How Strict is Your Architecture?

Moritz Beller
Technische Universität München,
Institut für Informatik, D-85748 Garching
beller@in.tum.de

Elmar Jürgens
CQSE GmbH
Lichtenbergstr. 8, D-85748 Garching
juergens@cqse.eu

Abstract

Software architecture has an important impact on maintainability. One of its key functions is to restrict dependencies between system components. However, there are few objective criteria to quantify how well a given architecture does so. In this paper, we propose an objective measure for architecture strictness and report the findings of our case study on architecture strictness with nine real-world software systems.

1 Introduction

Every software system has an architecture [5]. It comprises components, which structure the system into smaller, coherent parts, and policies, which define the allowed and denied dependencies between the components.

A system architect has to design and maintain the architecture specification. During the evolution of a software system, the architecture specification is subject to change [4].

Architecture conformance analysis detects deviations between the architecture specification and the architecture present in the source code. Violations of an intended architecture abound in practice. To remove the violations, the system architect often has to change the architecture specification.

As for other software artifacts, one promising way to perform quality assurance is to review these changes. However, it is difficult to rate the impact of a change, even for an experienced architect with profound knowledge about the system.

As a consequence, during a review it often remains unclear how strong the ability of the architecture to prevent certain dependencies is affected by a change. Thus, we need a tool which allows us to capture this aspect of the architecture design. To support the quality assessor, we propose the architecture strictness ρ and evaluate its relevance in practice in a case study with nine real-world industrial systems answering the following research questions.

2 Architecture Strictness

Architecture strictness ρ is the probability that for two arbitrarily chosen types a and b ($a \neq b$), type a may not access type b in the underlying architecture.[1]

A value of $\rho = 0$ characterises a lax system, where all types can access each other. $\rho = 1$ characterises a restrictive system, in which every type can only access itself.

A type pair is a tuple $< t_1, t_2 >$ of two types. If t_1 may depend on t_2 according to the architecture specification, then the pair is an allowed pair. To calculate ρ on an architecture, we use the formula

$$\begin{aligned} \rho &= \frac{\text{Number of Denied Pairs}}{\text{Number of All Pairs}} \\ &= 1 - \frac{\text{Number of Allowed Pairs}}{\text{Number of Possible Pairs}}. \end{aligned} \quad (1)$$

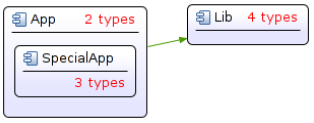

Figure 1: Example ConQAT architecture with one allowed policy from App to Lib. The number of classes mapped to the component is denoted behind the component's name.

As an example, we calculate ρ on figure 1 by summarising all allowed and then all disallowed type pairs.

Reason	Allowed type pairs
Types in same component	$2 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 = 20$
Policy from *App* to *Lib*	$(2 + 3) \cdot 4 = 20$
Sub or super type (*SpecialApp* and *App*)	$(2 \cdot 3) \cdot 2 = 12$
Total	52

Denied pairs are from *Lib* to *App*, which are $(2 + 3) \cdot 4 = 20$ type pairs. Thus, the number of all possible pairs is $52 + 20 = 72$. Using equation 1, we receive $\rho = 1 - \frac{52}{72} \approx 0.278$.

3 Study Objects and Procedure

We performed a case study on nine industrial systems developed and maintained by Munich Re. Munich Re is an international reinsurance company with over 40,000 employees worldwide. The study objects are nine heterogeneous C#-written projects (cf. table 1). A prerequisite for projects at Munich Re is to manage architectural knowledge with a ConQAT architecture specification.

We implemented an automatic calculation of the architecture strictness with the open source quality assessment toolkit ConQAT[2]. The details of ConQAT architecture specifications are described in [2, 3]. The

[1] In the context of C# and Java, types are classes.

