Automated Conformance Monitoring: Exploring the Path to Industrial Adoption

Andrea Caracciolo
SCG, University of Bern
3012 Bern, Switzerland
caracciolo@inf.unibe.ch

Mircea Lungu
University of Groningen
9747 Groningen, The Netherlands
m.f.lungu@rug.nl

Oskar Truffer
Studer + Raimann AG
3097 Liebefeld, Switzerland
ot@studer-raimann.ch

Kirill Levitin
bbv Software Services AG
8050 Zürich, Switzerland

Oscar Nierstrasz
SCG, University of Bern
3012 Bern, Switzerland
oscar@inf.unibe.ch

ABSTRACT

Architectural decisions can be interpreted as structural and behavioral constraints that must be enforced in order to guarantee overarching qualities in a system. Enforcing those constraints in a fully automated way is often challenging and not well supported by current tools. Practitioners are reluctant to invest in new solutions because of the difficulty of estimating the cost-effectiveness of a new tool, scarce resources allocated to quality related activities and a general lack of expertise in the domain. In this paper we investigate the dynamics involved in choosing and adopting a new automated conformance checking solution within an industrial context. We analyze a series of interviews identifying the most relevant criteria that affect user's decisions. We also report on multiple case studies, describing how those criteria were used to steer the actual evaluation, integration and operation of a prototypical solution. Our study shows the advantages of using a flexible and semi-formal language for describing relevant design concerns. We observe that assuming any pro-active behavior from end-users is often unrealistic and that developers need to be actively informed and motivated. We show that integration with established quality reporting infrastructure is a key strategy for minimizing the impact of a new tool on processes and roles.

Categories and Subject Descriptors

D.2.9 [**Management**]: Software quality assurance (SQA)

General Terms

Experimentation, Human Factors

Keywords

Conformance checking, process description, architectural rules

1. INTRODUCTION

As a system evolves, its architecture tends to drift away from its intended design, leading to a phenomenon called architecture erosion [1]. Architectural erosion leads to a gradual decrease in quality and to the accumulation of technical debt. To limit the impact of this phenomenon, practitioners resort to commercial analysis tools that provide various insights into the quality state of an application. Some of these tools are easy to set up but provide limited configurability. Other are highly customizable but hard to maintain or integrate with existing platforms and dashboards.

In this paper we aim to empirically understand the dynamics involved in choosing and adopting an automated conformance monitoring solution. We investigate the criteria that need to be taken into account when designing, adapting and/or deploying an automated quality monitoring solution within a company. We analyze a set of 14 interviews conducted with practitioners involved in architectural duties working in four distinct Swiss companies to derive a list of criteria that influence the decision of whether to adopt a conformance monitoring solution.

These criteria describe the different priorities that professionals take into account when evaluating the possibility of transitioning to a new quality assessment tool for conformance checking. Understanding which forces are involved and how to maximize one factor over another can make a difference between setting up a solution that everybody ignores and one that actually contributes to improving the quality of a system. To further understand how these criteria are valued in an industrial context, we planned 3 case studies. We developed our own prototype and tried to lead various professionals to adopt it within one of their project.

Thanks to the acquired experience we identified five phases that might be encountered on the path towards successful adoption of a conformance checking solution. Each phase is related to multiple judgement criteria that, based on our experience, are worth to be taken into account in order to maximize the chances of higher acceptance and impact of the conformance monitoring solution. Different criteria may be considered to be more or less important depending on various socio-technical aspects that characterize the target organization.

The paper is structured as follows: We motivate and describe our study (section 2-3); identify relevant decision factors (4); analyze how these influence the adoption of an automated conformance monitoring solution (5).

2. BACKGROUND

During an on-the field study [2], we observed that software architects are clearly aware of having little control over the implementation of their architecture. They typically express design decisions in terms of guidelines or specification constraints. Those are supposed to be read and periodically checked by developers but this hardly happens. As a project grows, quality checks become less frequent and more arbitrary. Knowing whether a certain architectural invariant is actually correctly reflected in the implementation often becomes a matter of trust.

To overcome this issue and properly assess the conformance of a software system with respect to a set of formerly defined architectural constraints, practitioners typically resort to specialized commercial tools (*e.g.*, SonarQube[1], SonarGraph[2]). Unfortunately the setup and maintenance costs related to such tools often outweigh the benefits that can be derived from their results. In fact, most of these tools suffer from three main limitations:

Poor maintainability: Tools are typically hard to set up and configure. Practitioners are often afraid of adopting new solutions because of the effort typically required for customization and integration. This effort is hard to estimate and to justify to higher management.

Narrow scope: The tools currently available on the market are very specialized and typically offer support for checking at most a couple of rule types. This means that practitioners are likely to need multiple tools, based on heterogeneous conceptual models and low chances of integration.

Low usability: Introducing a new solution typically involves training dedicated personnel, and producing explanatory documentation. The specification of rules is typically a non-collaborative process and which requires specific technical knowledge.

In our study, 4 out of 14 participants said they used tools in the past and later on decided to switch to manual inspections due to the excessive costs involved in the process. 6 participants did not even attempt to set up an off-the-shelf automatic solution. The reasons behind these choices could often be related to missing support from key stakeholders, lack of availability of an adequate tool and limitations in formalizing and testing the desired invariant. Participants reported that the value of conformance checking is often underestimated by decision makers. The "cost of misalignment is not perceived" (A) and the "return of investment [of a conformance checking solution] is not clearly seen" (A). "Developers do not care about non-functional requirements" (E). "Automatic validation [of architectural constraints] would be useful and will eventually probably be implemented, but is not feasible at the moment" (J) since "it's not a high priority and nobody would exactly know how to do it" (I). In other cases practitioners recognize that "there are things which can't be verified and decided automatically" (B). Sometimes, "verification is not heavily used because most of the concerns cannot be formalized" (E). In those cases "it's better to delegate to humans (*e.g.*, pair-programming, checklists)" (E). This might be due to mismatches between a specific architecture and an existing tool (*e.g.*, F could not use JDepend because the tool did not allow to identify components over package naming). Sometimes professionals recognize that "all their rules could be checked with static analysis" (J) and that "if a tool existed it would be very useful" (I). Unfortunately "the tools available today are not enough" (I) and nobody feels qualified to contribute a new one on his own.

