Architecture Enforcement Concerns and Activities - An Expert Study

Sandra Schröder(✉), Matthias Riebisch, and Mohamed Soliman

Department of Informatics, University of Hamburg,
Vogt-Koelln-Strasse 30, 22527 Hamburg, Germany
{schroeder,riebisch,soliman}@informatik.uni-hamburg.de

Abstract. Software architecture provides the high-level design of software systems with the most critical decisions. The source code of a system has to conform to the architectural decisions to guarantee the systems' success in terms of quality properties. Therefore, architects have to continuously ensure that architecture decisions are implemented correctly to prevent architecture erosion. This is the main goal of Architecture Enforcement. For an effective enforcement, architects have to be aware of the most important enforcement concerns and activities. Unfortunately, current state of the art does not provide a concrete structure on how the process of architecture enforcement is actually applied in industry. Therefore, we conducted an empirical study in order to gain insight in the industrial practice of architecture enforcement. For this, we interviewed 12 experienced software architects from different companies. As a result, we identified the most important concerns that software architects care about during architecture enforcement. Additionally, we investigated which activities architects usually apply in order to enforce those concerns.

Keywords: Software architecture · Architecture enforcement · Software architecture in industry · Empirical study

1 Introduction

Software architecture [1] builds the basis for the high-level design for a software system and provides the basis for its implementation. It defines the fundamental rules and guidelines that developers have to follow to ensure achieving quality attributes such as performance or security.

In software engineering literature and community, the role of the architect is widely discussed, especially in the context of agile development processes. For example, McBride [14] defined the role of the architect as being *"responsible for the design and technological decisions in the software development process"*. However, the software architect role [7,12] is not only limited to making architecture design decisions [10]. Additionally, the software architect is also responsible for *"sharing the results of the decision making with the stakeholders and the project team, and getting them accepted"* [23]. This task is called *Architecture Enforcement*.

B. Tekinerdogan et al. (Eds.): ECSA 2016, LNCS 9839, pp. 247–262, 2016.
DOI: 10.1007/978-3-319-48992-6_19

During implementation or maintenance activities, developers could intentionally or accidentally deviate from the prescribed architecture. This may result in the degradation of architectural quality. Consequently, the architect needs to proactively care about the adherence of the implementation to the chosen architecture decisions as a necessary part of the architecture enforcement task. For this, he needs to detect implementation decisions made by developers that indicate *architectural violations*, i.e. low-level decisions that do not follow the prescribed architectural constraints. The accumulation of architectural violations results in a phenomenon called *architecture erosion* [17,20].

Architecture enforcement faces several challenges such as the high effort required to assess the adherence of the implementation to architecture decisions, as well as the social and technical complexities in dealing with the development team. Facing these challenges requires methods and tools to support the architect during the architecture enforcement activities. To the best of our knowledge there is no detailed study about what are the necessary responsibilities and concerns related with architecture enforcement and about how architects actually monitor an implementation of architecture decisions.

As a starting point to achieve this goal, we conducted an empirical study with the purpose of understanding the process of architecture enforcement in industry. We interviewed 12 experienced architects from various companies. Based on their answers, we elaborated the most important concerns that are targeted by software architects during architecture enforcement, together with the related architects' activities and Best Practices. By defining the most important concerns, we provide the basis for focusing architecture enforcement on the essential aspects.

The following two research questions guided the empirical study:

- **RQ1: What are the concerns which architects consider during the enforcement process?** With this question we investigate which categories of concerns software architects usually consider important. This will give us further directions for our research activities in terms of detection and prioritization of architectural violations concerning decisions that are especially important for software practitioners.
- **RQ2: What are the activities performed by the architects in order to enforce and validate those concerns?** The answer to this question gives us a basis for developing appropriate approaches that best integrate with methods that are currently used in practice in order to gain acceptance by practitioners for new approaches.

2 Background and Related Work

In this section we present topics that are related with our study. We first present related work concerning architecture decision enforcement. After that we present related studies that investigate the concerns and activities of software architects and discuss to which degree those studies consider architecture enforcement.

