8
Conclusions and Future Work

The degradation of the software architecture is quite hard to be avoided (Parnas, 1994; Eick *et al.*, 2001; van Gurp and Bosch, 2002). Among other factors, degradation is hard to be avoided due to the continuous manifestation of code anomalies while implementing and changing a system. Many code anomalies manifest themselves in the implementation of software systems independently of the modularization technique. However, code anomalies not always imply the degradation of the system's architecture. In other words, not all code anomalies are considered to be relevant from the architectural perspective. A plethora of researchers have proposed techniques and tools for detecting code anomalies (e.g. Emden and Moonen, 2002; Ratiu *et al.*, 2004; Marinescu, 2004; Ratzinger, 2005; Murphy-Hill, 2008; Tsantalis, and Chatzigeorgiou, 2009; Marinescu *et al.*, 2010; Moha *et al.*, 2010; Mara *et al.*, 2011). The most used and referred technique is called detection strategies. Each detection strategy corresponds to a logical composition of metrics that detect code elements suspects of suffer from a particular anomaly (Marinescu, 2004). Others researchers have investigated the impact of code anomalies under different perspectives, such as change-proneness (e.g. Kim *et al.*, 2005; Mäntylä and Lassenius, 2006; Lozano *et al.*, 2008; Khomh *et al.*, 2009; Rahman *et al.*, 2010 and Zazworka *et al.*, 2011) and fault-proneness (e.g. Li and Shatnawi, 2007; D'Ambros *et al.*, 2010; and Zazworka *et al.*, 2011).

To the best of our knowledge, however, there is no research work that analyses the impact of code anomalies on the software architecture. In particular, this occurs because there is no knowledge about to what extent code anomalies relate to architectural degradation, despite of several researchers have highlighted such relationship (Fowler *et al.*, 1999; Godfrey and Lee, 2000; Eick *et al.*, 2001; van Gurp and Bosch, 2002; Maccormack *et al.*, 2006). Also, there is no evidence about how accurate the conventional strategies for code anomaly detection are when identifying architecturally-relevant ones. Furthermore, conventional

strategies tend to detect long lists of code anomaly occurrences even in small software systems, many of which are false positives. Therefore, due to time and resource constraints developers are neither able to refactor all the code anomalies nor identify those that demand immediate attention for the architectural perspective.

In this context, the systematic assessment of architecturally-relevant code anomalies is essential to help developers to identify sources of architectural degradation, fostering smooth system evolution. The assessment of these critical code anomalies encompasses the following processes: the detection of single and inter-related code anomalies as well as the analysis of how they relate to the system's architecture. In this thesis, a suite of six aspect-oriented code anomalies was empirically identified. Then, a suite of architecture-sensitive metrics and detection strategies was proposed. Finally, a catalog of nine recurring inter-related code anomalies - anomaly patterns - was documented and formalized. During the empirical evaluations, we investigated: (i) the frequency rates of the proposed and already published aspect-oriented code anomalies, (ii) the relationship between code anomalies and architectural degradation in aspect-oriented and object-oriented systems, and (iii) the accuracy of the conventional detection strategies, of the architecture-sensitive ones and of code anomaly patterns when identifying architecturally-relevant code anomalies.

8.1. Revisiting the Thesis Contributions

In this thesis we discussed the need of understanding the relationship between code anomalies and architectural degradation as well as providing mechanisms to support the identification of architecturally-relevant code anomalies. In this direction, we documented a suite of code anomaly patterns which impact on the architecture design in a wide range of ways. These code anomaly patterns are built on a set of anomalous code elements identified by using a suite of architecture-sensitive detection strategies. In order to define such architecture-sensitive strategies, a suite of architecture-sensitive metrics was proposed. In summary, the six contributions of this research work are described in what follows.