After collecting a considerable number of opinions, we reached the conclusion that an automated conformance checking solution was typically desirable but considered to be too expensive to set in place. To solve this problem we started developing a solution that could at the same time address most of the identified requirements and minimize the drawbacks encountered by the interviewed practitioners. The prototype that we eventually developed reflected our understanding of this multitude of viewpoints and experiences. We developed an informal and intuitive model for describing the needs and expectations of our ideal user matured further as we started using the prototype in multiple case studies. During almost two years of collaboration, we started discovering new concerns, similar adoption patterns and unexpected requirements across different organizations. This led us to explicitly analyze and characterize all different factors that influence the way an automated conformance checking solution is evaluated by professional users. In the remainder of the paper, we try to reconstruct and analyze the process that needs to be followed in order to establish a quality assessment solution within an industrial organization.

3. STUDY DESIGN

In this paper we report on a blocked subject-project study [3]. Our goal is to observe different case studies and generalize the adoption process through common characteristics. To carry out our case studies we developed a prototypical solution which was refined in response to emerging requirements and feedback. The aim of this study is not to prove intrinsic properties of the tool, but rather classify the events that happen around its introduction process.

#	Role	Organization	Project	Team
A	CEO, architect	govt. (1)	enterprise	5
B	business manager	govt. (2)	enterprise	10-50
C	project manager	insurance (3)	enterprise	50+
D	architect	logistic (4)	enterprise	5
E	developer	logistic (4)	enterprise	5
F	CTO	banking (5)	enterprise	50+
G	architect	govt. (2)	enterprise	5-10
H	architect	govt. (2)	enterprise	10-50
I	architect	logistic (6)	enterprise	50+
J	developer	govt. (2)	dev. tool	5
K	architect	banking (5)	enterprise	5-10
L	architect	trans. (6)	control sys.	5-10
M	developer	banking (5)	code analysis	5-10
N	architect	banking (5)	dev. tool	5-10

Table 1: Interview participants. All have direct professional experience with architectural design.

Before planning our case studies, we interviewed 14 profes-

[1]`http://www.sonarqube.org`

[2]`https://www.hello2morrow.com/products/sonargraph`

sionals working on tasks related to software architecture (See Table 1). We recorded approximately 18 hours of conversation and collected several project documents (*e.g.*, architecture specification, developer guidelines). The qualitative data collected in this preliminary phase were analyzed using coding techniques [4]. Our goal at this stage was to identify important factors that may have a role in deciding on the selection, adoption and maintenance of a quality assurance tool for monitoring architectural conformance. As a result, we identified most of the criteria described in section 4.

In a subsequent phase, we developed a prototypical tool for monitoring architectural conformance. We designed a generic DSL, called Dictō, capable of expressing the architectural conformance criteria previously identified, and used it as a unifying front-end to a range of conformance checking tools [5].

Finally, we carried out five industrial case studies (see Table 2) with the intension of introducing our tool in the context of a live development project. In the first case study (*i.e.*, C1) we dealt with a consortium of vendors of an open-source PHP application called Ilias[3], an e-learning platform used internationally by millions of users. In C2 and C3 we approached two teams working for a company with one of the largest IT divisions in Switzerland. The team in C2 counts more than 30 developers working full-time on a 10+ year migration project (from Cobol to J2EE) of a B2B application used for managing orders and coordinate traffic. C3 was a smaller team responsible for the development of a J2EE basic framework employed in almost all the hundreds of projects running in the company. In C4 we approached a medium-sized consulting company with public sector contracts. In C5 we tried to establish a collaboration with a branch of one of the largest Swiss insurance companies.

#	**Organization** domain (employees)	**Project** tech. - size (team)	**Phase Reached**
C1	E-learning (12 vendors)	PHP - 1 M (25)	5
C2	Transportation (1.000+)	J2EE - 0.5 M (30+)	4
C3	Transportation (1.000+)	J2EE - 50 K (5)	1
C4	E-government (1.000+)	J2EE - 50 K (5)	1
C5	Insurance (1.000+)	J2EE - n/a (n/a)	1

Table 2: List of project teams participating to our case studies.

The study was conducted over the span of almost one year. All the companies were using some sort of commercial quality assurance tool. In C1, the collaboration was carried out through a special interest group (SIG) responsible for the proposal and design of new architectural concepts that could improve the overall quality of the system.

The case studies reached various stages of maturity (See section 5). In the first case study (C1), the tool was fully deployed and integrated into the production environment. In the second, we got it installed on single workstations and have it used in isolation. In all other case studies, the tool was officially presented to the team but was never fully adopted. In C3, we introduced the tool by showing some violations that we knew were relevant and partially already checked by the team. The inability to continue to further stages highly depended on a general lack of trust and low motivation. In C4, we interacted directly with higher management and immediately gained interest and willingness to collaborate. Unfortunately the company was subsequently acquired and restructured by a larger company. Our agreement, which was in the process of being formally defined by the legal department of one of their clients, was never made official. In C5, we similarly got in touch with higher management and presented our tool. After the initial meeting we tried to propose a project, but we didn't hear back from them.

[3]`http://www.ilias.de`

4. DECISION FACTORS

In this section we characterize the decision factors that play a decisive role in driving decisions when discussing over the adoption of a conformance monitoring solution. This classification is non-exhaustive and is entirely based on our direct experience. It includes factors that were discovered by analyzing 18 hours worth of interviews applying coding techniques (as described in section 3). Additional categories (*i.e.*, performance, accuracy, feature set, analytics support) were discovered during the cases studies.

