A Genetic Algorithm Approach to Design Evolution Using Design Pattern Transformation

Mehdi Amoui, Siavash Mirarab, Sepand Ansari, Caro Lucas

[1] Control and Intelligent Processing Center of Exellence,
Electrical and Computer Eng. Dept., University of Tehran,
Tehran, Iran
{mehdi.amoui , s.mirarab, sepans}@ece.ut.ac.ir , lucas@ipm.ir
http://cipce.ut.ac.ir/

Abstract. Improving software quality is a major concern in software development process. Despite all previous attempts to evolve software for quality improvement, these methods are neither scalable nor fully automatable. In this research we approach this issue by formulating it as a search problem. For this purpose, we applied software transformations in a form of GOF patterns to UML design model and evaluated the quality of the transformed design according to Object-Oriented metrics, particularly 'Distance from the Main Sequence'. This search based formulation of the problem enables us to use Genetic Algorithm for optimizing the metrics and find the best sequence of transformations. The implementation results show that Genetic Algorithm is able to find the optimal solution efficiently, especially when different genetic operators, adapted to characteristics of transformations, are used. Overall, we conclude that software transformations can successfully be approached automatically using evolutionary algorithms.

Introduction

High quality software needs to meet both its functional and non-functional requirements, such as reusability, performance, and robustness. Design and implementation defects that cause systems to exhibit low maintainability, low reuse, high complexity and faulty behavior are reduced in high quality software. Because of this, major proportion of total cost of the software development process is devoted to software maintenance [1, 2, 3]. Therefore, the need of techniques that reduce software complexity by incrementally improving the internal software quality becomes more obvious [4]. The first step toward achieving this goal is to quantize non-functional properties. In software engineering, metrics have long been studied as a way to assess the quality of large software systems [5] and have been applied to object-oriented systems as well [6, 7, 8, 9].

Secondly, there is a need of systematic way for software quality improvement at different stages and abstraction levels. To cope with this complexity there is an urgent

need for techniques that reduce software complexity by incrementally improving the internal software quality. The research domain that addresses this problem is referred to as restructuring or, in the specific case of object-oriented software development, refactoring [4]. The term refactoring was originally introduced by William Opdyke [10]. The main idea is to redistribute classes, variables and methods across the class hierarchy in order to facilitate future adaptations and extensions [11]. Based on object-oriented refactorings, Tokuda and Batory presented a pragmatic approach for high level object-oriented transformations to Design Patterns while preserving the software behavior [12]. In another approach for higher level transformations, Tahvildari et al. used a catalogue of object-oriented metrics as an indicator to automatically detect where a particular meta-pattern can be applied to improve the software quality via refactoring [13]. However, former attempts on design evolution are hardly automated. In case of software refactoring, the numerous combinations of valid transformations, makes the decision of applying appropriate transformations to achieve the highest software quality difficult. Therefore, search-based and evolutionary methods can be extensively used to automatically find merely the best possible sequence of valid transformations. This type of software evolution by using Genetic Algorithms and other search-based methods has been developed recently for simple “line of codes “(LOC) metric [14].

Throughout this contribution, we will propose a search-based evolutionary method, particularly Genetic Algorithm, to find the best sequence of valid high level design pattern transformations to improve software reusability. Improvement evaluation is based on measurement of some high level object-oriented design metrics.

The rest of this paper is organized as follow: First, we define the characteristics of chosen object-oriented design metrics, and then we discuss the details of our Genetic Algorithm functions, assumptions, and encodings. Next, our developed framework for automating the whole process is introduced. Afterwards we demonstrate our approach through a case study. Finally, conclusion and further works are given.

Metrics for Transformation Evaluation

Transformations that are commonly used in different situations in the process of software design are formalized as design patterns, which are described as a general solution to a common problem in software design [15]. Due to the common nature of design patterns, in this research we tend to use high level design transformations using GOF design patterns. Most of these patterns improve the design quality and reusability by decreasing diverse coupling metrics while increasing the cohesion.

To measure the design quality improvement, many object-oriented metrics have been introduced since the birth of object-oriented programming. These metrics are capable of measuring wide range of software properties at different abstraction levels [16]. Throughout this research we narrow our metrics to those related to measuring the reusability of design which is one of the major factors of software quality.

According to object oriented design principles, a high quality design consists of stable packages that are surrounded by other loosely coupled subsystems which could possibly minimize the propagation of software changes. Thus, software stability is

highly dependent on the coupling between components. Because none of usual metrics can represent the overall system reusability, we will use *Distance from the Main Sequence (D)* metric introduced by Robert C. Martin [17] as our main GA fitness value. The D metric ranges between [0, ~0.707] and can be calculated by:

$$D = \frac{|A + I - 1|}{\sqrt{2}} \quad (1)$$

Where:

$$I = \frac{Ce}{Ce + Ca} \quad (2)$$

$$A = \frac{AbstractClasses}{TotalClasses} \quad (3)$$

In our framework we will use the normalized version of D metric which is more convenient and ranges between [0,1]. The visual representation of this metric is available in Fig. 1. According to *D* metric, any package which stands far from the origin is unbalanced and should be reengineered in order to define it more reusable and less sensitive to changes.

