of details of an architectural design. It can logically be assumed that the more details are included in the architectural design, the easier it is to validate the design. Such assumption depends on the level of certainty that an architectural design is viable and of good quality whilst it still depicts an abstract view of the system [4]. In other words, *how can one be assured that the architectural design would lead to a finished system that satisfies all its requirements, both functional and non-functional?*

In practice, many architecture design evaluation methods were proposed to answer this question. However, their scope of work and process vary widely depending on their objectives. For example, some measuring techniques evaluate quality attributes only rather than the entire design as a whole. Moreover, some architectural review methods only aim at evaluating selected aspects of an architecture design. Since software architecture design is influenced by many factors, the challenge is to find a

* This work was supported in part by a grant from the DCRG scheme, Faculty of Information and Communication Technologies, Swinburne University of Technology.

review method(s) that can address the quality of architectural design holistically while keeping unnecessary subjective interpretations to a minimum. In this paper, we propose an independent software architecture review (ISAR) approach.

2 Issues of Existing Techniques

Evaluation techniques [4, 5], such as *questioning techniques* and *measuring techniques*, are used for architectural design quality assurance. Among these evaluation techniques, expert reviewers are often required to assess the architectural design. When an evaluation is depended heavily on human expertise and face-to-face meetings, its soundness relies on the review team, especially when the contents and the details of architectural design description are not sufficiently documented. Under such circumstances, review team and architectural team may have different interpretations on the same design. Such phenomenon has been observed by the authors in real-life projects. Architects who argue most strongly can win a design argument, sometimes undermining objective assessments. This kind of scenarios has an adverse effect on the quality assurance capability of an evaluation technique.

Questioning Techniques assess the quality of architecture via qualitative questions. Architecture reviewers perform these qualitative evaluations via design reviews and inspections. Scenarios-based and attribute-based analysis methods such as SAAM [6], ATAM [7], SBAR [8]; and architectural review methodologies such as AT&T's software architecture review and assessment methods [9, 10] belong to this category. In these methods, reviewers analyze scenarios and quality attributes with the architecture team to determine if an architectural design satisfies the goals of a system.

There are four major issues related to these methods. First, they rely on expert analysis which is quite often a subjective process particularly when the analysis is based on incomplete specification and unavailability of architects' expertise. Second, architecture design reviews rely on reviewers' objectivity and expertise in uncovering issues in the architecture design. The objectivity and independence of the design evaluation may be compromised when the evaluation does not involve independent parties.

Third, some of these methods depend on the selection of relevant scenarios to identify critical assumptions and weakness of the architectural design [5]. What scenarios and architectural significant requirements (ASRs) [6, 7] are included in the evaluation would influence the quality. When the selected scenarios are incomplete or their relative priorities are unclear, there would be a lot of assumptions to be made by the architecture reviewers affecting the quality of the evaluation. Fourth, the architectural scenarios used to test the design can remain at a conceptual level and not all aspects of the architectural designs can be reviewed in details [8]. Thus architectural design flaws can be hidden from the reviewers unless sufficient details become available.

The first two issues relate to the dependency of human interpretation and its objectivity and the latter two issues relate to the quality and sufficiency of documentation.

Measuring Techniques assess the quality of architecture via quantitative measurements. These methods include the use of metrics, simulations and prototyping to collect information for evaluation. For instances, SBAR [9] supports simulation and mathematical models to analyze software qualities such as performance or fault-tolerance;

CBAM [10] and ArchDesigner [11] use utility measures for trade-off analyses; a metrics-based approach [12] evaluates an architecture by using module coupling and cohesion to predict the software quality.

Measuring techniques are useful in dealing with certain aspects of an architectural design but they are not comprehensive enough to assure the overall quality of the architectural design. They have two problems. First, for all measuring models, the input sensitivities for the studied quality attributes are crucial. Sensitivity degrees have an impact on the confidence of predicting design quality with respect to the quality attribute. For example, in Architecture Level Prediction of Software Maintenance (ALPSM) method, the accuracy of the sensitivity of the inputs used in the software modification model would affect the predicted maintenance effort [5]. Second, there is a lack of accurate estimates for quality requirement budgets, and a lack of accurate estimates on how a design model satisfies those quality properties such as performance and reliability [5, 13]. This is true for different levels of the architectural design, especially in the detailed levels.