4.1 Product

Cost – As in almost any industry, cost is often a primary concern. Embracing a new quality assurance solution typically entails new licensing costs and often requires skilled labor for adapting and maintaining the acquired solution. "The automation of conformance tests is expensive" (interviewee A) and "budget resources allocated to quality related tasks is limited" (G) in most of the cases. "Automatic checking [of architecturally relevant constraints] would be a big advantage" (G), but is not always a "high priority" (J), given that the customer and non-technical management often think that "the only important thing is that the product is delivered with the right functionalities" (G). This leads to situations where architectural conformance is checked "manually" (G), through generic tools (*e.g.*, SonarQube (H)) or "is evaluated [on the client side] on the basis of produced specification documents" (A). In general, software quality" is hard to quantify and management is always skeptic because it concerns issues out of its domain".

Usability – A usable software product should be easy to understand, learn and use [6]. "Software architects usually do not care about [implementation] details, they reason on a more abstract level" (K). They typically find themselves in the situation of taking decisions over architectural invariants which, in order to be checked, would require them to deal with variably complex tools. Some architects "prefer visual representations when it comes to understand architectural structure" (H). Some may even develop their own toolchain in order to check dependencies as specified in a well maintainable Excel spreadsheet (D). This shows that non-technical declarative specifications are normally preferred over tool-specific configurations.

Performance – Software efficiency can be measured in terms of time and resources consumed to complete a given task [6]. Analysis execution time is an important feature of a quality assessment tool. If architectural invariants need to be checked in near real-time (*e.g.*, at commit-time, on request), performance becomes key in providing a usable experience.

Accuracy – This quality is ensured if the software product is able to provide results with the needed degree of precision [6]. Analysis results need to be as correct and complete as possible. In our studies, we spent a considerable amount of time in validating the results of our tool. This process was crucial to increase the reliability of our solution and to proceed with our evaluation.

Feature set – Users select and compare software products according to the features they support [6]. In our case, the ability to describe and check multiple characteristics of a system was relevant to our collaborators. It influenced the requirements elicitation phase and was a clear discussion point when deciding on the adoption of the tool. Features are often used for comparison with other commercial products.

Integrability – Software products don't exist in isolation, but need to co-exist and interact with other, independently-developed products in the target environment [6]. Conformance checking solutions are typically introduced into established contexts and have to harmonize with pre-existing processes and tools. Practitioners expect that a new solution can non-intrusively enrich their experience by providing information when and where required. In fact, G advocated a conformance checking tool with "integration with word or enterprise architect" while J said that "it would be nice to have rules checked by an IntelliJ plugin". Some organizations have some kind of periodic reporting mechanism already in place (*e.g.*, email reports generated during the nightly build (G)) and would appreciate if those could extended instead of being replaced or replicated.

Proactiveness – To effectively enforce guidelines and guarantee architectural invariants, one needs to be proactively reminded of relevant anomalies and opportunities for improvement. "Constraints cannot be enforced if they are simply described in a document; they either need to be implemented in a framework or a verification tool must check them" (K). In fact, "people forget about rules or don't even know of their existence" (I). "An architectural specification is almost worthless" and "it's important to have continuous feedback (*e.g.*, checks integrated in the continuous integration server)" (E). Developers need to be reminded of their mistakes and tools should support them to prevent accidental violations.

4.2 Process

Transparency – Architects and developers are keen to have an accurate and up-to-date overview of particular aspects of their system. More transparency over quality related issues leads to "easier risk assessment and reduced hidden costs" (N). For example, "detecting dependency violation early during development [..] would be very helpful" (K). Delivering convenient reporting reduces the risk of incurring into technical debt and "exposes conceptual fallacies" (K).

Analytics support – Rules and analysis results may have managerial value (*e.g.*, estimate relevance/effort of tasks, validate new design concepts). In our studies we observed that violations can be used to show progress over maintenance or migration processes. Similarly, rule specifications can be used as a means to assess the feasibility and effort involved in realizing complex design changes.

4.3 User

Engagement – Another important factor that contributes to the success of a conformance checking solution is its level of acceptance. As we have seen, developers need to be properly motivated and encouraged to contribute. If they don't feel sufficiently involved in the process of defining and following common objectives, the solution is likely to be soon ignored and eventually abandoned.

5. ADOPTION PHASES

In this section we describe the phases that we encountered while introducing an actual prototypical solution in various IT companies. Each phase is linked to one or more decision factors that must be taken into consideration during that particular phase of the process. The single phases and the associated decision factors are described in Figure 1 and in the subsequent sections.

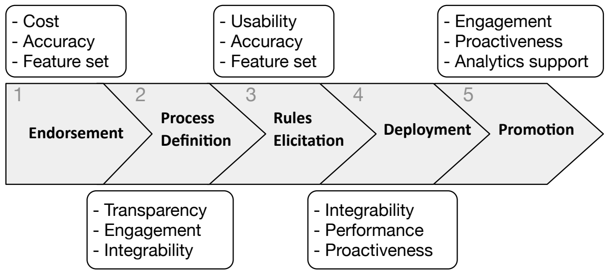

Figure 1: Adoption of an automated conformance monitoring solution: process phases and influencing decision factors.

5.1 Endorsement

Introducing a new technical solution within a company requires support from a competent and motivated person that understands the value of the proposed tool and the implications that this has on a chosen project. We typically approached this phase by setting up a introductory meeting with our contacts during which we presented the main features of our solution. The presentation also included some exemplifying rules, and an outlook over possible integration options with currently employed monitoring solutions.

In our experience we tried to approach different kind of users: developers and architects. In C1, our first case study, we interacted with a special interest group (SIG) consisting of 18 members founded to discuss architecturally relevant refactorings. This community was clearly aware of the benefits that can be derived from extra-functional maintenance activities. They have a distinctively proactive mindset, as their group has set the goal to propose new tasks aimed at improving the overall quality of the system. In this context we could easily convince them of the advantages of our prototype. Most of the discussion that followed concerned the possible political implication that the introduction of a new tool would have had on the organization.