2.1 Architecture Decision Enforcement

Zimmermann et al. propose a model-driven approach called *decision injection* in order to enforce the correct implementation of decisions in the source code [23]. Jansen et al. implemented a tool that allows the management of architectural decisions [11]. Among other things Jansen et al. emphasize that the tool should provide appropriate support for checking the implementation against architectural decisions. The tool implemented by the authors warns the architect or the developer if the team ignores a specific decision or introduces a violation against a decision. However, it is not clear what type of decision violations are detected with this tool. In order to control erosion of architecture decisions, traceability approaches as proposed by Mirakhorli and Cleland-Huang [15] can be applied. Their approach allows tracing architectural tactics to architecture-relevant pieces of code and warns the developer if he/she changes code of this significant part.

Software architecture conformance checking is another enforcement method. It allows the enforcement of the modular structure and dependencies of the software system. Well-known approaches encompass reflexion modeling [16], dependency structure matrices [18], design tests [2], or domain specific languages [3,21] - to name a few. However, static conformance checking methods are restricted to the modular structure and are not able to enforce arbitrary types of decisions, e.g. for checking constraints of architectural styles or the adherence to the separation of concerns principle.

2.2 Architects' Concerns and Activities During Enforcement

In [5] the authors conducted an empirical study about the architects' concerns. They present some interesting findings. For example they found that "People quality is as important as structure quality". This is also confirmed by our study, but we investigate in a more detailed fashion which are the actual dimensions of "people quality". Additionally, the authors' understanding about "architects' concerns" is a bit more general than ours. While they actually regard all the phases (i.e. architecture analysis, evaluation, architecture design, realization etc.) during the software engineering process as architects' concerns, we especially focus on the concerns that architects have corresponding to the architecture enforcement process.

The study of Caracciolo et al. [4] is similar to ours. They investigated how quality attributes are specified and validated by software architects and therefore also investigated what are important concerns for architects in terms of quality attributes. They also conducted expert interviews as part of their study. They identified several quality attributes that are important to software architects. Nevertheless, they solely concentrate on quality attributes and they do not especially focus on architectural enforcement and by which activities it is achieved.

3 Study Design

In our study we followed a qualitative research approach by applying a process with two main phases: Practitioners Interviews, and Literature Categories' Integration. The main purpose of the first phase is to explore the important aspects in the current state of the practice regarding architecture enforcement, while the second phase complements and relates the interviews' findings with existing concepts from the current state of the art. The two phases will be explained in the following sub-sections.

3.1 Phase 1: Practitioners Interviews

In order to collect data, we used expert interviews with open questions. Those interviews are an integral part of this type of research and are commonly used in order to generate new knowledge about a specific topic. An interview guide helped us to focus on the research questions, but also let the participants speak freely about their experiences. In this way, we could get as many examples as possible about architects' concerns and architectural rules and the methods architects use in the context of architecture enforcement. We followed the interviews with an inductive content analysis for the interview transcripts, from which we were able to derive our concepts.

Selection of Interview Participants. As participants of the study, we targeted experienced software architects from industry. Those architects come from different companies from Germany and Switzerland. All study participants had at least a master's degree or a similar qualification in computer science or related fields, e.g. electrical engineering or physics. In total, we interviewed 12 architects from 11 different companies. The interview participants are listed in Table 1. The

Table 1. List of study participants, their domain and their years of experience.

