6

Final Considerations

"The mission given is a mission accomplished!"

Kleinner Farias
PhD classmate - class 2009 - PUC-Rio

Composition properties play a role in the stability of advanced programming techniques. When programs evolve without taking into consideration the effects of these properties, their stability tends to decrease. As a consequence, programs become difficult to maintain and evolve if developers are not aware of composition properties. This occurs mainly because it is not trivial to identify and understand certain composition properties. Then, our intention is to provide developers with support for managing composition properties. Measurement frameworks are the traditional means for characterizing and quantifying the basic properties of a program.

A number of metrics have been proposed to quantify changes in evolving programs. However, most of these metrics are defined based only on the properties of modules. This means that existing metrics are not sensitive to the composition properties. In turn, the understanding of composition properties is hindered because conventional modularity metrics are not able to quantify the impact of composition properties on program stability. In order to understand this impact, a formalism on composition properties is required. Based on this formalism, metrics can be created to quantify composition properties and study their impact on the program stability.

Finally, it is also recognized that many of these composition properties should be identified at earlier development stages; for instance, explicitly describing such properties in design models. However, there is no investigation on the benefits of specifying composition properties yet at the design level. For instance, there is no study investigating if explicit composition modeling supports developers in dealing with changes through software maintenance and evolution.

6.1 Revisiting the Thesis Contributions

In this thesis we discussed the need of managing composition properties in order to better manage the program stability. Therefore, we claim that quantitative assessments of advanced programming techniques should be rooted not only on quantifying module properties. In fact, composition code assessment must be guided by the understanding of the composition properties that exert influence on program stability.

Based on a set of composition properties (Chapter 4), which are harmful for program stability, this research work defines a basic terminology and a measurement framework to support the quantification of composition properties. In addition, we evaluate the availability of the explicit composition models on performing program changes. Hence, the contributions of this work, as stated in Chapter 1, are:

- **Empirical Findings on the Role of Modularity in Indicating Software Stability (Chapter 3).** We carried out a number of empirical studies in order to evaluate whether modularity is a good indicator of composition-enriched programs stability. Our investigation (Chapters 3) was accomplished using three different systems (see Appendix A). These studies provided evidence that composition-enriched programs modularity is not a good indicator of their stability. In addition, it was possible to compare the effectiveness, in terms of stability, of each composition mechanism. This analysis is a novel contribution as there is a lack of a general understanding that more modular composition-enriched programs tends to be more stable.
- **Composition Measurement Framework (Chapter 4).** Based on the proposed formalism, we defined a composition measurement framework for supporting the characterization and the quantification of composition properties by means of a suite of composition metrics. It was developed to be applied to any advanced programming language. The detailed measurement framework is presented in Chapter 4. This framework is evaluated so far using AOP and FOP techniques. It is a contribution because it captures the nuances of composition properties regardless advanced programming techniques, which have been so far completely misunderstood. Finally, we design and implement a tool, named `CoMMes`,

which supports the composition properties quantification by means of the proposed framework (Chapter 4).

– **Empirical Findings about Maintenance Improvements with Explicit Composition Modeling (Chapter 5).** We carried on an Internet-based experiment towards the benefits of making the composition properties specification available to developers at the composition design level. Our investigation took into consideration evolving recurring scenarios of different evolving application (see Appendix A). Two groups of developers participated in the experiment: one group using a plain UML composition design and the other one using a UML composition design enriched with composition properties details. This experiment provides evidences that the availability of the composition properties specification alleviates the developers' effort in maintaining composition-enriched programs focusing on their stability.

6.2 Future Work

In spite of the contributions of this thesis described in Section 6.1, there are many other directions for and future work, some of which are described in the following paragraphs.

Additional Studies on Composition Properties. The evaluation studies provided evidence of the existing correlation between composition properties and stability. However, it is necessary to undertake additional studies in order to (i) assess the measurement framework in different contexts, such as Compose* (COMPOSE PROJECT, 2012), and (ii) use and evaluate our framework in the context of other quality attributes, such as error-proneness. We also aim at studying the composition properties impact on two different domains: the dependency of features in SPL applications and code anomalies. The study of feature dependency is particularly interesting because features often depend on each other in intricate forms, making the code hard to maintain. At the same time, maintainable aspect-oriented systems are still a challenge for software developers. In fact, the expressive power of AOP mechanisms might facilitate the introduction of certain code anomalies that emerge in the source code, such as god class (PIVETA *et al.*,2006, SRIVISUT *et al.*,2007, BERTRAN *et al.*,2007), whether these composition properties can lead

to code anomalous in the program source code. Regarding explicit composition modeling, further studies are still required to confirm our results on the impact of this modelling strategy on the realization of maintenance programming tasks. The main reason for this is that we need to better understand whether UML+ models also play a role when using other composition techniques, such as feature-oriented programming.

Composition Measurement Framework Refinement. These additional studies would enable us to reveal any extensions needed in our composition framework. This extension is particularly important to validate the generality of our framework. In addition, this extension can also contribute for the identification of additional composition properties that are harmful to composition-enriched program stability.

Tool Support. At least two improvements are needed to make CoMMes usable in practice: (i) creating a graphical interface for it, and (ii) incorporating more sophisticated strategies for identifying and mapping composition properties. This way, the tool CoMMes could be integrated to popular IDE, such as Eclipse, in a way that developers could be aware of the composition properties while developing their software projects.

UML profile. The last study provided evidence that developers tend to generate more stable programs when models enriched with composition properties are available. Based on this need, an UML profile that provides support for properly modeling these properties is required. One of the major advantages of this UML profile would be the ability to systematically introduce further advanced programming techniques without having to re-create the whole modeling environment.