Studying Expert Initial Set and Hard to Map Cases in Automated Code-to-Architecture Mappings

Tobias Olsson, Morgan Ericsson and Anna Wingkvist

Department of Computer Science and Media Technology, Linnaeus University, Kalmar/Växjö, Sweden

Abstract

We study the mapping of software source code to architectural modules.

Background: To evaluate techniques for performing automatic mapping of code-to-architecture, a ground truth mapping, often provided by an expert, is needed. From this ground truth, techniques use an initial set of mapped source code as a starting point. The size and composition of this set affect the techniques' performance, and to make comparisons, random sizes and compositions are used. However, while randomness will give a baseline for comparison, it is not likely that a human expert would compose an initial set on random to map source code. We are interested in letting an expert create an initial set based on their experience with the system and study how this affects how a technique performs. Also, previous research has shown that when comparing an automatic mapping with the ground truth mappings, human experts often accept the automated mappings and, if not, point to the need for refactoring the source code. We want to study this phenomenon further.

Audience: Researchers and developers of tools in the area of architecture conformance. The system expert can gain valuable insights into where the source code needs to be refactored.

Aim: We hypothesize that an initial set assigned by an expert performs better than a random initial set of similar size and that an expert will agree upon or find opportunities for refactoring in a majority of cases where the automatic mapping and expert mapping disagrees.

Method: The initial set will be extracted from an interview with the expert. Then the performance (precision and recall f1 score) will be compared to mappings starting from random initial sets and using an automatic technique. We will also use our tool to find the cases where the automatic and human mapping disagrees and then let the expert review these cases.

Results: We expect to find a difference when performance is compared. We expect the expert review to reveal source code that should be remapped, source code that needs refactoring (e.g., possible architectural violations), and points where the automatic technique needs to be improved.

Limitations: The study will only focus on only a single system, which limits the external validity significantly. The protocol for the interaction with the human expert can also introduce validity problems; for example, a mapping presented by an algorithm could be perceived as more objective and thus more acceptable for a software engineer.

Conclusions: We seek to improve our understanding of how a human creates an initial set for automatic mapping and its effect on how well an automated mapping technique performs. By improving the ground truth mappings, we can improve our techniques, tools, and methods for architecture conformance checking.

Keywords

Orphan Adoption, Software Architecture, Source Code Clustering, Naive Bayes

1. Introduction

Creating a mapping from the source code to an architectural model is perceived as labor-intensive. It hinders widespread use of Static Architecture Conformance Checking (SACC) practices such as Reflexion modeling in industry [1, 2]. A mapping is an assignment of a source code entity, e.g., a source code file or class, to an architectural module, e.g., a layer or a sub-system. The architectural modules and the dependencies between them form an intended architecture, see Figure 1. The mappings are used to determine if the dependencies in the source code conform to or violate the intended dependencies as described in the architecture, i.e., conformance checking.

2. Background

Current semi-automatic techniques build on the Human Guided clustering Method (HuGMe) and introduce different attraction functions that guide the automatic mapping [3, 4, 5, 6, 7]. HuGMe consists of a few essential steps as described below:

1. An initial set is created manually.
2. The entities to be mapped are determined.
3. The attraction function calculates an attraction for each entity and module.
4. If the attraction of a single module is deemed valid, the entity is mapped to this module.
5. If no valid attraction is found, the decision is left to a human user.

ECSA2021 Companion Volume

EMAIL: tobias.olsson@lnu.se (T. Olsson); morgan.ericsson@lnu.se (M. Ericsson); anna.wingkvist@lnu.se (A. Wingkvist)

ORCID: 0000-0003-1154-5308 (T. Olsson); 0000-0003-1173-5187 (M. Ericsson); 0000-0002-0835-823X (A. Wingkvist)

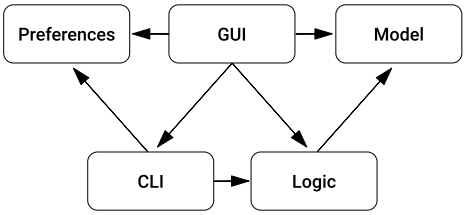

Figure 1: The intended architecture of JabRef version 3.7. The boxes represent modules, and the arrows represent allowed directed dependencies between these modules. The module Global has been omitted for clarity.