#	Domain	Role(s)	Exp. (years)
A	Enterprise (application, integration)	Software architect	>15
B	Enterprise	Software architect consulting	10–15
C	Enterprise	Software architect	>20
D	Logistic/enterprise	Software architect agile test engineer	10
E	Accounting/enterprise (migration)	Software architect section manager	10–15
F	Enterprise	Software architect lead developer	10–15
G	Enterprise/embedded	Software architect coach	10–15
H	Insurance/enterprise	Software architect project manager	5–10
I	Medical	Software architect software developer	5–10
J	Government/enterprise (application)	Software architect consulting	10
K	Logistic/enterprise	Software architect	5–10
L	Banking, control systems, enterprise	Software architect project manager	>20

professional experience of the participants ranged from 5 to over 20 years, with an average of 13 years. They worked in teams of size varying between 2 and 200 developers (team size not shown in the table). All of them work as a software architect or made significant practical experience in architectural design. Other participants are also responsible for project management tasks. The main criteria of choosing participants for this study was that architects should be closely involved with the implementation of the software architecture, for example in code reviews, and that they consider the maintenance of architectures with a long-term perspective. That means that architects should not solely participate in the modeling phase, but also in the implementation and maintenance phase. We did not focus on any specific domains, since we believe that architecture enforcement problem is a relevant and well-known problem in almost every domain.

Interview Guide Design and Conduction of the Interviews. The interview guide was designed for a semi-structured interview containing open questions that were chosen according to the research questions. The method helps to gain a possibly comprehensive overview of the state of the practice [9]. As we wanted to collect as much new knowledge as possible, we let the participants talk freely about their experiences concerning architectural enforcement.

The interview guide contains three parts. In the first part, we wanted to classify the experiences and background of the participants such as the domain in which they are working, years of experience, the team size and the development process (agile, waterfall etc.). The second part is related to the first research question. In this interview part we let the experts talk freely about their experiences concerning violations against decisions and important concerns. In the third part, we wanted to discover methods that are used by the architects in order to enforce architecture. The detailed interview guide is given in the supplementary and can be accessed via the paper's website[1]. When presenting our study results in the next sections, we are going to present some of the questions from the guide and the corresponding responses of the participants.

The interviews were conducted personally or via Skype by the same person and were recorded on agreements using a Dictaphone on the interviewers laptop or a call recorder for Skype conversations. The twelve interviews took between 40 and 90 min and 56 min on average. The interview guide directed the interviews, so that no important questions were missed concerning the research goals.

Data Analysis Phase. For further analyses, all interviews were transcribed word-by-word. After transcribing the interviews and checking them for correctness and completeness, we followed an inductive method for data analysis. Instead of defining codes before analyzing the interviews, we let the categories directly emerge from the data. For this, we first adapted Open Coding [19]. In this step phenomena in the data are identified and labeled using a code that

[1] http://swk-www.informatik.uni-hamburg.de/~schroeder/ECSA2016/.

summarizes the meaning of the data. During this process, emerging codes are compared with earlier ones in order to find similarities and maybe to merge similar codes. Then we compared codes with each other and aggregated them where possible to a higher level category. We used *AtlasTi*[2] in order to support the codification process.

3.2 Phase 2: Literature Categories' Integration

In this phase, we integrate our findings from Phase 1 with existing categories in the current state of the art. We analyzed existing related work (see Sect. 2), and identified categories related to architecture enforcement. The analysis has been done independently of the concepts derived from the interviews. We combined deductively the results of inductive content analysis (Phase 1) with the identified categories from existing literature. We used Mind Mapping in order to visualize the categories. By comparing the categories derived from both phases, we found that some of the categories derived by the literature review act as high level categories for inductively derived categories. On the other hand, some of the inductive categories could not be related to existing categories from the literature review. Section 4 presents our identified categories.

4 Results

Because of space limitations we discuss only the most interesting aspects in more detail. The complete discussion with the data used is provided as supplementary material and can be accessed through the paper's website (see footnote 1).

4.1 Enforcement Concerns

As Enforcement Concerns (Fig. 1) we summarized all aspects that have to be assessed by architects to ensure the correct implementation of decisions. Figure 1 gives an overview over the identified concerns from the interviews.

