Supplementary Material: Architecture Enforcement Concerns and Activities - An Expert Study

Sandra Schröder, Matthias Riebisch, and Mohamed Soliman

University of Hamburg, Department of Informatics,
Vogt-Koelln-Strasse 30, 22527 Hamburg, Germany

1 Architects' Activities for Enforcement

During the interview we asked all participants the question: *How do you ensure that your architecture and your concerns are implemented as intended? Do you follow any strategies?*. The result of this question is a categorization of activities that architects apply in order to enforce and validate their architecture decisions. Figure 1 shows the identified categories. In the following we describe the two categories Coaching and Supporting and Assessing the Decisions' Implementation in more detail. Moreover we discovered several dimensions that are important for those activities.

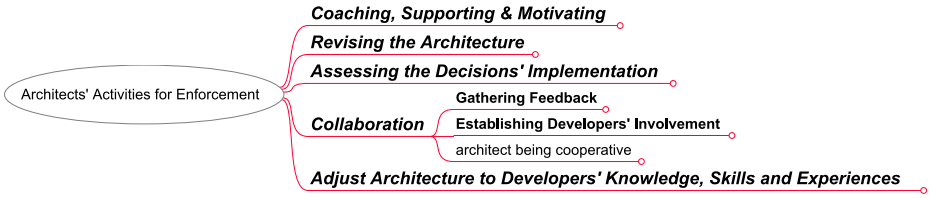

Fig. 1. Enforcement Activities as a mindmap

1.1 Coaching and Supporting

It is important that architects provide guidance during the implementation phase in order to support developers in their programming activities. Coaching was mentioned to be highly important both for explanation and motivation. Both is crucial to provide a clear picture and a shared understanding for architecture solutions, the corresponding design decisions, together with its goals, motivation and benefits: *"I have to explain [the developers] the term "architecture" and they have to internalize and understand what are the goal of architectural design and what has to be supported by the architecture..."* (code: architect as a coach, Participant **B**) or *"...as an architect you are committed to teach the developers*

and explain them what it is about..." (code: architect as a coach, Participant **G**). A combination of coaching and supporting can be done in several ways.

- **Architecture Models** For example, coaching can be supported by using appropriate architecture models so that a shared understanding between the people in the team about the architecture can be achieved: *"...we need models. In order that we can discuss appropriately, otherwise everyone has a different picture in his mind [...] and you don't know if the other has understood the same thing as you."* (code: using models for comprehension, Participant **B**).
- **Architectural Templates and Prototypes** For example, the architect can provide architectural templates and prototypes in order to guide how a specific decision has to be implemented or a specific technology has to be used, and he provides support by pre-fabricated building blocks. Architectural prototyping is another effective technique that combines support for developers with early identification and solution of high-risk aspects during early stages of the development process. Those templates can also be used to provide a reference during the implementation for the developers, that is for coaching and guiding purposes: *"...you build something as an example and present it to the developers..."* (code: architectural templates, Participant **A**). It was emphasized by participant **A**, that those templates should be built precisely and carefully according to architectural decisions and state-of-the-art best practices. Otherwise developers could violate the underlying decisions without knowing it because the architect did not show it correctly.

Dimensions for Feedback and Coaching During the data analysis we found an interesting statement made by an expert: *"...this is an aspect that you can easily underrate, but should not to be. There is no process, no methodology, no tool that makes sure having a good architecture. You simply need a number of people that have a feeling, the talent, the intelligence how to make architecture right..."* (codes: architecture awareness, personal quality; Participant B). This statement reports that personal quality in terms of knowledge, skills and experiences are an important factor in architecture enforcement. This was an important aspect we did not considered before, that is that people are as important - if not more - as dedicated tools and processes. Consequently we additionally searched for statements that were related non-technical aspects regarding architecture enforcement. We found the following dimensions: During the enforcement activities it is important to consider the different dimensions for feedback and coaching in an integrated way. Both dimensions emphasize that personal quality is an important factor in architecture enforcement. If those dimensions are not appropriately addressed during enforcement activities, it is likely that concerns as presented in the previous sections cannot be satisfied. We found the following dimensions during the analysis:

- **Skills, Experiences, Programming Habits:** Every developer has a different set of skills and experiences, e.g. from previous projects and from his

education. Those qualities and together with personal programming habits influence greatly how developers make low-level decisions and how they implement architectural decisions. The low-level decisions could violate important architecture decisions: *"...and if I leave it to the developers then it does not work since every developer has a different background and experiences. When I tell them that they should start with programming, then this leads to chaos..."* (code: programming habits and experience of developers).

- **Architecture acceptance**: We define architecture acceptance as the degree to which a programmer is willing to implement the prescribed architecture. The architect should always be *"... anxious that the developers accept the architecture and that they want to implement it this way."* (codes: encourage acceptance of developers for architecture, willingness; Participant B). The architects have to encourage developers to achieve the architecture's acceptance, otherwise it is likely that architecture rules are not followed and consequently violated.
- **Architecture awareness** describes the consciousness of developers regarding the prescribed architecture, its rationale and its goals that have to be achieved with it: *"skilled people do automatically know how they ensure architecture, because they know, why it should be like that. Then - without help - developers have the architecture in their mind and recognize if architectural goals are ensured or not."* (codes: architecture awareness, personal quality; Participant B). If developers are not aware of architecture goals it might happen that they unintentionally violate the architecture. The architect is responsible for achieving and encouraging architecture awareness appropriately; coaching and supporting are activities to address this dimension.
- **Shared understanding**: There must be a coherence of concepts between the members of a team about how an architecture looks like. Mostly, an architecture is constructed in the mind of the developers and the architects – either supported by models, diagrams or by speech – and it is important that all of them have the same imagination about the architecture in their mind: *"a common picture - keyword modeling - is very important here, to have a starting point and to have it started in the same direction"* (code: common understanding of architecture, using models for comprehension, Participant B). If a shared understanding about the architecture is achieved it is more likely that architectural rules are ensured and followed by the developers.