In C2 we also interacted with a quality-aware proactive senior developer who believed in the benefits of quality assurance tools. He recognized the limits of the current monitoring environment and was willing to experiment with other solutions. In this case, the major concern was regarding the capabilities and the accuracy of the proposed solution. As a developer, he was very interested in uncovering new existing flaws and inconsistencies. Since his role did not entitle him to take any organizational decision, we had to follow a long procedure to define a pilot project. During this procedure, we had to negotiate the terms of our collaboration with the legal department of the company and subsequently discussed the project with the leader of the team our contact person was part of.

In C3 we interacted with the heads of a team responsible for a smaller, yet very strategical, project run within a larger organization. These 4 people maintained a framework that was used as a foundation by most of the 200 projects developed in the same company. Unfortunately, they were less inclined to consider the adoption of a new quality monitoring solution. We analyzed their code base and showed them new violations that they were not capable of finding with their current toolset. Despite that, they dismissed the idea of refactoring the uncovered flaws and preferred to maintain their code base in its current state. Their attitude towards code quality was more reactive. If somebody reported a major architectural violation, they would have looked into the problem and discussed a solution. Introducing an automated tool that supported this task was in contradiction with their approach. The effort invested in preventive maintenance had to be kept to the minimum.

In C4, we had the chance to discuss the adoption of the solution with two company-wide branch managers responsible for all major architectural decisions. Having the opportunity to discuss the subject with technically competent decision makers clearly facilitated our task. We could easily convince them of the utility of our prototype and we could quickly define a pilot project for testing it out. Getting in touch with these persons was less complicated because of the limited size of the company (100-150 employees). The advantage of encountering less resistance during the first encounter was lessened by the fact that both stakeholders were often very busy and tended to schedule subsequent meetings at longer time intervals (compared to the other case studies).

Relevant Decisional Factors

Cost: One of the primary concerns when discussing the adoption of the new technical solution was its cost. In our experience, especially when dealing with people having little or no decision power, we encountered appreciation for our choice of relying on open-source analyzers. On the other hand, the interest seemed to decrease when discussing analysis features that were offered by equivalent commercial tools already in use in the company. In C3, the architects were already using a rather expensive tool for checking dependency constraints. The possibility of integrating the results produced by the existing solution with the information produced by our tool seemed to be less appealing since money had already been invested in the competing solution. Also in C2 developers were using a commercial solution for checking dependencies. In that case, the person responsible for maintaining and operating the tool clearly admitted that the costs related to the use of that solution (*i.e.*, licensing, training) were clearly not proportional to the benefits offered. Given the tight budget typically allocated for quality analysis related tasks, it is important to define a price which is reasonably contained and proportional to the amount of distinguishing features offered.

Accuracy: The results produced by an automated quality monitoring solution should be sufficiently precise in order to be considered useful. In C2 and C3 we discovered that our tool found between 400% to 500% more violations than the competing tool currently in use within the project (C2: our tool found 7 illegal logical dependencies, SonarGraph only 2; C3: our tool found 18 dependency cycles, SonarQube only 3). This difference in accuracy is due to different analysis strategies employed by the different tools. In C2, accuracy played a significant role in deciding on the utility of the proposed solution. In C3, accuracy was overshadowed by the cost of setting up and integrating a new tool, justifying its existence to management and investing effort into dealing with the detected violations.

Feature Set: One of the differentiation factors that distinguishes our prototypical solution from competing alternatives is the support for a wider variety of constraint types. The possibility of specifying rules concerning multiple design facets in the same specification was a key deciding factor in C1. During the first meeting, participants immediately started to think about the type of invariants that could be useful to check in their project. Several people even proposed new types of rules that were not described in the initial presentation. Learning about the extensibility of our approach and the option of designing new custom analyses with a relatively modest effort was clearly one of the main arguments that convinced them to invest in the solution. In C5, one of the software architects participating in the initial meeting asked about the possibility of defining and checking cross-project invariants. Also in this case we can see how functionality is an important factor when discussing the high-level characteristics of a conformance monitoring solution.

5.2 Process Definition

If an organization agrees on supporting the introduction of the proposed solution, one must define how this can be done in concrete terms. In our case studies, we typically discussed various aspects. We analyzed how the solution was supposed to be integrated with the current infrastructure as well as the way future stakeholder would have to interact with it. Technical aspects are easily outlined and should be quickly sorted out by the service provider. Changes to the process require more careful analysis, since they may have a deeper impact on the performance of the organization. If a tool heavily interacts with current procedures and does not provide obvious advantages to its users, this tool will soon be neglected. Our solution provides contextual information that exposes anomalies in the developed code. By delivering our information through an existing information medium, we reduce the chance of altering established procedures and minimize the training costs.

In C2, we were asked to implement rules for expressing constraints described in an internal documental repository (*i.e.*, wiki website). All developers were asked at one point in time to read them, but few of them managed to keep them in mind and to periodically check whether they were updated. To reduce the overhead caused by a potentially useless activity, we decided to combine the existing documentation with executable rules that could be used to check the consistency of the developed system. This would have required them to delegate the task of defining new rules to the author of the guidelines. Rules written by this person would have, where necessary, required the intervention of a technical facilitator trained to integrate third party analyzers to support the checking of the defined rules. This technical facilitator would have initially been assisted by the original author of the tool. At the end of an initial training phase, the facilitator should have been capable of dealing with common integration scenarios. More complex cases, requiring deeper changes in the evaluation process, would have still required the knowledge of the original developers. The users ultimately inspecting the violations resulting from the validation of the rules, were expected to autonomously understand and react to the reported anomalies. Results were expected to be displayed through a pre-existing dashboard and handled through an integrated ticket system (See Section 5.3).