Macro and Micro Architecture Decisions. When talking about software architecture, it was interesting that experts differentiate basically between decisions in two different views, namely macro architecture and micro architecture [22]. Other terms like strategic or global (i.e. macro) and tactical or local (i.e. micro) views were used. The architect can decide which decisions are located in the macro architecture, and which decisions are left open for the development team. In this way the architect can decide how much freedom he gives teams in designing the micro architecture. The macro architecture represents the general idea (or "philosophy", the "spirit", the "big picture" or metaphor) of the system and its fundamental and most critical architectural decisions, e.g. on structures, components, data stores, communication style or architectural styles: *"... it is*

[2] http://atlasti.com/.

important how you regard it. For me there do exist basically two views about how software is built. First you have the global view [. . .] There I decide how I design my software, for example using Domain Oriented Design or SOA." (code: two different views of architecture, Participant D) and another participant reported: *". . . then we have the micro architecture, this is the architecture within each team. A team can decide for its own component for which it is responsible which libraries it wants to use."* (code: two different views of architecture, macro architecture, micro architecture; Participant K). Those two views define what architects basically consider as important for architecture enforcement in different ways. The architects report to be concerned with macro architecture issues and consider the micro architecture as developers' responsibility, except the coding style because of its relevance for maintainability: *". . . architecture is also present in a single code statement. Code styles belong to it. Or simple things like how do I define an interface. . . "* (code: micro architecture, Participant J).

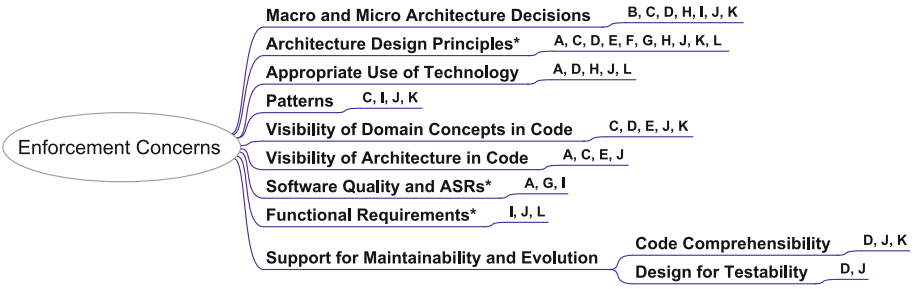

Fig. 1. Overview of the identified categories of Enforcement Concerns from the interviews and the corresponding participant. Concerns marked with an asterisk are not explained in detail in this paper but are available in the supplementary on the paper's website (see footnote 1).

Appropriate Use of Technology. Technology was also mentioned as an important concern. The architect may not check technologies used within a single component, but may for example enforce the technology for the communication style between several components. Since technologies like frameworks or libraries offer a lot of complex functionality, software architects also monitor the way how those technologies are used by developers. One architect stated that developers can easily violate important architecture rules due to this complexity. Some architects reported that developers often tend to use a lot of tools and technologies that are not necessary: *". . . aim for technologies is the biggest problem. And if you like to use those frameworks because they are providing gross things, but those gross things cannot be controlled. . . "* (code: aim for technologies, Participant J). They stated that software architecture is likely to erode where too much technology is used, because this part of code is hard too understand (see also "Support for Evolution and Maintenance"). This is why some experts emphasized it is important to control what kind of technologies are used.

Patterns. The architect may want to ensure that patterns are implemented accordingly. Patterns related with the macro architecture view have to be enforced and validated, patterns on the micro level are mostly considered as a developers' concern. But sometimes, pattern implementations are also checked by architects on the micro architecture level, e.g. in order to discover what types of design and architecture patterns are implemented and if they fit in the specific context: *"which patterns are used and in which context. Are they only used just because I have seen it in a book or because I wanted to try it or is it really reasonable at this place..."* (code: pattern suitability, Participant C).

Visibility of Domain Concepts in Code. Some experts emphasize a clear representation of domain concepts in the architecture, e.g. by expressing a mapping between them. For this, some architect strive to use a domain oriented terminology, that means using terms and names adapted from the business domain: *"...I like to be guided by the domain instead of using technical terms [...] both can work, but from my experience using domain oriented terms is easier to understand..."* (code: domain oriented terminology, Participant J). This additionally helps to talk with domain experts about the software design and to easier locate where changes have to be made in case of new or adapted requirements.