3 Independent Software Architecture Review (ISAR)

Early detection of problems in architectural design through evaluation techniques reduces development costs and improves the quality of systems [5, 8, 14]. Thus, improving the effectiveness of evaluation techniques (in terms of problem detection) for architectural design is important. There are two approaches. First, one might propose totally new evaluation techniques that outperform existing ones. Second, one might enhance existing evaluation techniques by removing the problematic issues or reducing their negative impacts in order to achieve better evaluation. In this paper, we adopt the latter approach.

As discussed previously, most existing evaluation techniques face the problem of using the same architects to design and review the architectural design. Hence, human factors may lead to biased evaluation. To obtain an objective evaluation outcome, we argue that an Independent Software Architecture Review approach (ISAR) should be adopted as a standard practice of independent validation and verification (IV&V) [15]. In this section, we first analyze the barriers towards ISAR. Then, we propose seven conditions to facilitate better reviews through ISAR, and finally discuss the benefits as well as drawbacks towards applying ISAR.

3.1 Barriers towards an Independent Review

As discussed earlier, one key barrier of attaining independent evaluation is related to the lack of sufficient information provided to review teams for quality evaluation, which hinders the quality of the review process. Currently there are little or no standards on the minimum set of information necessary to support such evaluation. To address this, architecture design evaluation should be based on a well-defined engineering-style architectural design specification. The architectural description contained in such specifications must be structured, precise and detailed to avoid subjective interpretations.

Another key barrier towards an ISAR approach is the subjectivity of architects and evaluators based on their prior experience. There is no doubt that the expertise of architecture design and review teams is important. However, as noted in [4], there is sometimes a skepticism that exists between design and evaluation teams, the mentality of “*Why Should I Believe You?*” It hints strongly about the recognition of experience and the existence of subjective views based on that experience in the evaluation. Such phenomenon has been observed by the authors in real-life projects. Architects who have the experiences and expertise can play a dominating role during design review meetings, making it difficult for evaluators who are less experienced to argue their case.

To alleviate this problem, we suggest that the architecture team capture design rationale to provide further evidences and explanations to justify their decisions. Architectural design rationale helps to explain why a design decision is made and to justify any trade-offs [16]. They can articulate implicit constraints and assumptions in a design [17]. Most evaluation techniques, however, do not mandate the review of design rationale. Therefore, these techniques do not validate whether the design under evaluation was properly made under reasonable assumptions and constraints.

3.2 Supporting Independent Software Architecture Review (ISAR)

Software testing requires that the software to be tested is already programmed. Similarly, to enable an independent review, sufficiently documented information must be available. We argue that current software architecture practice does not provide all necessary documented information for an independent review. Reviews or evaluations are very much people dependent causing the problems discussed earlier. We therefore propose to work towards a design specification format that supports an ISAR approach based on the following conditions.

1. **Requirement completeness** – Based on the agreed ASRs of a system, there are no new requirements and scenarios arising from these ASRs and from the architectural design implementation.
2. **Requirement conflicts detection** – There are no requirements that conflict with each other in terms of achieving the business goals of the system.
3. **Architectural design completeness** – All functional and non-functional ASRs are satisfied by some design components.
4. **Architectural design conflict detection** – No parts of the architectural design will be in conflict with the other parts of the design because they cannot satisfy the constraints set by ASRs or a design.
5. **Quantifiable non-functional requirements fulfillment** – All quantifiable non-functional requirements should be explicitly documented, and the architectural design should describe how the requirements are fulfilled.
6. **Soundness of design decisions** – The design rationale has been sufficiently captured so that the quality assurance team can judge the soundness of key architectural design decisions.
7. **Architectural design sufficiency** – The architectural design description should have sufficient details and there are no design omissions that could cause architectural design changes.

These seven conditions for architectural design are based on the fundamental relationships that exist between the requirements, design outcomes and their design rationale [18]. That is, a requirement causes a design issue, therefore a decision must be made to create or select a design to satisfy or resolve the design issue.

