On the Evaluation of a DSL for Architectural Consistency Checking

Andrea Caracciolo
SCG, University of Bern, 3012 Bern, Switzerland
http://scg.unibe.ch

Abstract

Software architecture erodes over time and needs to be constantly monitored to be kept consistent with its original intended design. Consistency is rarely monitored using automated techniques. The cost associated to such an activity is typically not considered proportional to its benefits.

To improve this situation, we propose Dictō, a uniform DSL for specifying architectural invariants. This language is designed to reduce the cost of consistency checking by offering a framework in which existing validation tools can be matched to newly-defined language constructs.

In this paper we discuss how such a DSL can be qualitatively and qualitatively evaluated in practice.

1 Architecture erosion

Software architecture is the result of a design effort aimed at ensuring a certain set of quality attributes. The decisions deriving from such an effort are typically constraints on various aspects of an implementation. These may include invariants overs structural (*e.g.*, naming conventions, dependencies) or behavioral (*e.g.*, communication) aspects of the system. Even though explicitly specified, these constraints are rarely checked automatically. In a previous study [Cara14], we show that only 40% of software architects formally specify and automatically test such constraints. This situation can be explained by analyzing the limitations associated with the tools currently available on the market. In fact, tools suffer from the following drawbacks:

- Scattered Functionality: Most tools are specialized in a narrow domain and are typically capable of evaluating only limited types of constraints.
- Specification Language Heterogeneity: Current tools are based on different specification languages that differ in both syntax and semantics.
- Specification Language Understandability: Tools often force the user to express constraints in a typically overly technical and verbose form.

To improve this situation we built Dictō, a DSL (Domain Specific Language) for the specification of architectural constraints.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

2 Dictō

Dictō is a language that aims at supporting software architects in formalizing and testing prescriptive assertions on functional and non-functional aspects of a software system [Cara15]. Instead of dealing with multiple tool-specific formalisms, one can define several types of architectural constraints using one uniform, highly-readable, formal language (as shown in the example below).

```
Test = Package with name:"com.app.Test"
View = Package with name:"com.app.View"
Model = Package with name:"com.app.Model"
Controller = Package with name:"com.app.Controller"

Test, View can only depend on Model, Controller
Model cannot contain cycles
only Test can contain dead methods
```

The language is based on a plugin framework in which new language constructs can be defined along with the logic required to validate the concepts they are expressing. Developers can create a new rule template (*e.g.*, *Method* must be executed in $<$ *Integer* ms) by implementing a set of pre-defined data transformers for a given target tool. These transformers must be capable of (1) generating an input specification that is consistent with the user specified invariants; (2) interpreting the results produced by the tool. The advantages of this language are the following:

- Separation of concerns: conceptual design (specification of constraints) and technical effort (rule evaluation) are managed separately.
- Support for communication: a specification encoding valuable architectural knowledge should be accessible and readable by multiple parties, including stakeholders that do not have the skills necessary to operate the tool used to verify the expressed constraints.
- Reduce overall specification and validation effort: the time and effort involved in the process of writing and testing rules should be minimized as much as possible. To achieve this goal, we built a solution that offers reusable validation functionality, a uniform language syntax and a simple extension mechanism for the integration of new tools.

3 Evaluation

In this paper we discuss how Dictō, a DSL designed for encoding architectural constraints, can be exhaustively evaluated in an industrial context. Based on preliminary considerations, we would like to analyze the following properties by answering the following questions:

Impact on the product : Does the solution improve code quality in any measurable way?

Impact on developers : Does the solution increase architectural awareness?

Impact on process : How is the solution integrated into the process? Are there any conflicts with pre-existing practices?

Specification Expressivity : Does Dictō fit the specification needs of practitioners?

Specification Usability : Are constraints easy to read and write? Are the results actionable?

Ease of adoption : Is the effort required to support new requirements sustainable and cost-effective?

To answer these questions, we are currently running 3 different case studies with 3 different industrial partners. Our approach is to analyze their needs, encode their constraints using our DSL and analyze the effect Dictō has on the project. We plan to analyze the initial stages of the integration process, taking note on unsatisfiable expectations and similarities with pre-existent solutions. We will ask the study subjects to customize the rules

and adapt them to emerging requirements. We also plan to involve stakeholders with different background (*e.g.*, developers, analysts, technical managers) and ask them to explain their understanding of the formalized rules. The general impact of the solution will also be measured by observing if reported violations are actually taken into consideration and fixed, and if this has some other beneficial effect on other quality indicators (*e.g.*, coupling, test coverage, bug resolution time). Each constraint expressed in Dictō will be also compared with pre-existing coding guidelines, and implicitly known rules. We also aim at reporting on the potential reuse of developed tool adaptors (is a standard set of analyzers sufficient to express and test quality concerns across organizations?). By collaborating with multiple industrial partners, we hope to see how our solution is accepted in different contexts.

4 Conclusion

In this paper we discuss several best practices for the evaluation of a quality assessment tool in an industrial context. Our goal is to outline a concrete strategy for identifying the practical limitations of a prototypical solution. This is done by measuring the impact and applicability of a tool together with practitioners. The results of such an effort should help reaching a deeper understanding of the domain and generate new ideas for research.

5 Acknowledgements

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).

References

[Cara14] A. Caracciolo, M. F. 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.

[Cara15] A. Caracciolo, M. F. Lungu, and O. Nierstrasz, "A unified approach to architecture conformance checking" in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, ACM Press, 2015.