Visibility of Architecture in Code. Some architects consider it as important to make the architecture visible in the code, e.g. by using appropriate naming conventions and package structure: *"...therefore it is important that the architecture is recognizable in the source code. This is absolutely essential for the structure of the project."* (code: making architecture visible in the code, Participant J). This is helpful for tools like Sonargraph[3] that for example use naming conventions in order to highlight layers. It was also mentioned to be useful during code inspections in order to easily locate architecture decisions in code. This concern is similar to the idea using an architecturally-evident coding style suggested by Fairbanks [6].

Support for Evolution and Maintenance. A challenge in constructing long-lived system is to make decisions that support the software system's ability to easily be adapted to future changes, that is, we need support for evolution and maintenance during the entire software lifecycle.

– **Code Comprehensibility** was explicitly mentioned as a concern, on the basis that comprehensibility helps preventing architectural violations: *"if you strictly follow this approach then you have very readable code and normally readable Code - from my experience - tends to be stable that is conform concerning architecture and does not have any [architecture] violations..."* (code: code comprehensibility, code comprehensibility supports architecture conformance; Participant J).

[3] https://www.hello2morrow.com/products/sonargraph.

– **Design for Testability.** Another interesting aspect that might be surprising was that architects are strongly concerned with tests. Systems that cannot be properly tested, cannot be changed successfully since software modifications during maintenance and implementation activities may lead to errors. That is why testability is an important concern especially in the context of evolution and maintenance. Participants aim a high test coverage in order to avoid architectural violations: *"...in case there exist only a few tests, then it is likely people do not build it correctly. This leads to incomprehensible code and consequently to architectural violations."* (test coverage supports architecture conformance, Participant J). Tests are therefore an important concern for enforcement.

4.2 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 2 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. The complete mindmap with a mapping of codes and interview statements is provided in the supplementary (see footnote 1).

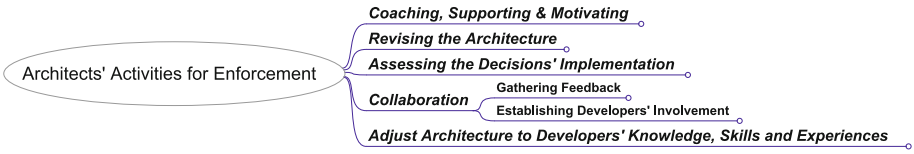

Fig. 2. Identified categories of enforcement activities.

We discovered the need for a distinction between situations with more or less equal architectural skills among the development team, as typical for agile processes, and situations with a leading role of the architect, frequently referred to as "architecture-first" approach. The latter is for example driven by limited skills of developers, or by stronger constraints and higher risks of the project. The first situation was mentioned by participant **B**, and the second one by participant **H**. In both situations, enforcement is necessary with different priorities, affecting the balance between the different dimensions of coaching and supporting (see below) on the one side, and assessment on the other side.

(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 goals 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 [the architecture] is about..."* (code: architect as a coach, Participant **G**). A combination of coaching and supporting can be done in several ways. 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 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 for getting the architecture accepted by the developers and that they [the developers] 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.

(2) 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 implementation 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 hot spots, 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 UML class 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.
- **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. We found that architects seem not to formalize to a great extent. Some experts formalize and evaluate the adherence to the layer pattern or general module dependency rules automatically by using 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. Other aspects of software architectures, for example other architectural patterns, are not formalized.

5 Discussion

In this section we discuss the results of the study and additionally important implications of these results for future approaches concerning architecture enforcement.

Social Dimensions of Architecture Enforcement. Based on our findings we can summarize that experienced practitioners understand enforcement as a supportive process for developers, instead of an authoritative, dictating or leadership-like process. They actively want to involve developers, and gather feedback on the architectural solutions. They strive a shared understanding about the software architecture. While motivating, encouraging and supporting developers in implementing the architecture, architects are open for revisions of their architecture solutions to minimize the risks of malfunction, misunderstandings or failures. Moreover, architects need to be anxious for encouraging acceptance and architecture awareness of the developers to decrease the probability of intentional and unintentional architecture violations. We propose that future approaches and tools should also respect the social dimension.