6. If new mappings are made, and there are entities remaining, continue at Step 2.
7. If entities are remaining, let the human user decide a mapping.

There are some things to note. First, the method is iterative, as the set of mapped entities can grow, and more mappings can be done. An initial set is needed to start the method, i.e., the attraction functions need some initial mappings to work with, see Figure 2. The human user is involved in several steps of the method. Thus it is semi-automatic and human-guided. However, the focus of most studies has been on the development and comparison of attraction functions, i.e., the automatic step of the method.

2.1. Initial Set

The attraction functions used in HuGMe all need an initial set of pre-mapped entities to work. In general, the previous studies have focused on comparing the automatic performance, e.g., precision and recall of different functions. In these studies, the initial set has been treated as a random variable considering size and composition [3, 4, 5, 6]. This assumption is fair in a general performance comparison. However, it does not necessarily reflect a realistic scenario.

A system expert would not select entities to map at random. Realistically, a system expert user could make a careful and well-thought-out initial set, select representative parts of the source code to map, or map everything easy to map and leave complex cases to the machine.

We have previously studied the effect of the initial set. We found large variations in attraction function performance depending on both size and composition of the initial set: A representative initial set can give good results even if the set is small, and vice versa [8]. Our goal was to help guide a system expert to produce a minimal but high-performing initial set with the help of standard source code metrics, and we found limited success in using inheritance-based metrics. However, the results did not generalize well for different subject systems.

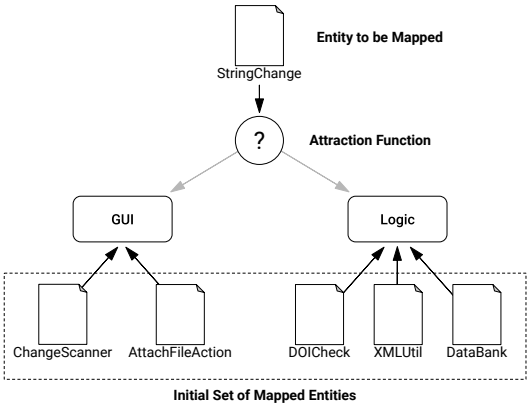

Figure 2: An example of how entities are mapped to modules. In this case, two classes are mapped to GUI, and three classes are mapped to Logic. These form the initial set. A sixth class, StringChange, is about to be mapped. The Attraction Function uses the information in the initial set to calculate an attraction to each module.

2.2. Ground Truth Mapping

Related to the performance of a technique is also the quest for a perfect mapping compared to the ground truth mappings. Previous research indicates that mappings' differences often reveal points where the source code needs refactoring, or developers made a mistake in the mapping.

Tzerpos and Holt [9] found that their technique suggested 46 entities to change mapping compared to the developers' assignment in one of their case studies. The developers agreed on this new suggested mapping in 33 cases, and the remaining 13 original mappings were considered valid but not optimal. In these 13 remaining cases, the developers expressed that restructuring the entities was needed to motivate their inclusion in the original modules. We have previously constructed a heuristic for automatic mapping of source code to Model-View-Controller-based architectures [10]. We evaluated the approach on four products in a product-line of games.

We compared the automatic mapping to the manual mapping of 653 entities and found a difference in mappings in 96 cases. Architectural problems caused 76 of these. Source code refactorings were suggested and implemented for two of the projects covering 23 architectural problems. An interesting finding is that the most common refactoring was *Move Type* (12 instances), i.e., move the type to the correct module. This indicates that a perfect automatic mapping is an elusive or even undesirable target, especially if a system has evolved for some time and has accumulated some erosion or drift.

2.3. Attraction Function and Tool

We have previously evaluated and implemented the attraction functions found in research by Bittencourt et al. [3], and Christl et al. [4] as well as implemented our own attraction function *NBAttract* based on machine learning [5]. To aid evaluations of attraction functions, we have set up an open-source tool aimed at allowing experimentation of different parameters and settings [11]. NBAttract has shown the most promise in our previous evaluation [5] and will be the main function used in this study. Our tool allows us to vary the information the function uses, and we plan on evaluating different combinations of the following:

- File names and paths.
- Architectural module names.
- Source code dependencies.
- Names of identifiers in the source code, e.g., method names, variable names, etc.