1. *A Catalog of Recurring Code Anomalies in Aspect-Oriented Systems (Chapter 3)*. The catalog of code anomalies guides developers in promoting the modularity of aspect-oriented systems (Macia *et al.*, 2011a). In particular, the documented anomalies were inspired by the limitations identified in this sort of systems (Piveta *et al.*, 2006; Srivisut and Muenchaisri, 2007), such as the mimic of object-oriented anomalies. Our contribution is the documentation of code anomalies that represent recurring misuses of aspect-oriented mechanisms.
2. *A Suite of Architecture-Sensitive Metrics (Chapter 6)*. We defined a suite of architecture-sensitive metrics to be exploited in the detection of code anomalies (Macia *et al.*, 2013). These measures quantify two kinds of architecture-sensitive information: (i) how architectural components are implemented by the code elements and (ii) how architectural concerns are modularized by the code elements. These kinds of information can be partially recovered from the system implementation by third-party tools (Eisenbarth *et al.*, 2003; Maqbool *et al.*, 2007; FEAT, 2009; Garcia *et al.*, 2011; Nguyen *et al.*, 2011). Therefore, the proposed metrics could be gathered even when the architectural design is not available as it often occurs. On the other hand, when the architectural design is documented, the proposed metrics can be used in different architectural views, such as component-and-connector view and module view.
3. *A Suite of Architecture-Sensitive Detection Strategies (Chapter 6)*. We also proposed a suite of architecture-sensitive detection strategies, which combines source code and architecture-sensitive metrics (Macia *et al.*, 2013). The goal of this suite is to overcome the limitations of conventional detection strategies and, thus, provide developers with a more accurate list of architecturally-relevant code anomalies. This suite can be extended and, by no means, we claimed that the proposed strategies are exhaustive. In other words, new architecture-sensitive strategies could be elaborated to detect additional code anomalies.
4. *A Catalog of Code Anomaly Patterns (Chapter 7)*. We also documented a first catalog of nine recurring inter-related code anomalies, the *so-*

called code anomaly patterns. The patterns were grouped into four intuitive categories according to their common characteristics, facilitating their recognition by software engineers. The catalogue of code anomaly patterns aims at guiding software engineers in promptly identifying the anomalous code elements that harmful impact on the architecture design. These patterns were systematically observed through the analysis of evolving software systems from several domains, implemented using different programming languages and modularization techniques. This catalogue is a novel contribution as there is no evidence about which inter-related anomalous code elements can harmful impact the software architecture.

5. *Tool Support (Chapter 7)*. We also designed and implemented a tool, named SCOOP (Macia *et al.*, 2012c), which supports the detection of code anomaly patterns. To this end, SCOOP provides engineers with a Domain Specific Language allowing them to define their own detection strategies, including the architecture-sensitive ones. This means that SCOOP supports: (i) collecting the proposed architecture-sensitive metrics, (ii) employing the architecture-sensitive detection strategies, and (iii) identifying existing code anomaly patterns. The output generated by SCOOP includes lists of single anomalous elements (identified by detection strategies) and code anomaly patterns. SCOOP was implemented as an Eclipse plug-in (Eclipse, 2011). Its current implementation exploits the architectural concern projections documented using ConcernMapper and Vespucci, which have a list of concerns names and the code elements that modularize each concern. Additionally, SCOOP relies on Vespucci to exploit how code elements relate to architectural components.
6. *Empirical Evaluation*. We designed and executed empirical studies to evaluate our catalogue of aspect-oriented anomalies, the suite of architecture-sensitive metrics and strategies as well as the code anomaly patterns.

a. *Frequency of aspect-oriented Code Anomalies (Chapter 3)*. The aim of this evaluation was centered on observing the occurrence and frequency of the documented code anomalies. We carried out this evaluation using three software systems (Macia *et al.*, 2011a). Many successive releases of the target systems were investigated as well as their evolving code anomalies.

b. *Relationship between Code Anomalies and Architectural Degradation (Chapter 4).* We also investigated the interplay between code anomalies and architectural degradation (Macia *et al.*, 2012b; Macia *et al.*, 2011d). This investigation relied on the evolution of five software systems implemented using different programming languages and modularization techniques. First, we analyzed to what extent code anomalies are likely to favor architectural degradation. Additionally, we assessed the proportion of architectural degradation symptoms in the system implementation that could be removed through refactoring code anomalies. Second, we investigated whether certain properties of the code anomaly indicate their architecture-relevance. Finally, we assessed how often developers remove architecturally-relevant code anomalies by means of refactoring actions.

c. *Accuracy of Conventional Detection Strategies (Chapter 5)*. Once the correlation between code anomalies and architectural degradation symptoms was confirmed, the accuracy of conventional strategies was assessed when detecting architecturally-relevant anomalies. The evaluation involved the employee of nineteen conventional detection strategies using the same systems in which that correlation was confirmed (Macia *et al.*, 2012a).

d. *Architecture-Sensitive Strategies and Architectural Degradation (Chapter 6)*. A study was carried out to analyze the accuracy of architecture-sensitive strategies when detecting architecturally-relevant code anomalies (Macia *et al.*, 2013). In particular, such accuracy was assessed in the context of five Java systems. In this study, we also investigated the contribution of each kind of architecture-sensitive information - components and concerns

projections. Finally, we analyze the influence of projecting concerns at different granularity levels - method- and class level - on the strategies accuracy.

e. *Code Anomaly Patterns and Architectural Degradation (Chapter 7)*. The last evaluation performed in this thesis aimed at assessing whether and to what extent the documented anomaly patterns are better indicators of architecturally-relevant anomalies than single code anomalies. This evaluation was carried out in the context of five software systems that present different stages of the architectural degradation.

