**Protocol for Manual Validation**

**on Incomplete Composite Refactorings**

1. **Contextualization**

Code refactoring is the process of applying code transformations aimed to improve the internal code structure of a software program [1]. However, due to its fine-grained nature, a single refactoring rarely suffices to improve the internal code structure. Developers often composite refactorings (shortly composite), *i.e*, two or more interrelated single refactorings [2].

Previous studies recommended some composites to remove poor code structures, code smells[1, 2, 3]. However, recent studies presented that the composite application has failed to remove the code smells [2].

This happens because developers often do not apply one or more single refactorings that are recommended to remove the code smell. In other words, developers apply incomplete composite refactorings (shortly incomplete composites), *i.e*, composites that do not have one or more single refactorings recommended to remove some particular smell type.

The incomplete composites are often applied to remove two types of code smells. They are Feature Envy and God Class [2]. A Feature Envy is when a method is more interested in other classes [1]. A God Class is when a class implements too many software features and other classes have a high dependency on this class.

1. **Validation Goal**

This validation aims to evaluate if composite refactorings that have at least one refactoring type of recommended refactoring types to remove Feature Envy or God Class are incomplete composites for these code smell types. Moreover, this validation aims to collect data about the context of the incomplete composites are applied. We then selected randomly 31 composite refactorings from seven source projects, in which these composites were applied on code elements that have at least one Feature Envy or God Class.

1. **Validation Steps**

The steps of the validation are described as follows.

1. Open the document Manual Validation - Incomplete Composites. The document has three tabs.
   1. The first tab is “Guide of Incomplete Composites” which presents the types of incomplete composites for each recommended composite refactorings for the Feature Envy removal and God Class removal.
   2. The second tab is “Incomplete Composites - Refactorings” which presents the refactorings of each composite refactoring. The first column presents the composite id. The second column presents the classes and code smells before each refactoring of the composite. The second column presents the classes and code smells after each refactoring of the composite.
   3. The third tab is “Incomplete Composites - Questions” which presents the complementary data about each composite refactoring and the questions to be answered. The columns of this tab are described as follows.
      1. The first column presents the names of the evaluator(s).
      2. The second column presents the composite id.
      3. The third column presents the code smell type to be evaluated (E.g Feature Envy or God Class)
      4. The fourth column presents the project name which the composite was applied
      5. The fifth column presents the link of the commit in which the composite refactoring was applied.
      6. The next columns present the questions to the validation.
2. Open the link of the commit in which the composite refactoring was applied (tab “Incomplete Composites - Questions”). To open this link is necessary to have a GitHub account (Please, access <https://github.com/>)
3. Search for the code elements in which the composite refactoring was applied. Please, click ‘Ctrl + F’ and digit the code element names (eg. “ClassA.java”). If the code element names were found, then confirm if the package name is the same. Otherwise, please, send an email for Ana Carla Bibiano <[abibiano@inf.puc-rio.br](mailto:abibiano@inf.puc-rio.br)> and report that the composite refactoring was not found, and go to step 2 for the next composite refactoring.
4. If the code elements were found, then confirm that the composite refactoring was applied to the code elements. If yes, then, fill the questions to this validation.
5. Go to step 2 for the next composite refactoring.

1. **Validation Questions**

The steps to answer each validation questions are described as follows.

**First Question. Is Incomplete Composite for the evaluated code smell removal? (Yes/No).**

→ To answer this question, you may see the code elements in which the composite was applied from Github, and you may evaluate if the code smell was removed or not according to the source code and the data presented in the “Incomplete Composites - Refactorings” tab. If the code smell was not removed, then the composite refactoring is an incomplete composite.

**Second Question. Could you please justify your previous answer?**

→ You may justify why you consider that the composite is or not an incomplete composite according to the previous answer.

**Third Question. In case of yes for the previous question: which other refactorings could be have been applied to remove the smell?**

→ To answer this question, you may see if the recommended refactorings described in the “Guide of Incomplete Composites” tab could have been removed the code smell.

**Fourth Question. Could you please justify your previous answer?**

→ You may justify why the recommended refactorings could be or not applied to remove the code smell according to the previous answer.

**Fifth Question. What was the intent of the developer in this commit? (e.g, refactoring only, feature addition, bug fixing and etc).**

→ To answer this question, you may observe: (i) the commit message, (ii) description, (iii) the code changes that were applied in the classes in which the composite was applied, (iv) if the commit is related to some pull request. These observations can help you to understand the intent(s) of the developer in the commit in which the composite was applied. Otherwise, you may answer “Intent not found”.

**Sixth Question. Could you please justify your previous answer?**

→ You may justify why and how you considered the intent(s) of the developer to apply the composite.

**Last Question. What were the types of classes in which the refactorings were applied? (Types: Superclass, Subclass, Abstract class, Inner class, Simple class)**

→ To answer this question, you may observe the signature of the classes in which the composite was applied to collect if the types of the classes.

**References**

[1] Fowler, 1999, "Refactoring: Improving the Design of Existing Code"

[2] Bibiano *et* al. "A Quantitative Study on Characteristics and Effect of Batch Refactoring on Code Smells." ESEM'19

[3] Cedrim et al. "Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects." FSE’17