In C1, we discussed the target process together with the participants of the refactoring SIG. These people warned us of the risk of developing a solution that could not be fully accepted inside the community. In the past, another user deployed a continuous integration server that periodically built the core module of the project. This service was largely ignored because it was badly advertised to the community. Our solution had the potential of supporting vital quality assurance tasks but needed to be promoted in a convincing way. The tool had thus to be silently deployed and revealed as a complementary feature of a new, yet to be set up, continuous integration server. The plan consisted in assigning the responsibility of deciding on new rules to the SIG. New rules would have been announced during a bi-weekly physical meeting that involved representatives of the major vendors involved in the community. People participating in this meeting should be gradually sensitized towards non-functional issues and should have the right to veto the proposed rules. The discussion of rules should not require too much time, as the meeting is mostly designed to propose and discuss over functional requirements. Users should eventually be encouraged to look into the current violations by reporting the results produced by our tool. The details of each violation can be found in the continuous integration web front-end.

Relevant Decisional Factors

Transparency: In C1, users were positively inclined towards policies that supported and encouraged transparency. The fact of being part of an open community with a flat hierarchy made them more prone to engage in public assessment activities. Despite this, we decided to report on the introduction of new violations upon commit sending only private emails to the developers responsible for their introduction. During the bi-weekly meeting it was also expected that only the positive interventions (*i.e.*, violation removal) were mentioned publicly. Transparency is a good principle if counterbalanced by respect for the dignity of the involved people. In C2, developers were divided in 2 categories: internal (*i.e.*, developers contracted by the company) and external (*i.e.*, consultants hired through a third-party company). External developers can be easily dismissed and care more about maintaining their reputation. Exposing flaws that could be associated to them was immediately considered as a threat to their position. Based on this consideration, we decided to first test our solution within a smaller group of users exclusively composed of developers. This strategy would have helped to avoid unnecessary tensions and to gradually change the attitude of the team towards quality related issues.

Participation: In C2, we had the chance to interact with the developer responsible for maintaining a previously established dependency checking tool. This person emphasized that providing prompt feedback to the users is a key feature of any quality assessment tool. This must be true for users interested in the results of the analysis as much as for users interested in maintaining the rules checked by the used tool. In our case we aimed at integrating our tool with a dashboard system that was regularly refreshed after each build of the project. Rules defined for our tool would be defined though a dedicated web editor that supported the ability to interactively check their applicability to the target system. New tickets and email notifications would be created after the introduction of new violations. Pre-configurable ticket prioritization may also be used to direct the attention of the user. In C1, participation was seen as a key ingredient of a successful service. To reach a sufficient number of developers, we aimed at maximizing transparency and introducing new incentives for rewarding active developers. Users would be automatically listed in a leaderboard where the most contributors are ranked based on the number of violations they have removed. Another strategy to promote the involvement of developers consisted in presenting only the latest introduced violations while consulting the online report displayed within the continuous integration service. This reduces the chance that the user may feel overwhelmed and makes the effort required to eliminate the violations more easy to estimate. Remaining violations can still be browsed by expanding the view.

Integrability: Quality assessment tools are typically expected to be naturally streamlined into the process. In C2, the previously adopted dependency checking tool offered support for integrating the results into SonarQube. This feature was fundamental for making the violations visible to the developers. Separately produced reports cannot be regularly inspected without a clear incentive. SonarQube not only provides current measurements of the system but is also used to manage change tasks. This last feature makes it an essential platform of communication, that happens to be visited frequently by all the users involved in the development process. Our choice of creating new tickets for newly introduced violations reduces even more the distance of our solution from the center of attention of our target users. In C1, we chose to integrate our results inside TeamCity. This service already reports failed unit tests. By adding an additional view that shows the results produced by our tool, we emphasize the duality of a software system by displaying non-functional violations side-by-side with functional failures. Also in this case, having all information seamlessly integrated behind one coherent interface is likely to reduce resistance to adoption and increases productivity. The possibility to also link violations to impacted code elements also increases the convenience of the solution.

5.3 Rules Elicitation

Architects and developers have a very personal understanding of what constraints regulate the architecture of their system. This interpretation is typically vague and might not always fit the framework proposed by quality assessment tools. To bridge the gap that separates tools from design ideas, we first started by designing a language that could be used to replicate actually observed specification patterns. Putting language before functionality increased our chances of acceptance and played an important role in reducing the barriers to usage of our prototype.

The language that we used in our case studies, Dictō [5], is based on a simple model (Figure 2). Dictō can be used to describe properties and relations on and between entities defined in a system. Rules can be expressed in different modes (*e.g.*, TestMethods *must* have annotation '@Test' or *only* TestMethods *can* have annotation '@Test'). The DSL resulting from this model proved to be effective in eliminating any distraction related to unimportant technicalities linked to the the tools used for the analysis. The language was generic and abstract enough to accommodate a large variety of concrete rules. This encouraged our collaborators to explore very different requirements and led us to iteratively adapt and refine the language to meet their requests.

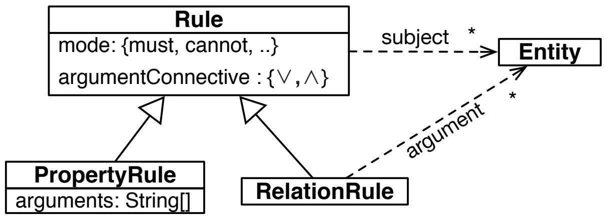

Figure 2: Language meta-model

In both case studies, rules were initially defined based on examples provided for illustration purposes. This helped us to gain familiarity with the general concepts governing our language. Later on we encouraged them to engage in the definition of more experimental rules. We suggested to examine current developer guidelines and to formalize rules that were (or that could have been) specified in such a context.

In C2, the user formalized a set of dependency constraints that were currently checked on his project through another tool (*i.e.*, SonarGraph). The specification of these constraints was previously regarded as technically demanding and was managed by one single developer, trained for the task. In its new translated form, the rules were finally under the control of our collaborator. This helped him to get a better understanding of the current architecture and to extend the current set of invariants with new rules concerning the module he was working on.

