Continuous Architecture Evaluation

Eric Bouwers and Joost Visser
Software Improvement Group, Amsterdam, The Netherlands
E-mail {e.bouwers,j.visser}@sig.eu

Most software systems start out with a designed architecture which documents the important design decisions. These decisions include, but are not limited to, responsibilities of each of the main components and the way in which the components interact. Ideally, the implemented architecture of the system corresponds exactly with the design, only those components which are described are implemented and all components interact through pre-defined communication channels. Unfortunately, in practice we often see that the original design is not reflected in the implementation. Common deviations are more (or less) components, undefined dependencies between components and components implementing unexpected functionality.

There are many reasons for these discrepancies to occur. For example, the choice for a technology can lead to an unwanted implementation because the chosen technology does not allow a particular construct. Deviation can also arise because of process-related issues, for example because new functionality is added to the system without taking into account the design. Lastly, it could simply be the case that there is an error in the designed architecture.

In these situations the development team can decide to solve the issue by means of a quick fix outside of the designed architecture to meet a deadline. Even though everybody knows that this type of fixes should be temporarily, the priority of solving these 'cosmetic' issues is low. After all, the system is working correctly, so why change something which is not broken?

The examples illustrate legitimate reasons for deviating from the designed architecture. By involving both the development team as well as the architects in an evaluation of the implemented architecture both the implementation and the design can evolve together. Many methods are available for such evaluations, varying greatly in depth, scope and required resources. The end-result of such evaluations are, amongst others, an up-to-date overview of the implemented architecture and the corresponding design.

But when should such an evaluation take place? Depending on the amount of resources required the evaluation can take place once or twice during a project, or periodically (for example every month). Unfortunately, in between the evaluations issues can still arise, which still leads to deviations between the design and the implementation. The later these deviations are discovered the more costly it is to fix them.

A solution to these problems is to continuously monitor important aspects of the implemented architecture. This can be done automatically by the means of software metrics. A basic metric, such as the number of components, is easy to calculate after each change and can serve as a trigger to perform a quick manual evaluation to see whether the change fits into the current design. If this is not the case a more detailed evaluation can be performed, potentially leading to a full-scale architecture evaluation.

Basic metrics (number modules, number of connections) are easy to measure and provide relevant information. However, just examining these two metrics does not make it possible to detect all types of changes, for example when a single components is implementing too much of the overall functionality.

In our current research project we are extending the set of available architecture metrics by new metrics which are related to quality aspects as defined in ISO 9126. More specifically, we have designed and validated two new metrics which quantify the Analyzability and the Stability of an implemented software architecture.

The first metric we designed is called "Component Balance" [1]. This metric takes into account the number of components as well as the relative sizes of the components. Due to the combination of these two properties, both systems with a large number of components (or just a few components) as well as systems in which one or two components contain most of the functionality of the system receive a low score. We validated this metric against the intuition and opinion of experts in the field of software quality assessments by means of interviews and a case study. The overall result is a metric which is easy to explain, can be measured automatically and can therefore be used as a signaling mechanism for either light-weight or more involved architecture evaluations.

The other concept we introduced is the "dependency profile" [2]. For this profile each module (i.e. source-file or class) is placed inside one of four categories; hidden inside a component, being part of the requires interface of a component, being part of the provides interfaces of a component, or being part of both interfaces. The summation of all sizes of the modules inside a category provides a system-level quantification of the encapsulation of a software system. This metric has been validated by an empirical experiment in which the changes which occurred to a system are correlated to the values of each of the four categories. The main conclusions of the experiment is that with more code encapsulated within the components of a system more of the changes remain localized to a single component.

Both of the metrics have shown to be useful in isolation. We are taking the next step by determining how these metrics can best be combined in order to reach a well-balanced evaluation of an implemented architecture. In order to answer the question when a more elaborate evaluation should take place we are planning to determine appropriate thresholds for these two metrics. The combined results of these studies ensures that these metrics can be embedded within the services currently offered by the Software Improvement Group.

References

[1] E. Bouwers, J. Correia, A. van Deursen, and J. Visser. Quantifying the analyzability of software architectures. In *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)*. IEEE Computer Society, 2011.

[2] E. Bouwers, A. van Deursen, and J. Visser. Quantifying the encapsulation of implemented software architectures. Technical Report TUD-SERG-2011-031, Delft Software Engineering Research Group, Delft University of Technology, 2011.