8.2. Future Work

In spite of the contributions of this thesis described in Section 8.1, we have identified some future work. Basically, five main topics can be derived and they are described as follows.

1. *Further Evaluations*. The proposed architecture-sensitive strategies and the documented anomaly patterns were evaluated in the context of five representative Java systems (Chapters 6 and 7). The performed studies provided evidence with respect to the benefits of exploiting architecture-sensitive information and relationships between anomalous code elements in the detection of architecturally-relevant code anomalies. However, it is important to undertake further empirical investigations considering other software systems.

   a. Our evaluation focused on system versions that presented different stages of the architectural degradation. Thus, it could be interesting to perform further studies considering the evolution of degraded software architectures. The goal of this kind of study is to gather findings about how architecturally-relevant code anomalies are formed aiming at promptly identify their occurrences. This early identification allows engineers to save effort in later maintenance tasks.

b. The performed studies considered system versions that presented different architectural decompositions. It could be interesting to perform studies involving groups of systems following the same architectural decomposition. The goal is to investigate which anomaly patterns are likely to occur more often in systems built on the basis of certain architectural design.

c. In the evaluation we only relied on object-oriented systems. This occurred because we did not find any available system developed using other programming techniques that meet the relevant criteria listed in Table 6.4 (Bergmans *et al.*, 1992; Harrison and Ossher, 1993; Prehofer, 1997; Kickzales *et al.*, 1997; Rajan and Sullivan, 2005). Therefore, it is necessary to replicate the performed studies in systems developed with post object-oriented programming techniques. This kind of study is interesting because the relationship between code anomalies and architectural degradation was confirmed not only in object-oriented systems.

2. *Improved Tool Support*. We also identified some improvements to be implemented in SCOOP regarding the code anomaly and patterns detection.

   a. It would be interesting to rank the code elements identified by each strategy according to different criteria, such as architectural role, change-proneness, and fault-proneness (Arcoverde, 2012). These rankings will provide engineers with a broader perspective about the harmful impact of the anomalous code elements. Therefore, software engineers may better decide which anomalous code elements must be refactored first.

   b. It could be useful to extend SCOOP to incorporate mechanisms for architecture design and concern recovery. Therefore, the tool could include an automate process of code anomaly detection when the architecture design information is not given as input. As not all concerns are architecturally-relevant, SCOOP could allow engineers to select which recovered concerns they consider to be relevant.

3. *Refactoring Recommendation for Code Anomaly Patterns*. Besides the detection of code anomaly patterns, engineers should be guided with refactoring sequences to remove their occurrences. The suggestion of such sequences is a challenge due to several reasons. First, they have to remove anomalies infecting inter-related code elements. Second, the suggested sequence might have to remove different kinds of code anomalies. Third, the sequence has to be effective not only in removing the code anomalies without affecting the system behavior, but also in removing the architectural degradation symptoms. Finally, the proposed sequence should contain a low number of refactorings in order to save engineers' effort.
4. *Expanding the Suite of Code Anomaly Patterns*. In this thesis we documented a catalog of nine code anomaly patterns. A natural question is how complete the proposed suite of anomaly patterns is. By no means we have claimed that the proposed suite is exhaustive. In fact, the proposed code anomaly patterns act as a stepping stone towards the understanding of particular "shapes" of inter-related code anomalies and their relation with architectural degradation. Therefore, new anomaly patterns can be further observed either individually or by composing existing ones.
5. *Visualization of Code Anomaly Patterns*. As software systems evolve and the number of code anomalies increases, analysis of code anomaly patterns may become a challenge and a time-consuming task. Visual representations (Carneiro *et al.*, 2008a; Carneiro *et al.*, 2008b; Wettel and Lanza, 2007; Ducasse *et al.*, 2006; Lanza, 2004; Lanza and Ducasse, 2003; Stasko *et al.*, 1998) have been shown useful in supporting code anomalies and concerns analysis (Novais *et al.* 2012; Wettel *et al.*, 2011; Novais *et al.* 2011; Carneiro *et al.* 2010). In this direction, it would be interesting to analyze whether existing visualization techniques help engineers in the analysis of code anomaly patterns and, if necessary, propose new visual representations to this end.

In conclusion, this thesis represents a first investigation that focuses on the relationship between anomalous code elements and architectural degradation symptoms in software systems implementation. In order to reach this macro goal, this work achieved a set of contributions (Section 8.1). In particular, this thesis has initiated a new avenue in architecture revision based on the analysis of anomalous code structures and their relationships. Nevertheless, as aforementioned, there are also uncovered research topics to be explored in the future.