The seven conditions underpin the quality of a review and they are subtly related to each other, further complicating their impacts on the review quality. First, we make a fundamental assumption about the scope of a system as defined by the requirement specification. We assume that the requirement statements are complete (*condition 1*) but this can be proved to be wrong during architectural review if reviewers uncover missing scenarios or find conflicting requirements (*condition 2*). The architectural specification would need to demonstrate that the design can satisfy all the requirements (*condition 3*), and that there are no conflicting design elements (*condition 4*). Often non-functional requirements such as performance and reliability are vaguely specified or not quantified. A verifiable specification would describe explicitly how these requirements can be fulfilled, for instance, a design component can process x transactions per second to fulfill the requirement (*condition 5*). All key design decisions must be justifiable. The justifications can be illustrated by what design options have been considered, as well as documenting the design trade-offs, the pros and cons that have been considered (*condition 6*). Finally, there must be enough details in the design to demonstrate that indeed the design is implementable and further design details would not change the architecture design (*condition 7*).

3.3 The Benefits of ISAR Approach

The seven review conditions discussed in previous section are applicable to both designers who prepare the architectural design specifications as well as to review teams who use an ISAR approach. Besides establishing certain guidelines for the architecture team to follow for supporting better software architecture review, they help to create a software architecture quality assurance baseline. Such baseline can further support quality assurance: (a) it supports independent quality assurance of the work delivered by the software suppliers; (b) it provides a check and balance mechanism to the architecture development team; (c) it assures the quality and the viability of the architectural design before development and implementation thus preventing costly rework; (d) it systematically measures the risks involved in the architectural design through exploring design details.

An ISAR approach alleviates two major existing problems of architecture review, namely, subjective interpretations of and lack of (or imprecise) information in architecture design. This can be achieved by identifying necessary information and conditions that can enable an objective software architecture design evaluation. The ISAR approach differs from existing evaluation techniques in a number of ways:

- **Objective of review.** ISAR approach focuses on verifying the technical feasibility and the requirements fulfillment of an architectural design. It does not review other aspects such as project management issues [13] or skills and responsibilities issues [6].
- **Specification-based assessment.** An engineering-style architectural design description would provide a better structured and more detailed specification than

the existing practice. Instead of questioning and interviewing architects, quality assurance teams can rely on architectural description.

- **Design rationale centric.** ISAR approach requires software architects to justify and document the design. Information such as design rationale, alternative design options and traceability of design to requirements allow the quality assurance team to understand the design's viability and justifications.
- **Sufficiency of details.** ISAR approach requires software architects to document the architectural design to a sufficient level of details to prove the viability of the design. Such details should be sufficient to a point where assumptions and risks would not adversely affect the implementation of such a design.
- **Less reliance on expert reviewers.** Although knowledgeable architects are required to perform the review, the ISAR approach has less reliance on the expertise of reviewers as existing architectural review techniques.

The ISAR approach would provide a more stringent review on the architectural design by strengthening the architectural design based on architectural descriptions. It complements rather than replaces architectural evaluation techniques. Similar to IV&V, ISAR approach has some drawbacks too. For example, the selection of the review team may play a part in achieving objective evaluation because, in some cases, bias may be introduced due to human factors such as past experience and prejudice.

4 The Next Step

In this paper, we suggest that the quality assurance function of existing software architecture evaluation techniques is not independent of the architects who create the design. However, a comprehensive quality assurance technique should be independent of the architects. Thus, it should depend on the information in the design specification as well as the review process. We posit that in order to improve the effectiveness of quality assurance of architecture design, we must first consider a software architecture design specification standard that meets the seven conditions. We suggest a number of further research directions in software architecture specification and architectural review:

- **Meta-model(s) for documenting architecture design.** Such meta-models are used to guide the architects to better prepare architecture design documentation for architecture evaluation. One issue of assessing the quality of architecture specifications is a lack of assessment criteria. Although this is to some extent addressed by having a standard such as IEEE 1471-2000 [1] for architectural description, a more detailed guideline from the perspective of quality assurance of architectural specification would be advantageous. An architectural design meta-model should encompass elements such as ASRs, design decisions, design components and their interrelationships. The interrelationships between model elements ought to provide traceability to allow reviewers to understand and assess the architectural design.
- **Quantifiable quality attributes.** Quality attributes in design, e.g. performance, should be specified in a verifiable way. This means the specification can show how quality requirements can be achieved by each relevant design component.

