SORMASA: A tool for suggesting model refactoring actions by metrics-led genetic algorithm

Thierry Bodhuin Gerardo Canfora Luigi Troiano
RCOST – University of Sannio Viale Traiano – 82100 Benevento, Italy {bodhuin,canfora,troiano}@unisannio.it

ABSTRACT
In this paper we introduce SORMASA, Software Refactoring using software Metrics And Search Algorithms, a refactoring decision support tool based on optimization techniques, in particular Genetic Algorithms.

1. INTRODUCTION
During the development of object oriented software, dependencies (e.g. method call and use of attributes) between classes emerge that were not identified or specified explicitly at design time. This leads to a more complex software system than one desired having the same functionalities, hence, design modifications could be done in order to create a more manageable software in terms of maintenance and code reuse. Hereby, design decisions should be revised in the spirit of improving coupling and cohesion [4]. In order to reach this result, refactoring actions are generally performed aiming to improve the quality of the software architecture [1]. The ultimate goal is to increase cohesion of code and reducing coupling, but still keeping the initial idea of the solution architecture and semantics. Agile programming emphasized the role of refactoring up to support a process of continuous refactoring. This resulted into a pressing demand for tools able to automate refactoring tasks, or to support refactoring decisions [3].

Refactoring is often done by applying a series of transformations on an existing and often incomplete software system. When such transformations are applied, the software system remains fully compliant to the original requirements, differing only the implementation. While a single transformation (e.g. moving a method from a class to another) may not improve too much the software system, a series of such transformations may produce significant effect. Obviously, not all refactoring primitives (more than 80 primitives are listed at www.refactoring.com) are suitable for keeping low the software complexity. Among them, we have used two primitives, namely the MoveField and MoveMethod, that concerns respectively a field and method movement between two classes, hence affecting the coupling and cohesion of such classes. However, even considering few simple primitives, the effect of combining several transformations at the same time can lead to hard decisions due to the high number of possible combinations.

Refactoring can be viewed as an optimization problem, where each solution represents a set of refactoring actions, that if applied lead the system to an architecture entailing a different cohesion and coupling. A search based approach provides an interesting and viable solution to this problem, as it is able to automatically consider a high number of refactoring alternatives, then suggesting those that, if undertaken, can lead to a more cohesive and less coupled architecture. In this paper we describe SORMASA, Software Refactoring using software Metrics And Search Algorithms, developed at our research center with the goal of supporting decision making in refactoring a software architecture.

The current version of SORMASA makes some simplifying assumptions, that are (i) only field and method movements between classes are considered, (ii) transformations are assumed in isolation, ignoring the effect they can produce on each other, and (iii) all original classifiers are kept, with no new class or interface introduced or removed. This leads to transformations that are independent on the order in which they are applied and without modifications of the overall architecture. In the context of suggesting redesign actions, as addressed by SORMASA, these assumptions are not very limitative. Indeed, the goal is to suggest actions at model level concerning the best way of allocating class properties (i.e. methods and attributes), leaving the final decision to the user. Preliminary experimentation shows encouraging results.

Figure 1: The role of SORMASA in a process of continuous refactoring.

Figure 1 depicts the role of SORMASA in the process of continuous refactoring. The initial class model is coded in software artifacts (i.e. .java source files, .class bytecode), generally using a Java IDE such as Eclipse or NetBeans. SORMASA analyzes them (in particular bytecode) in order to identify and make explicit structural dependencies. The optimization process of SORMASA is aimed to identify refactoring opportunities that could improve the model quality in terms of high cohesion and low coupling. Opportunities are presented to the user that can decide which refactoring actions to undertake. The modifications will lead to a new revised software model from which the refactoring process may start again.

2. SEARCH ALGORITHM
SORMASA’s architecture is designed to work with different quality measurements (e.g. fitness function based on cohesion, coupling, complexity, etc.) and search algorithms. The current release supports Cohesion, Coupling [4], and Distance (that is the number of changes applied to the initial model) metrics for quality measurements and Genetic Algorithms (GA) as search algorithm. A set of refactoring primitives that are under consideration are coded like in Figure 2.

Refactoring suggestions can be obtained by comparing the solution to the initial model, then identifying the relocation of properties. SORMASA uses a Simple GA as described in [2], and that be outlined as follows:

1. An initial population of model candidates is randomly chosen.
2. The fitness of each candidate is evaluated.
3. (a) Select individuals for mating, according to their fitness
   (b) Perform crossover of selected individuals
   (c) Perform mutation
   (d) Replace individuals on population with offspring
4. Until the maximum number of generations is reached, repeat from 4.

Key aspects in the algorithm are (i) the fitness function, (ii) the genetic operators and (iii) the replacement policy. SORMASA allows to specify each of these aspects.

The effectiveness (fitness) of refactoring actions $fit$ can be obtained using a function based on structural metrics such as cohesion $ch$ and coupling $cp$, as they respectively represent the relatedness of class functionalities and the degree of dependency of a class on other classes. Moreover, we also consider the distance $d$ from the initial model as we prefer solutions that do not disrupt the original architecture. These variables are combined by weighted product, as

$$fit = ch^{w_{ch}} \cdot (1 - cp)^{w_{cp}} \cdot (1 - d)^{w_d} \quad (1)$$

All metrics are within the unary interval $[0, 1]$. This is the function we used in our experimentation, but other fitness functions may be specified, also including additional metrics.

The available genetic operators are selection (Tournament, Roulette Wheel), crossover (One-point, Two-points) and mutation (Simple) [2]. The replacement policies supported by SORMASA are (i) replacement of worst individual, that is slower but facilitating the algorithm convergence, and (ii) random replacement of individuals, that is faster [2].

3. AN EXAMPLE OF APPLICATION

Considering the example depicted in Figure 3, we can notice a set of dependencies between ClassA and ClassB, resulting into structure coupling. Moreover the structure is not cohesive, as methods can be partitioned according to the usage of class attributes. Moving method3 to ClassA

and method2 to ClassB provides a structure that is minimally coupled and maximally cohesive.

We notice that, these dependencies can emerge and become explicit mostly during the coding phase, as they depend on the actual use of class members. Hereby the need for refactoring. Obviously, the decision in undertaking refactoring actions depends also on the semantics of code, and this is a task left to the user. SORMASA only provides support in exploiting refactoring opportunities that can lead to code that is more cohesive and less coupled. SORMASA is able to deal with situations more complex than the one depicted in Figure 3, including inheritance of properties along with class generalization/specialization, and interface implementation.

4. CONCLUSIONS AND FUTURE WORK

SORMASA is a tool for supporting refactoring decisions aiming at optimizing the quality of software system (e.g. maximizing the cohesion and minimizing the coupling). This is obtained by implementing a search-based approach that is able to identify refactoring opportunities and to propose them to the user. SORMASA is at an early stage of development, and we plan to expand the feature set in order to:

1. Include more optimization techniques. In particular genetic programming looks a promising approach for searching a structured set of refactoring primitives that optimizes a quality function (i.e. fitness function). This would make possible to consider refactoring procedures, instead of simple primitives, such as the move of properties among classes.
2. Integrate SORMASA with Eclipse and NetBeans IDE. This can lead to have online refactoring suggestions during the coding of software solutions, thus enabling a continuous refactoring process.
3. Consider explicit refactoring and semantic constraints, able to better preserve software requirements.

5. REFERENCES