[2] Available from http://www.conqat.org

System Characteristics			System Metrics		Strictness Analysis	
System Name	#Components	#Types	LoC	CC	ρ	Change Range
System A	41	1,732	322,870	27.8%	0.673	$[-0.077; 0.033] = 11.0\%$
System B	34	499	56,247	12.4%	0.555	$[-0.064; 0.064] = 12.8\%$
System C	37	660	53,155	27.2%	0.799	$[-0.069; 0.015] = 8.4\%$
System D	39	1,072	160,916	6.2%	0.522	$[-0.131; 0.116] = 24.7\%$
System F	98	4,479	328,975	7.2%	0.611	$[-0.075; 0.075] = 15.0\%$
System H	31	2,034	526,513	20.0%	0.462	$[-0.116; 0.116] = 23.2\%$
System I	71	8,436	1,417,578	21.7%	0.724	$[-0.124; 0.024] = 14.8\%$

Table 1: Excerpt of the nine study objects System A-I.

detailed results to the research questions are given in table 1.

4 Q1: How does ρ differ between systems?

We found that at a range of 0.462 to 0.799 ρ is well-spread.

5 Q2: How do low-restrictive architectures differ from high-restrictive ones?

We manually examined the architecture of the systems with the lowest strictness (Systems D and H), and compared them to System C with the highest strictness in the study. Systems D and H have two and three top-level components, while System C has seven top-level components. Allowed accesses between the top-level components in Systems D and H lower their architecture strictness, but are absent in System C.

The results suggest that for a higher ρ the architecture ought to have several unconnected top-level components of equal size (like System C).

6 Q3: Does ρ correlate with established metrics?

We observed no correlation between LoC (Lines of Code, including blank lines and comments) and strictness: Pearson's $r = 0.18$. However, there is a moderate correlation between CC (Clone Coverage, the probability that a source code statement is covered by at least one clone, an indicator of duplication and quality defects in the code) and strictness at $r = 0.64$.

Together with Q1 this indicates that strictness is largely independent of the system size, allowing us to compare different-sized systems. Generally, a high strictness equals few allowed dependencies in relation to the possible dependencies in the system (cf. equation 1). However, allowing only few dependencies to other types could hinder the principle of code reuse. Therefore, in a restrictive systems, code duplication could be a consequence. This might be a reason why we observed a moderate correlation between strictness and CC.

7 Q4: How does the range of atomic changes to ρ differ between systems?

An atomic change is the removal or addition of an arbitrary policy in an architecture. The variables change range and ρ show a strong negative correlation $r = -0.74$. This supports the interpretation of Q2: High ρ architectures are designed so that a modification to the architecture does not change its ρ so much: An atomic change between small to medium components never has such an effect on ρ as between large components (like Systems H and D).

We found that change range in ρ varies. Moreover, it is negatively correlated with ρ.

8 Related Work

Bouwers et al. [1] propose the Component Balance Metric to capture the analyzability of an architecture. The metric is a function of the number of top-level components and their relative sizes.

Wermelinger et al. [6] perform a case study on the Eclipse SDK's architectural changes from version 1.0 to 3.5.1. They compute a series of related metrics, among which are the coupling and cohesion metrics.

While these are computationally similar architectural metrics, to the best of our knowledge there are no measures that focus on an explicit reference architecture and measure its ability to prohibit dependencies.

9 Conclusion

Based on the results of our case study, we found that strictness differs among systems. We explained outliers by manual inspection and extracted some theses for system designs that lead to strict architecture specifications. Other known metrics do not encompass the architecture strictness. The topic remains an interesting field for future research: We will make a larger case study with (open source) non-C# projects from a different domain. A more elaborate study on code duplication and its implications on the architecture strictness could clarify the correlation.

References

[1] E. Bouwers, J.P. Correia, A. van Deursen, and J. Visser. Quantifying the Analyzability of Software Architectures. 2011.

[2] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. Mas y Parareda, and M. Pizka. Tool Support for Continuous Quality Control. *IEEE Software*, 25(5):60–67, 2008.

[3] Florian Deissenboeck, Lars Heinemann, Benjamin Hummel, and Elmar Juergens. Flexible Architecture Conformance Assessment with ConQAT. In *ICSE'10*, 2010.

[4] M. Feilkas, D. Ratiu, and E. Jurgens. The loss of architectural knowledge during system evolution: An industrial case study. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*, pages 188–197. IEEE, 2009.

[5] M.W. Maier, D. Emery, and R. Hilliard. ANSI/IEEE 1471. *Systems Engineering*, 7(3):257–270, 2004.

[6] M. Wermelinger, Y. Yu, A. Lozano, and A. Capiluppi. Assessing architectural evolution: A case study. *Empirical Software Engineering*, pages 1–44, 2011.