6
Final Remarks and Future Work

Maintaining the original design of evolving program families and their respective features is a challenging task. Among other factors, this task is mainly challenging when there are several development teams maintaining and making decisions in different family members. These decisions vary from the architectural design to implementation stages. When such decisions are made without respecting the original design of the program families and their features, they tend to degenerate the program family's source code. Hence, it is fundamental to analyse the feature evolution throughout all the family members in order to forwardly recover (i.e. identify and classify) the implementation elements realizing each family feature. There are many techniques and tools for supporting feature mapping activity in single applications (Wong *et al.* 1999, Eisenbarth *et al.* 2003, Eisenberg and Volder 2005, Poshyvanyk *et al.* 2007). There are others that explore source code history to detect change dependencies, non-functional crosscutting features and characteristics of the software evolution. However, all of them are unable to forwardly recover the implementation elements realizing the features in evolving program families.

To the best of our knowledge, there is no research work that analyses the change history of the features' implementation elements in evolving program families over time. In particular, this occurs because existing techniques only concentrate on an individual version of a single program. Even worse, existing techniques rarely analyse and recover the evolution of each family feature. A few cases of recently-proposed techniques consider the change history of a particular program to support the identification of non-functional crosscutting features (Adams *et al.* 2010, Nguyen *et al.* 2011). However, they do not incrementally analyse the change history of the features' implementation elements in program families over time.

In this context, history-sensitive heuristics are required to forwardly recover the implementation elements realizing evolving common and varying features in program families. The forward recovery of features encompasses the multi-dimensional history analysis in order to identify and classify the features'

implementation elements. The forward recovery of program family's features can be useful to a wide range of repair actions, including the code restructuring activities across the program family. First, a set of eight recurring types of feature mapping mismatches were empirically identified and characterized in evolving members of a program family. Second, we defined and formalized a suite of five heuristics for expanding feature mappings for all the versions of the family members. Finally, we defined history-sensitive heuristics for the forward recovery of features in the code of evolving program families. In the empirical evaluations, we evaluated the frequency rate of the documented mapping mismatches and the accuracy of the two suites of heuristics. The next section revisits the key contributions of this thesis and Section 6.2 points out directions for future work.

6.1 Revisiting the Thesis Contributions

In this thesis we discussed the need of recovering the features' implementation elements in degenerate program families. In this direction, we proposed a suite of forward recovery heuristics that rely on expanded feature mappings. These expanded feature mappings were generated by a suite of mapping expansion heuristics. To define these mapping expansion heuristics, we documented a set of recurring types of mapping mismatches. In summary, the six contributions of this research work are described as follows.

1. *A Catalogue of Recurring Mapping Mismatches (Chapter 3).* The catalogue of mapping mismatches guides software engineers in promoting the correctness and completeness of their feature mappings. Additionally, it can also be used in conjunction with existing mapping techniques to check the correctness of their generated mappings. These mapping mismatches were empirically documented through the analysis of evolving members of a program family. This catalogue is a novel contribution as there are no evidence about which types of mismatches can arise during feature mapping activities (Nunes *et al.* 2011, Nunes 2011).

2. *A Suite of Mapping Expansion Heuristics (Chapter 4).* A suite of five heuristics for the feature mapping expansion was defined and formalized (Nunes *et al.* 2010, Nunes *et al.* 2012a). The mapping expansion heuristics rely on the multi-dimensional historical of program families and the catalogue of mapping mismatches. Their intent is also to support the automatic elimination of mapping mismatches given a set of evolving members of the same family. The proposed heuristics incrementally analyse

the change history of the features' implementation elements in program families over time.

3. *A Suite of Heuristics for Feature Element Recovery (Chapter 5).* We defined history-sensitive heuristics for the forward recovery of features in the code of evolving program families (Nunes *et al.* 2012b). The recovery heuristics use the expanded feature mappings as input information provided by the mapping heuristics. The recovery heuristics aim at classifying the implementation elements as part of common or variable features of the program family. The output generated by the heuristics is a Java project that is structured in terms of program family's features. These packages are further composed of sub-packages that represent the several categories defined by each recovery heuristic.

4. *Tool Support (Chapters 4 and 5).* We also designed and implemented a prototype tool, named MapHist (Nunes *et al.* 2012a), which effectively supports the use of the mapping expansion heuristics. Additionally, we also implemented the recovery heuristics taking into consideration the output generated by the mapping heuristics; i.e. the expanded feature mappings. The MapHist tool was implemented as an Eclipse plug-in (Eclipse 2011). MapHist uses the representation of features provided by ConcernMapper, which has a list of features' names and the implementation elements that realize each feature.