Often the design for such quality requirements is based on the judgments of architects without objective justifications. Condition 5 suggests that engineering-style specifications should be provided. This implies that standard templates and guidelines need to be created to help architects gather the information for the design specification.

- **Specification of design rationale.** Design rationale as an integral part of an architectural design specification should contain enough details to justify the key decisions of an architectural design. Architecture design is a set of design decisions but often this tacit knowledge is undocumented. It makes independent architectural review very difficult. To fulfill condition 6 and enable independent architecture design review, the documentation of design rationale in architecture specification needs to be improved.
- **Software architectural knowledge and design pattern.** Software architecture design is a function of designers' experience and knowledge that are inherently subjective. So checking the fulfillment of conditions 6 and 7 can be difficult. If architects claim to have fulfilled these conditions, how to verify these claims become the tasks of the review team. The study of architectural knowledge and architectural design patterns may provide support in this area, at least for similar design problems that have been solved previously. Assessment of the soundness of new design and its design reasoning could be based on previous design cases. The similarities and differences when comparing their requirements, design rationale and design outcomes can provide a baseline for reviewers.

In order to overcome subjective interpretation and insufficient or imprecise information in architectural design evaluation, we need to find ways to improve and standardize software architecture description. Such research should attract attention from the research community as well as from the software industry because of the impact to software architecture design in the development life-cycle, especially when an improved quality assurance process could potentially reduce the failures and the rework costs in system and software development. We have received supports from organizations in Australia to carry out case studies into the areas of architectural specification and architectural quality assurance process. We hope that the case studies would help improve their current practices in these two areas.

References

1. IEEE: IEEE Recommended Practice for Architecture Description of Software-Intensive System (IEEE Std 1471-2000). IEEE Computer Society, Los Alamitos (2000)
2. Perry, D.E., Wolf, A.L.: Foundation for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes 17(4), 40–52 (1992)
3. Bosch, J.: Software Architecture: The Next Step. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 194–199. Springer, Heidelberg (2004)
4. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods & Case Studies. Addison-Wesley, Reading (2002)
5. Dobrica, L., Niemela, E.: A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28(7), 638–653 (2002)

6. Obbink, H., Kruchten, P., Kozaczynski, W., Postema, H., Ran, A., Dominick, L., Kazman, R., Hilliard, R., Tracz, W., Kahane, E.: Software Architecture Review and Assessment (SARA) Report (version 1.0) (2002)
7. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison Wesley, Boston (2003)
8. Maranzano, J.F., Rozsypal, S.A., Zimmerman, G.H., Warnken, G.W., Wirth, P.E., Weiss, D.M.: Architecture reviews: practice and experience. *IEEE Software* 22(2), 34–43 (2005)
9. Bengtsson, P., Bosch, J.: Scenario-based software architecture reengineering. In: Proceedings of Fifth International Conference on Software Reuse, pp. 308–317 (1998)
10. Kazman, R., Asundi, J., Klein, M.: Quantifying the costs and benefits of architectural decisions. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), pp. 297–306 (2001)
11. Al-Naeem, T., Gorton, I., Babar, M.A., Rabhi, F., Benatallah, B.: A quality-driven systematic approach for architecting distributed software applications. In: Proceedings. 27th International Conference on Software Engineering (ICSE 2005), pp. 244–253 (2005)
12. Briand, L.C., Morasca, S., Basili, V.R.: Measuring and Assessing Maintainability at the End of High Level Design. In: Proceedings of IEEE Conference in Software Maintenance, pp. 88–97 (1993)
13. Avritzer, A., Weyuker, E.J.: Investigating Metrics for Architectural Assessment. In: Proceedings of the Fifth International Software Metrics Symposium, pp. 4–10 (1998)
14. Babar, M.A., Zhu, L., Jeffery, R.: A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In: Proceedings 2004 Australian Software Engineering Conference, pp. 309–318 (2004)
15. IEEE: IEEE Standard for Software Verification and Validation (IEEE Std 1012 - 2004) (2004)
16. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley, Reading (2002)
17. Lee, J.: Design Rationale Systems: Understanding the Issues. *IEEE Expert* 12(3), 78–85 (1997)
18. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software* 80(6), 918–934 (2007)