1.2 Collaboration - Involvement and Gathering Feedback

Regular discussion with the developers was mentioned as necessary, especially for discussing architectural violations or getting feedback from the developers concerning the architecture design. The architect has to be available for potential feedback given by the developers as they might not agree with the architecture solution given by the architect. Gathering feedback from the developers and having regular discussions with them is considered as crucial, e.g. in order to possibly revise the architecture and its underlying decisions (see Revising the Architecture). It is seen as valuable to involve the development teams during the code

review which allows discussions about architecture violations: *"...we had regular meetings with the developers and showed the developers where are deficiencies in the architecture and where rules were violated. Of course we tried not to blame them, but developed with them a solution in order to repair the deficiencies."* (code: discussion of violation, Participant E). Collaboration is strongly related with the activities "Coaching and Supporting" and "Revising the Architecture".

1.3 Adjust Architecture to Developers' Knowledge, Skills and Experiences

This activity is also strongly related with the "Dimensions for Feedback and Coaching". As described in the previous section, developers have different levels of knowledge and experiences, either influenced from previous projects or education. The architect should ideally know about the level of experiences and knowledge, e.g. what kind of patterns and technologies they know. In order to get the architecture accepted (see architecture acceptance), the architect should use patterns and technologies that developers already know - if it fits to the intended architecture goals and requirements. Then it is likely that architecture rules are less violated, since developers are familiar with those concepts and they know how to implement a specific decision. If concepts are new to them, the architect is responsible for coaching and supporting developers adequately (see Coaching and Supporting).

1.4 Revising the Architecture

In case the architect detects violations against the architecture, he always has to think about reasons that could have caused those violations. One important aspect could be that the architecture and the underlying decisions have to be updated. For example, an architectural solution could be too hard to implement: *"...for example they [the programmers] said that it was not possible to do it differently, because this and that was too complicated, so that we adapted the architecture rules in consequence and said it has to be different here actually, but otherwise it is too complicated."* (code: solution too complicated, Participant E). Consequently, the architect has to be prepared to compromise concerning his architecture solution and find an alternative solution that can both implemented by the developers and meets the requirements. Another reason could be that the assumptions on which the decisions were based were wrong: *"...[the architect] makes a lot of decisions based on assumptions. And maybe some of those assumptions were wrong. And the software developers [...] see that some of those decisions cannot be implemented as it was intended by the architect."* (code: gathering feedback by the developers, solution cannot be implemented, Participant G).

1.5 Assessing the Decisions' Implementation.

During the interviews we asked all participants the following question: *What are the specific steps when you inspect the source code in order to assess the imple-*

mentation of the architecture decisions? We developed the following categories of activities that are strongly interwoven.

- **Code Review.** We found that code review is a consent activity for assessing the decisions' implementation. One architect stated that this activity *"is similar to the comprehension process of a developer who is new in the team and tries to understand how the software systems works. But developers and architects have each different goals during this process. The developer mainly wants to implement new features, while the architect wants to check architecture conformance"* (participant **C**). Architects form a mental model of a software system and its relation to implementation based on architectural decisions. By doing this they have specific imagination about what they expect in the code: *"...a picture about if the components are appropriate, if the modules are implemented according to how it was intended..."* (Code: expectation about intended design, Participant C). In this process, software architects often ask questions about the observed software systems that entail exploration and navigation, such as who implemented this component and where is a specific feature, architectural pattern, design pattern, technology implemented or used. It is then evaluated informally if an implementation roughly represents this mental model. During this process, code analysis tools can be used as a source of information: *"...what you can do is, you run a code analysis tool and then you are looking at the spots that are interesting..."* (code: finding hotspots, results from code analysis tools as first impression, Participant K).
- **Repository Mining.** One expert uses review systems in order to review the implementation concerning architecture issues. In this way it is possible to investigate *what type of changes* were applied on a set of classes and especially *who* did the change. Moreover they can trace back how an architecture violation was introduced. They reproduce the steps of implementation and try to understand rationale and code-level decisions behind past changes. If an architect knows about the individual skills in a team, he can focus source code inspections on changes by developers with less skills, inexperienced, or new to a project. In this way he can raise his overall productivity as well as reducing the risks: *"...you know basically who works on which parts, this means if I know from experience that I have to have a closer look on what he or she has created then it is possible that I have to inspect each class [...] because he or she can create an unusual solution on the most unobtrusive parts"* (code: focused inspection based on individual skills of developer, Participant C).
- **Model-Code-Comparison.** We asked the participants how and if the architecture documentation and models are used in assessments. Some experts (**B**, **I**, **J**, **L**) use documented diagrams and models for conformance validation between implemented software system and architecture. For this, they use UMLclass diagrams, sequence diagrams or component diagrams and compare them with models extracted from the underlying implementation. The comparison is performed manually. For example they check if a message

exchange between components complies to the prescribed behavior by comparing UML sequence diagrams extracted from source code with prescribed sequence diagrams. (participant L).

– **Automatic Validation of Architectural Constraints** We also asked architects to which degree they formalize architectural aspects in order to allow a formal validation of a software architecture. Some experts also assess the layer pattern automatically by using dedicated tools such as Sonargraph. Additionally software architects define rules concerning such as naming conventions, thresholds for complexity metrics or other low-level rules that can be performed automatically by tools like Sonarqube or Checkstyle.