5. *Feature Visualization in Evolving Program Families (Chapter 4).* The expanded feature mappings have been successfully used for feature evolution comprehension in program families (Nunes *et al.* 2012a, Novais *et al.* 2012). The feature evolution visualization was achieved through the integration of the mapping expansion heuristics with a visualization tool, the so-called SourceMiner Evolution (SME). This integration enables developers to analyse the feature code evolution through two or more family member versions using multiple graphical views.

6. *Empirical Evaluations.* We designed and executed empirical studies to evaluate our catalogue of mismatches and the two proposed suites of heuristics.

   a) *Experiments on Mapping Mismatches (Chapter 3).* The aims of the experiments were centered on observing the occurrence and frequency of the mismatch categories. We run the experiments with 26 participants using two different systems. From these experiments, we could notice that

in fact the documented mismatches occur in practice either performed manually or using conventional mapping tools.

b) *Experiments on Accuracy of the Heuristics (Chapters 4 and 5).* We also evaluated the accuracy of the two proposed suites on two industrial evolving program families named OC and RAWeb (Section 4.6.1). First, the expanded features mappings generated by the mapping expansion heuristics were evaluated with respect to recall and precision measures. Second, the forward recovery heuristics were also evaluated regarding their categories in terms of precision and recall. The goal was to verify the forward recovery of the elements associated with each category.

6.2 Future Work

In spite of the contributions of this thesis described in Section 6.1, we have identified some future work. Basically, five main topics can be derived and they are described as follows.

1. *Further Evaluations.* The catalogue of mismatches was evaluated through two representative systems from different domains (Chapter 3). Additionally, the two suites of heuristics were evaluated on the top of two evolving program families. The designed studies provided reasonable evidence with respect to the accuracy of the proposed heuristics. However, it is important to design other studies taking into consideration other program families in order to evaluate (i) the generalization of our catalogue of mismatches, (ii) the accuracy of our heuristics, (iii) the accuracy of the heuristics when varying the seed mappings, and (iv) if the execution order of the heuristics varies depending on the characteristics of the program family's source code. For instance, new mismatches might be observed and identified based on false positives and false negatives generated by the automatic support of the heuristics.

2. *Comparison with other Techniques.* We intend to evaluate our technique with existing ones by applying them multiple times for each member of the program family. It is important to highlight that existing techniques were conceived with specific purposes. For instance, Nguyen *et al.* (Nguyen *et al.* 2011) aims at identifying candidates crosscutting features that should be implemented as aspects. On the other hand, our method is more general and it is not related to a specific engineering activity.

3. *Improved Tool Support.* We also identified some improvements to be implemented mainly regarding the recovery heuristics. First, it would be

interesting to provide support to a ranking process of the elements classified by each recovery category. This type of information provides developers with a detailed knowledge of the elements identified by each category and their frequency throughout program family evolution. Second, it could be useful to implement a more efficient recovery process based on the feedback provided by the developers. Based on this interaction, the heuristics could learn and disregard future recovery of a given implementation element.

4. *Generation of Feature Models.* Feature models are used to represent the commonalities and variabilities of a program family (Kang *et al.* 2006). The availability of feature models is interesting as they provide a high level knowledge about the features, their classification and their dependencies. For this reason, it would be valuable to generate feature models for each family member version and analyse their change impact under the perspective of a high level representation. This analysis could anticipate to some extent certain implementation issues during the program family evolution.

5. *Detection of Architectural Violations.* As part of this thesis, we integrated the mapping expansion heuristics with the SME visualization tool to analyse the feature evolution through multiple views. Additional analyses could be derived from these expanded feature mappings. For instance, we could analyse how the presence of code anomalies, also known as code smells, in a given feature implementation affect negatively other features' implementations. The presence of code anomalies can introduce architectural violations, and consequently some architectural design rules could help developers visually check the conformance of feature code under evolution (Hochstein and Lindvall 2005, Eichberg *et al.* 2008).

6. *Analysis of Feature Dependencies and Stability.* Other analyses on expanded feature mappings can be derived in terms of feature dependencies and stability analyses. For instance, we could propose heuristics or even visualizations to observe the number of changes that a feature has undergone over time; i.e the feature stability throughout program family evolution. Additionally, we could visually observe the change impact of a feature over others or their dependencies. This way, we could rank which features or group of features are more harmful to the program family stability during its evolution.

In conclusion, this thesis represents a promising direction regarding our macro goal that is the recovery of features in code of evolving program families. In order to reach this macro goal, this work achieved a set of contributions (Section 6.1). In particular, this thesis has addressed relevant issues related to the feature mapping evolution and recovery of features' implementation elements. Nevertheless, as aforementioned there are also uncovered issues to be explored in the future.