Developers' Flexibility and Responsibility. As stated in the introduction, an architecture violation can result from a piece of code that contradicts the rules defined by the software architecture. Nevertheless, software architecture is described on a higher level of abstraction, whereas developers are working on a lower level. Consequently, violations may occur that cannot be avoided due to this abstraction gap. That is why architects need to define the degree of flexibility and when developers are allowed to violate certain aspects of the architecture. It needs to be further investigated which criteria are needed to define this flexibility, e.g. based on the qualification of the developers.

Appropriate Formalization Support. Formalization in context of software engineering and especially software architecture is still not widely accepted, due to the expected extra effort as well as the lack of usability and appropriate tool support [13]. This can be also implied from the findings of our study. For example, we found that static dependencies are often used as a main criteria to define architectural constraints. Constraints concerning layer dependencies are validated regularly, whereas other types of architectural solutions, e.g. Model-View-Controller-Pattern, architectural tactics or communication styles, are not validated. One reason for this could be that there is a manifold and well-established tool support for static conformance checking, e.g. by Sonargraph or Structure101[4]. Therefore we can conclude that the availability of easy-to-use tool support strongly influences the acceptance of formalization approaches. A consequence might be that research should provide easy-to-use verification including tool support for other architectural aspects as well.

Guidance for Software Architects in Violation Detection and Prioritization. Based on the statements we can conclude that architects evaluate the severity of architectural violations rather intuitively. As we can imply from the results of our survey, architects often use metrics for example on static package dependencies in order to find hot spots that could give hints for crucial

[4] http://structure101.com/.

architectural violations. We suggest that a better guidance is needed for software architects in order to evaluate architecture violations and their severity. For this, a catalog could be developed that lists common and well-known architecture violations, similar to a pattern catalog. The catalog maps decisions to possible violations. Furthermore, corresponding detection and repair strategies can be recorded in the catalog. Architects can use the catalog during code reviews to focus on the implementation of the most important decisions. Moreover, appropriate guidance in metrics analysis and interpretation is still missing. We think that research does not necessarily needs to invent new metrics, but needs to investigate how those metrics can be appropriately used during the analysis of implemented architectures.

5.1 Limitations

Gasson et al. proposed the criteria confirmability, dependability, internal consistency, and transferability [8] in order to evaluate qualitative studies. As we described and captured the background of all the study participants we address transferability. Confirmability is addressed by repeatedly discussing and restructuring the categories in an iterative process. In order to address dependability we followed a research process (Sect. 3) and described all the steps that were conducted. In terms of internal consistency the statements and the corresponding codes were cross-checked by another researcher. As similar to other qualitative studies we have a limited number of participants. However since we wanted to generate new knowledge and not to evaluate or confirm existing knowledge we find that this limited number is acceptable.

Another limitation might be that we did not consider specific dimensions that could influence the experts' view on enforcement concerns. For example, skills and tasks of a software architect could influence his view about what are important concerns and activities in context of architecture enforcement. The domain a software application is developed for could also influence the importance of specific concern or even add further concerns to the list presented in this study. As we tried to get a general overview about architecture enforcement concerns, this was not the focus of our study and creating the correlation between specific dimensions and enforcement concerns is left for future work.