3. Audience

The study will be valuable to researchers and developers of tools in the area of architecture conformance. The system expert can gain valuable insights into where the source code needs to be refactored.

4. Aim

We want to know more about how a system expert would create an initial set of entities for semi-automatic mapping and the rationale for the specific mappings. This would give insight into the composition and distribution of an initial set created by an expert. We also want to know more about discrepancies in the automatic mappings compared to system expert mappings. To enable this, we need to build a broad set of data over multiple systems and experts. This study would act as a first initial study towards this goal.

We hypothesize that an initial set assigned by an expert performs better than a random initial set of similar size when used by our current best automatic mapping attraction function, NBAttract [5]

We hypothesize that an expert will agree upon or find opportunities for refactoring in a majority of cases where the automatic mapping using NBAttract [5] and expert mapping disagrees.

5. Method

The study will involve a human that is a system expert for a subject system. We assume a subject system implemented in Java with a documented architecture with defined architectural modules and allowed dependencies between these and a mapping from the source code to the architectural modules. If these do not exist, they need to be prepared before we can initiate the study. Optimally we will study these artifacts beforehand.

We perform the study in three phases. 1. we interview the expert to create an initial set and the rationale for the mapping. 2. we conduct experiments to find the performance of the initial set created by the expert compared to a random initial set of similar size. During this phase, we will generate a list of mapping discrepancies for the automatic mappings. And 3. we will interview the expert once more. This interview aims at investigating the mapping discrepancies found in Phase 2.

5.1. Phase 1: Initial Set Creation

To create an initial set, we will interview the human expert in a semi-formal way. The interview will be held online and recorded. There will be an agreement on the use and handling of the recording. This session will likely take less time than two hours. The interview protocol will follow this design.

1. Introduce yourself and explain that the interview is recorded and that the expert agrees to this.
2. Explain the purpose of the study and the use of the data.
3. Ask the expert about their involvement in the subject system's development, what role they have, and the general experience and the time frame of involvement.
4. Ask the expert about the subject system, its basic purpose, the end-users, and the architecture: what is the purpose of the architecture, the defined modules and dependencies, and did the expert create the architecture and mapping? How were the architecture and mapping created?
5. Explain the mapping scenario and give a rough outline of how an automatic mapping would work.
6. Ask the system expert for where they would start to map, any parts that jump to their mind or any easy to map parts of the system, e.g., whole directories/packages that can be mapped.
7. For each module in the architecture, ask the system expert to provide the most typical and important source code files. Ask why each file is deemed typical or important. At least 10% of the files should be provided.
8. Ask the expert if there is anything they would like to add.
9. Thank the expert and explain the remainder of the study. Book a new interview for Phase 3 to take place a few days later.

5.2. Phase 2: Experiments

We perform experiments where the expert's initial set is used with different combinations of information for the NBAttract attraction function. If we get differences in the initial sets (e.g., typical mappings vs. easy mappings) based on the first interview, we can compare these initial sets to each other. For further comparison, we will use a random initial set combination. Note that for random initial sets, several hundred experiments are needed, depending on the number of architectural modules, the number of source code entities, and the size of the expert's initial set. This phase will likely take three days to complete.

Experiments will generate the mapping data to calculate the precision and recall of the mappings. The data will also include a record of failed mappings, with the name of the source code entity and the failure frequency. Depending on the number of failures, a limit may be needed to not create an overwhelming burden for Phase 3. A suggestion is that a failure rate of more than 50% suggests a mapping discrepancy.

We will consider the expert's initial set better than a random initial set if the F1 score is better than the median F1 score of the random initial sets.

5.3. Phase 3: Validation of Mapping Discrepancies

We will investigate the generated mapping discrepancies by interviewing the human expert in a semi-formal way. The interview will be held online and recorded. There will be an agreement on the use and handling of the recording. The interview will likely involve looking at source code, so both the expert and researcher should prepare a development environment. If the interview session extends over two hours or if the expert expresses fatigue, it should be split into several sessions. The interview protocol will be conducted as follows.