To enable the analysis of otherwise documented guidelines and constraints, we were required to extend our tool. As our collaborators realized that unsupported requirements could be easily integrated in the solution, they started to discuss additional ideas and to propose constraints that diverged more and more from the initially supported feature set. The proposal of unusual and unpractical rules represented a clear achievement, as it meant that our users could fully concentrate on what they wanted to check without considering how this could be done. Relying on a declarative language that can be easily understood by humans as well as tools, reduces the cost of formalizing and communicating knowledge and favors discussion.

In C1, other stakeholders started to propose rules related to their area of expertise or to recent tasks. Some managed to formulate syntactically correct rules by observing previous specifications. Others provided more prosaic descriptions. But eventually everybody could understand the rules written by others without any particular assistance. One of the points that contributed in making this possible was the introduction of inline Javadoc-like documentation in rule specifications. Thanks to this beneficial outcome, the refactoring SIG started to discuss new refactoring ideas together with enforcement policies. New rules were presented to superior decision organs and could be fully discussed without any previous instruction.

In both case studies we established short feedback loops during which we iteratively defined, analyzed and corrected each respective project ruleset. In each iteration we put care into manually validating the violations reported as a result of our rule analysis. By doing so, in C2, we discovered several infractions that were acknowledged as concrete issues and were ignored by other tools currently employed by the team (*e.g.*, SonarQube, SonarGraph). This increased the trust in our tool and reduced the distance between our solution and other comparable commercial products. In C1, the analysis of some reported violations brought the discovery of a previously unknown design anti-pattern (known as courier anti-pattern[4]). This in turn led to a new refactoring initiative and consequent rules that guarantee its implementation. In this case we observed how our solution could actually concretely support a feedback loop model for continuous quality improvement.

Relevant Decisional Factors

Usability: This was clearly one of the most deciding factors. In C1, stakeholders indirectly involved in the experiment were able to autonomously define new rules (*e.g.*, *IliasCodebase cannot depend on eval*; *IliasCodebase cannot depend on SuppressErrors*). The addition of documentation comments in the specification (used for describing the semantics of rules) reduced the gap between ordinary informal documentation and the more formal syntax of our language. Rules could be easily discussed without the presence of the original author. The overall friendliness of the language was partially undermined by the absence of comprehensive documentation and a proper rule evaluation environment (*i.e.*, sandbox environment). In C2, team members carefully avoided maintaining the rules specified for SonarGraph because of the poor usability of its configuration language. This shortcoming limited participation to a much smaller and less representative group of stakeholders.

Feature set: The capabilities of our prototype were soon questioned when users started to carefully consider their requirements. Questions like "is it possible to define.." or "how can I check if.." started to appear in C1 during physical and virtual discussions as soon as more users were involved in the process. Other users suggested rules (*e.g.*, "IliasCodeBase must be compatible with PHP5.3") that were syntactically correct but could hardly be checked using any kind of tool currently available on the market. Negotiating the scope of the offered service is a clear responsibility of the solution provider and will help him to focus the requirements elicitation process.

Accuracy: In C2 and C3, we defined rules concerning the presence of cycles and dependencies in the project. In both cases, our analysis reported violations that were completely ignored by popular commercial tools used by the organization (*e.g.*, SonarQube, SonarGraph). These tools have a good reputation and a solid user base but, without a means to establish the accuracy of their evaluation, it is hard to question the value of their application. Offering some reference measurements and generally assessing the accuracy of the produced results will increase trust in the proposed solution.

[4]`https://r.je/oop-courier-anti-pattern.html`

5.4 Deployment

Once defined, rules need to be verified automatically on a regular basis. To enable this behavior, we have to integrate our solution with an existing quality control system (*e.g.*, continuous integration server, dashboard). The challenges posed by this process may vary depending on the technical environment, the current development process and the general attitude of the team towards automated feedback mechanisms. In our case studies we opted for a simple setup (see Figure 3).

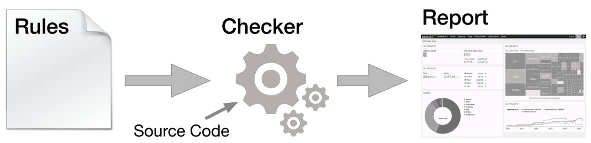

Figure 3: Automated conformance checking solution

In short, we analyze the rules defined during the previous phase through a set of pre-adapted off-the-shelf analyzers. The results are then integrated into a visual reporting system (*e.g.*, dashboard).

In C1, our collaborators decided to display the results as a custom view within TeamCity[5], a continuous integration system. Through this approach they succeed in addressing both functional (*i.e.*, unit test) and non-functional (*i.e.*, architecture conformance rules) aspects within a single interface. This allowed also for skeptical users, who should at least care about functional tests, to be exposed to architectural issues. In our implementation, we put care in not overwhelming the user by only showing the violations introduced in the latest build. All violations, including those introduced in the past, are still available for inspection by switching to a secondary view. This choice was mostly due to the fact that developers may be more interested in fixing issues introduced by recently contributed changes. Users might also be more motivated to address the issues if they see a smaller number of work items. An additional summary report was produced every two weeks and served as basis for discussion during the periodic physical meetings organized by all the organizations participating to the community.

In C2, the team used SonarQube as main issue tracker. We initially aimed at extending the current installation by automatically adding our violations as new issues. This strategy would have allowed for a very smooth transition with no changes in the workflow and full integration of our analysis. Unfortunately all SonarQube installations available in the organization stem from the same customized implementation. This made it almost impossible, given the size of the company and the intricacies of their governance, to extend the reach of our experiment to all the members of the team involved in the case study. As an alternative we decided to deploy our solution locally on one of the workstations used in the project. As a result we could simulate the productive environment and observe how our solution would have been used in the originally planned scenario.