6 Conclusion and Future Work

An expert study with the goal of understanding architecture enforcement process in practice has been presented. To reach this goal, we gathered data by interviewing experienced software architects from several companies. Our contribution in this paper is the determination of the most important concerns, which are considered by architects, as well as the activities performed by them during the architecture enforcement process. In addition, our results show the important role of architects during system implementation, and the importance of the

relationship between software architect and development team, in order to properly implement software architecture decisions. The findings of this paper contribute towards methods for systematic and goal-oriented architecture enforcement. Thus, we are willing to extend our study to additionally explore some of the architecture enforcement concerns and activities in detail, and to determine which methods and tool support is required for each. Furthermore, we intend to merge our findings on essential enforcement concerns with architecture modeling approaches for agile development processes.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. Addison-Wesley Professional, Boston (2012)
2. Brunet, J., Serey, D., Figueiredo, J.: Structural conformance checking with design tests: an evaluation of usability and scalability. In: 27th IEEE International Conference on Software Maintenance (ICSM), pp. 143–152. IEEE Computer Society, Washington, DC, September 2011
3. Caracciolo, A., Lungu, M.F., Nierstrasz, O.: A unified approach to architecture conformance checking. In: 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 41–50. IEEE Computer Society, Washington, DC, May 2015
4. Caracciolo, A., Lungu, M.F., Nierstrasz, O.: How do software architects specify and validate quality requirements? In: Avgeriou, P., Zdun, U. (eds.) ECSA 2014. LNCS, vol. 8627, pp. 374–389. Springer, Switzerland (2014). doi:10.1007/978-3-319-09970-5_32
5. Christensen, H.B., Hansen, K.M., Schougaard, K.R.: An empirical study of software architects' concerns. In: 16th Asia-Pacific Software Engineering Conference, pp. 111–118. IEEE Computer Society, Washington, DC, December 2009
6. Fairbanks, G.: Just Enough Software Architecture: A Risk-Driven Approach. Marshall & Brainerd, Boulder (2010)
7. Fowler, M.: Who needs an architect? IEEE Softw. **20**(5), 11–13 (2003)
8. Gasson, S.: Rigor in grounded theory research: an interpretive perspective on generating theory from qualitative field studies. In: The Handbook of Information Systems Research, pp. 79–102 (2004)
9. Hove, S.E., Anda, B.: Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International Software Metrics Symposium (METRICS 2005), pp. 10–23, September 2005
10. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: 5th Working IEEE/IFIP Conference on Software Architecture, WICSA 2005, pp. 109–120. IEEE Computer Society, Washington, DC (2005)
11. Jansen, A., van der Ven, J., Avgeriou, P., Hammer, D.K.: Tool support for architectural decisions. In: Sixth Working IEEE/IFIP Conference on Software Architecture, WICSA 2007, p. 4. IEEE Computer Society, Washington, DC (2007)
12. Kruchten, P.: What do software architects really do? J. Syst. Softw. **81**(12), 2413–2416 (2008)
13. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2013)
14. McBride, M.R.: The software architect. Commun. ACM **50**(5), 75–81 (2007)

15. Mirakhorli, M., Cleland-Huang, J.: Detecting, tracing, and monitoring architectural tactics in code. IEEE Trans. Softw. Eng. **42**(3), 205–220 (2016)
16. Murphy, G.C., Notkin, D., Sullivan, K.: Software reflexion models: bridging the gap between source and high-level models. In: Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering, SIGSOFT 1995, pp. 18–28. ACM, New York (1995)
17. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes **17**(4), 40–52 (1992)
18. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 167–176. ACM, New York (2005)
19. Strauss, A., Corbin, J., et al.: Basics of Qualitative Research, vol. 15. Sage, Newbury Park (1990)
20. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory, and Practice. Wiley Publishing, Chichester (2009)
21. Terra, R., Valente, M.T.: A dependency constraint language to manage object-oriented software architectures. Softw. Pract. Exper. **39**(12), 1073–1094 (2009)
22. Vogel, O., Arnold, I., Chughtai, A., Kehrer, T.: Software Architecture: A Comprehensive Framework and Guide for Practitioners. Springer, Heidelberg (2011)
23. Zimmermann, O., Gschwind, T., Küster, J., Leymann, F., Schuster, N.: Reusable architectural decision models for enterprise application development. In: Overhage, S., Szyperski, C.A., Reussner, R., Stafford, J.A. (eds.) QoSA 2007. LNCS, vol. 4880, pp. 15–32. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77619-2_2