1. Explain that the interview is recorded and that the expert agrees to this.
2. Explain the purpose of the study and the use of the data.
3. Roughly explain the results from the experiment.
4. For each mapping discrepancy, let the expert inspect the corresponding source code.
    a) Ask if the expert thinks the entity would need refactoring or that it contains serious problems.
    b) Remind the expert of the original mapping and ask if the expert still agrees to this mapping. If not, ask the expert for what mapping would be more appropriate and why.
    c) Show the automatic mapping results (several modules may be suggested). Ask if the expert would consider any of these mappings valid and why/why not.
5. Ask the expert if there is anything they would like to add.
6. Thank the expert.

6. Expected Results

We expect to find a difference when we compare performance. We expect that the human expert's initial set performs better than a random initial set of similar size. We expect the expert review to reveal source code that should be remapped, source code that needs refactoring (e.g., possible architectural violations), and points where the automatic technique needs to be improved.

7. Limitations

The study will only focus on a single system which limits the external validity. However, the long-term goal is to find more systems and experts and perform similar studies and build and refine the dataset over time. The protocol for the interaction with the human expert can also introduce validity problems; for example, a mapping presented by an algorithm could be perceived as more objective and thus more acceptable than the expert's informal knowledge. The expert may also be biased regarding certain parts of the source code that they have been more or less involved in. This will need to be noted in the interview protocol. And, the expert may be biased if they have created the original mapping or not, e.g., it is probably easier to accept a mapping presented by an algorithm if someone else did the original mapping. This has to be noted in the interview protocol.

8. Conclusions

We seek to improve our understanding of how a human creates an initial set for automatic mapping and its effect on how well an automated mapping technique performs. Based on what we learn, we can start to explore methods that actively suggest initial set candidates. By improving the ground truth mappings, we can improve our techniques, tools, and methods for architecture conformance checking.

Acknowledgments

The research was supported by the Centre for Data Intensive Sciences and Applications at Linnaeus University.

References

[1] G. C. Murphy, D. Notkin, K. Sullivan, Software reflexion models: Bridging the gap between source and high-level models, ACM SIGSOFT Software Engineering Notes 20 (1995) 18–28.

[2] N. Ali, S. Baker, R. O'Crowley, S. Herold, J. Buckley, Architecture consistency: State of the practice, challenges and requirements, Empirical Software Engineering 23 (2017) 1–35.

[3] R. A. Bittencourt, G. Jansen de Souza Santos, D. D. S. Guerrero, G. C. Murphy, Improving automated mapping in reflexion models using information retrieval techniques, in: IEEE Working Conference on Reverse Engineering, 2010, pp. 163–172.

[4] A. Christl, R. Koschke, M.-A. Storey, Automated clustering to support the reflexion method, Information and Software Technology 49 (2007) 255–274.

[5] T. Olsson, M. Ericsson, A. Wingkvist, Semi-automatic mapping of source code using naive bayes, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19, 2019, p. 209–216.

[6] A. Christl, R. Koschke, M.-A. Storey, Equipping the reflexion method with automated clustering, in: IEEE Working Conference on Reverse Engineering, 2005, pp. 98–108.

[7] F. Chen, L. Zhang, X. Lian, An improved mapping method for automated consistency check between software architecture and source code, in: IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), 2020, pp. 60–71.

[8] T. Olsson, M. Ericsson, A. Wingkvist, Towards improved initial mapping in semi automatic clustering, in: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18, ACM, 2018, pp. 51:1–51:7.

[9] V. Tzerpos, R. C. Holt, The orphan adoption problem in architecture maintenance, in: Proceedings of the Fourth Working Conference on Reverse Engineering, IEEE, 1997, pp. 76–82.

[10] T. Olsson, D. Toll, A. Wingkvist, M. Ericsson, Evaluation of a static architectural conformance checking method in a line of computer games, in: Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures, ACM, 2014, pp. 113–118.

[11] T. Olsson, M. Ericsson, A. Wingkvist, An exploration and experiment tool suite for code to architecture mapping techniques, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19, 2019, p. 26–29.