Relevant Decisional Factors

Process Integration: One of the developers working in C2 and assigned to the maintenance of SonarGraph said that "integration is key". He himself had to integrate the results produced by his tool into SonarQube before and recognized that without that step nobody would have cared about the violations it reported. Extending a familiar and consistently used reporting instrument is essential to reach out to the end-user instead of requiring him to change his practice. In C1, we introduced our results by enhancing a long needed continuous integration application. The appeal of this new application granted sufficient support for our solution.

Performance: The time needed to complete our analysis played an important role during the deployment phase. In C1, we spent considerable effort to bring the time needed to check our rules from hours to minutes. To make this possible we had to optimize our prototype and improve some analysis techniques. Without this engineering effort, the solution would have been useless since developers expect feedback shortly after they commit. Also in C2, we had to deal with a large code base. This meant that multiple iterations were just dedicated to reduce analysis time. In both cases we had to make compromises over the precision of the analysis. This meant discussing with our collaborators over the minimum amount of information needed to describe violations in such a way that they could lead to a concrete action plan.

Proactivness: A developer in C1 emphasized that that "immediate feedback is important". There is evidence [7] that providing timely feedback on potentially degrading quality issues can significantly contribute to preventing architectural decay. In C2 we send email to contributors after each commit to expose violations that she introduced in her last change. Once again we try to reduce the burden on the user by reaching out to her.

5.5 Promotion

One of the main focuses of our solution, as stated by our collaborators, should be to actively contribute to improving code quality and to communicate the value of the implied effort to management. In C1 the reactions towards our tool were initially rather conservative, since the tool had no commercial history and was not believed to be reliable. By involving our collaborators in all the phases of the study and openly discussing possible improvements and limitations we slowly gained their trust. We gladly observed that the representatives of the various sub-communities could be involved in the process of discussing new rules. In fact, our tool-agnostic specification could be used to provide a concrete insight into the activities of the SIG as well as enabling management to take decisions on the concrete aspects that defined them. Our DSL became an effective tool for negotiating work items, expressing new concerns and assessing completed tasks. The support so far provided by management, was essential to establish a wide and legitimated dialogue. The fact that this assembly usually prioritizes the discussion over functional features and typically finds place under tight time constraints, does not guarantee that the discussion will continue to be kept at regular intervals. We assume that backing from a motivated sub-community (*i.e.*, SIG) needs to continue to be provided.

After observing the beneficial contribution of our solution, other members of the community volunteered to deploy our solution in their own environment. In fact, beside the main branch, accessed by the whole community, there were other client specific customizations maintained by single vendors. One of those vendors is currently working towards integrat-

[5]`https://www.jetbrains.com/teamcity/`

ing our solution in his own continuous integration server (*i.e.*, Jenkins[6]).

Relevant Decisional Factors

Engagement: In C1, we carried out a survey and found that developers were positively impressed by our solution. They observed that "it definitely leverages the discussion about architecture and separation of concerns". They commented that rules were readable and meaningful for the project. To further incentivize them, we also decided to create a leaderboard[7] that tracks who removed the most violations from the code base. This simple expedient, already exploited in other communities (*e.g.*, StackOverflow[8]), helped us increasing curiosity towards our analysis. One contributor said: "I searched very deliberately for violations within our modules and fixed them, on one hand to get our modules violation-free and on the other hand the leader board influenced me".

Proactiveness: In C1, every developer that we managed to contact through meetings or surveys stated that he is strongly in favor of architectural checks. Yet, when it comes to proactive actions, most people do not seem to have the time to work on quality related issues. This led us to focus on the proactive elements of our integration with the continuous integration server. We paid attention to setting up passive mechanisms that required minimal or no effort from the user. Analysis results are generated and communicated automatically and results are easily reachable in a context where the user would already be looking if she is interested in reports related to quality. Also the rule definition process should support proactive thinking. In C1, rules are defined to guarantee the correct implementation of new design ideas. By continuously inspecting and discussing the violations resulting from the analysis of those rules, developers grew their understanding of the system and even discovered new architectural anomalies that needed to be addressed. This unexpected virtuous circle once again confirmed the role of a well engineered conformance solution in high-level design related discussions.

Analytics Support: As previously mentioned, reports produced by our tool were also used to show progress over ongoing development tasks. Violations could show how far the current implementation was compared to a specific target architectural design resolution. This helped at the same time to strengthen the sense of control over non-functional aspects of the system as well as increasing the transparency of the development process.

6. DISCUSSION

Our case studies help to gain a practical insight into how an architectural conformance solution can be introduced in an industrial context. We ran several case studies and report on the strategies employed to gain full adoption of our tool. Despite our best efforts, we did not manage to cover all the phases outlined in our model (section 5) for every case study. We recognize that our study required a significant commitment by both involved parties and that most industrial organizations are skeptical towards innovative ideas. This explains why most case studies ended during the first phase. At the beginning it is important to gain the support of an active community or a relevant decision maker, and this did not happen in some of those cases. In general, we observed that political tensions that pre-existed within the organization had the biggest influence over the success of our project. Different social aspects (like fear of being discredited, of exposing inconvenient truths, of contradicting a superior or losing credibility in front of the colleagues) played an important role over the decisions that have been taken along the way.

Despite the partial completeness of some case studies, we still believe that our experience helped us gain a deeper understanding into the process undergone while introducing a generic architecture conformance monitoring solution. Our case studies show that the users involved in the adoption process might have different priorities but normally share the same concerns. The decision factors described in section 4 were partially identified before the beginning of the study and proved to be important points in our decision making process. Some of the decisions we took had observable effects which could be later on evaluated and discussed. The usability of our prototypical language, for example, appeared to be a relevant discussion point during the elicitation phase and showed its beneficial effects towards the end of the study, when more users started to participate to the definition of rules. Similarly, the integration of our solution within the existing infrastructure was considered relevant as we had to define the overall process required to sustain the solution and appeared to be a crucial aspect in later phases of the study (according to surveys and usage statistics). The decision factors that we analyzed in this paper should be considered for general guidance when defining a plausible adoption strategy. The analyzed criteria described in our work won't necessarily help in reaching a successful outcome but should contribute in reinforcing the assumptions that one might have towards the general process.

The phases described in section 5 and elaborated more in detail in the following sections have been inferred directly from our experience. We compared the different case studies and tried to factor our common activities and processes. As an outcome we obtained a sequence of replicable phases that can be used to break down our case studies. The resulting process model should be sufficiently general to be recognized in almost any context that entails the introduction of a new system for technical support. The main purpose of the model is to create a link between easily observable phases and the deciding factors identified to answer RQ1. The fact that those phases could be used to describe the case studies encountered in our experience provides empirical evidence that those phases can be used to establish a successful quality monitoring solution within an industrial organization.

7. RELATED WORK

Several authors report on the application of tools for checking architecture conformance in an industrial context.

Rosik *et al.* [8] describes an industrial application of a technique based on reflexion models [9]. The authors conclude that developers value their solution positively but violations are not fixed in a timely manner. Our studies were considerably more complex (*i.e.*, more developers, legacy code, more type of rules) and partially confirmed the obser-

[6]`https://jenkins-ci.org`

[7]`http://ci.ilias.de/DictoStats`

[8]`http://stackoverflow.com`

vations reported by Rosik *et al.*. We observed that violations are typically resolved as long that the effort involved is contained and adequate incentives are provided. Herold *et al.* [10] elaborate on the technical details of a rule-based architecture conformance checking tool used in an industrial case study. The approach is conceptually similar to ours but restricted to a particular type of rule (*i.e.*, dependency constraints). The studies show the importance of process integration and the use of a simple and user-friendly formalism to increase maintainability. Ganea *et al.* [11] evaluate their quality assurance tool by defining a non-comparative experiment involving industrial users. The authors recognize the importance of reducing the number of false positives, seamless integration, unobtrusive feedback, performance and user feedback. All conclusions are drawn from a controlled experiment, but still provide a comprehensive picture of what factors may influence the acceptance of a quality assessment tool. In general all mentioned authors tend to focus mostly on describing their solution instead of analyzing the process followed to introduce it within a specific organization.

Other studies explicitly analyze the impact of introducing a quality assurance tool in an industrial context. Sadowski *et al.* [12] describe a program analysis platform that integrates multiple lint tools and exports all detected issues to a review system. The authors emphasize the importance of actionable results and workflow integrability. Users are particularly sensitive to false positives and value the ability to share their configuration with the other members of their team. Despite the slightly different nature of the tools discussed in their study, Sadowski *et al.* reach similar conclusions as those reported in this paper. Johnson *et al.* [13] interview 20 practitioners that use static analysis tools on a regular basis. The conclusions drawn from this study are similar to those reported by Sadowski *et al.*. Participants criticize the poor usability of the tools (*e.g.*, result navigation, result understandability, settings sharing, customizability) and stress the importance of quick feedback.

8. CONCLUSION

In this paper we describe how an automated conformance monitoring solution can be adopted in the context of an industrial project. We describe this process in terms of its composing phases and the deciding factors that influence its course. Our aim is to offer a comprehensive overview of the forces involved in such a delicate course of events.

Our experience shows that a quality assurance solution should be above all customizable and usable. Developers are typically not encouraged to react to quality-related issues and need to be properly informed and motivated. This can be achieved through continuous automated analysis, non-overwhelming reporting and various types of incentives (*e.g.*, reputation points). Architectural inconsistencies need to be communicated as small and easily manageable tasks. Resistance to change can be overcome by integrating the new solution with a pre-established quality control system.

Through our experience we hope to provide guidance to professionals and academics that intend to introduce a similar solution in a company.

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Assessment" (SNSF project Np. 200020-144126/1, Jan 1, 2013 - Dec. 30, 2015).

9. REFERENCES

[1] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, pp. 40–52, Oct. 1992.

[2] A. Caracciolo, M. Lungu, and O. Nierstrasz, "How do software architects specify and validate quality requirements?," in *European Conference on Software Architecture (ECSA)*, vol. 8627 of *Lecture Notes in Computer Science*, pp. 374–389, Springer Berlin Heidelberg, Aug. 2014.

[3] B. Boehm, H. D. Rombach, and M. V. Zelkowitz, eds., *Foundations of Empirical Software Engineering*. Berlin, Germany: Springer-Verlag, 2005.

[4] M. B. Miles and M. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook(2nd Edition)*. Sage Publications, Inc, 2nd ed., 1994.

[5] A. Caracciolo, M. Lungu, and O. Nierstrasz, "A unified approach to architecture conformance checking," in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 41–50, ACM Press, May 2015.

[6] "ISO/IEC 9126-1:2001 Software engineering – Product quality," 2001.

[7] J. Knodel, D. Muthig, and D. Rost, "Constructive architecture compliance checking – an experiment on support by live feedback," in *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM 2008)*, pp. 287–296, 2008.

[8] J. Rosik, A. Le Gear, J. Buckley, and M. Ali Babar, "An industrial case study of architecture conformance," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, (New York, NY, USA), pp. 80–89, ACM, 2008.

[9] G. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between source and high-level models," in *Proceedings of SIGSOFT '95, Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 18–28, ACM Press, 1995.

[10] S. Herold, M. Mair, A. Rausch, and I. Schindler, "Checking conformance with reference architectures: A case study," in *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, pp. 71–80, Sept. 2013.

[11] G. Ganea, I. Verebi, and R. Marinescu, "Continuous quality assessment with incode," *Science of Computer Programming*, 2015.

[12] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, (Piscataway, NJ, USA), pp. 598–608, IEEE Press, 2015.

[13] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pp. 672–681, IEEE Press, 2013.