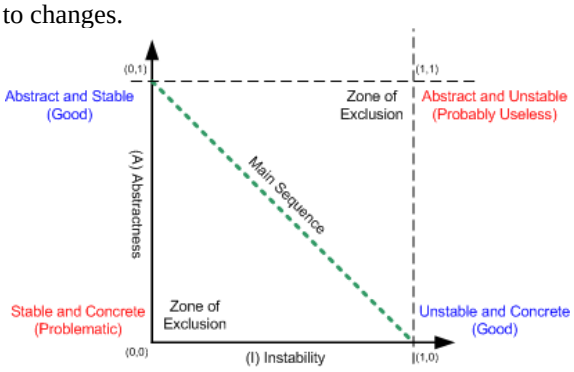

Fig. 1. Distance from the Main Sequence

Genetic Algorithm

In the context of software transformation, patterns can be applied to different classes with different parameters, making the enumeration of candidate solutions expensive. On the other hand, there is no efficient or complete solution to find the appropriate and optimum transformations. These characteristics of software transformation problem led us to use genetic algorithms as a good search-based method to handle the problem [18]. In this section we describe our formulation of our problem as a genetic algorithm:

- **Chromosomes:** The chromosomes are basically an encoding of a sequence of transformations and their parameters. Each individual consists of several supergenes, each of which represents a single transformation. A supergene is a group of neighboring genes on a chromosome which are closely dependent and are

often functionally related and certain combination of the internal genes are valid. These internal genes are the pattern number, package number, and the classes that the pattern is applied to, as illustrated in figure 2. Since each pattern has different kinds of parameters, after applying genetic operators, some invalid patterns, supergenes, may be produced. These invalid supergenes are found and discarded.

- **Fitness function:** To evaluate a chromosome, its represented transformations are applied to the original design using the *Transformer Engine*. The *Transformer Engine* transforms the design according to the data encoded in the chromosome, then the D metric is evaluated using the *Design Metrics Evaluator* engine and is set as the fitness value of given individual. In case of encountering a transformed design which contradicts with object oriented concepts, for example a cyclic inheritance, a zero fitness value is assigned to the chromosome.
- **Crossover:** We designed two different crossover operators which can be applied either simultaneously or separately. The first crossover is a single-point crossover with a randomly selected point applied at supergene level. This crossover swaps the supergenes beyond the crossover point, but the internal genes of supergenes remain unchanged. The second crossover operator randomly selects two supergenes from two parent chromosomes, and similarly applies single point crossover to the genes inside the supergenes. Each of these crossovers has different effects. The first one combines the promising patterns of two different transformation sequences, while the second one combines the parameters of two successfully applied patterns.
- **Mutation:** The mutation operator randomly selects a supergene and mutates a random number of genes, inside that supergene. After applying the mutation the validity of the supergene should be checked.

Fig. 2. Supergene Representation

Framework

Our developed framework, *Automatic Design Enhancer (ADE)*, designed to automate the whole process, consists of three major subsystems. The block diagram of ADE is illustrated in Fig. 3.

Design Transformer engine is based on an open-source project DPA tool [19]. It can reverse engineer the software program and extract its design. DPA can also get the required design model directly from an XML. The main feature of DPA is the facility to apply design patterns to the object-oriented design of the project. To do so, the developer can use the existing design pattern plug-in or create a custom plug-in.

To evaluate the transformed design, generated by DPA tool, with object-oriented design metrics, we use RefactorIT tool [20] as our *Design Metrics Evaluator* engine. Finally we put them all together using JGAP GA toolkit [21] as our *Genetic Algorithm* engine.

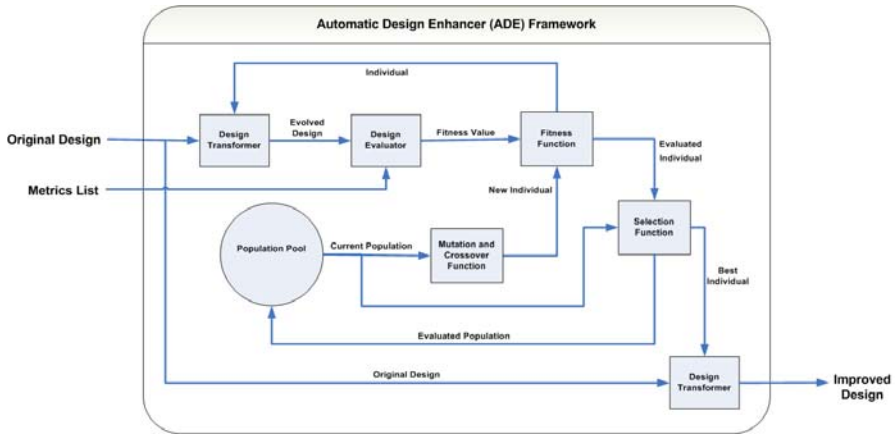

Fig. 3. Automatic Design Enhancer (ADE) Framework Schema

Case Study

In order to test our framework, we extracted the UML design of some free, open source applications. Then we executed our framework with genetic algorithm in two versions. In one version only the first crossover operator was applied but in the other both operators was used. Figure 4 and 5 illustrates the increase in fitness value –D metric- for both runs, tested on an application called Gold Parser [22], which consists of 9 packages and approximately 100 classes. Also, we used random search in similar manner to find out if the genetic algorithm performs out the random search. The results demonstrate that the genetic algorithm finds the optimal solution much more efficiently and accurately. Table 1 compares genetic algorithm and random search.

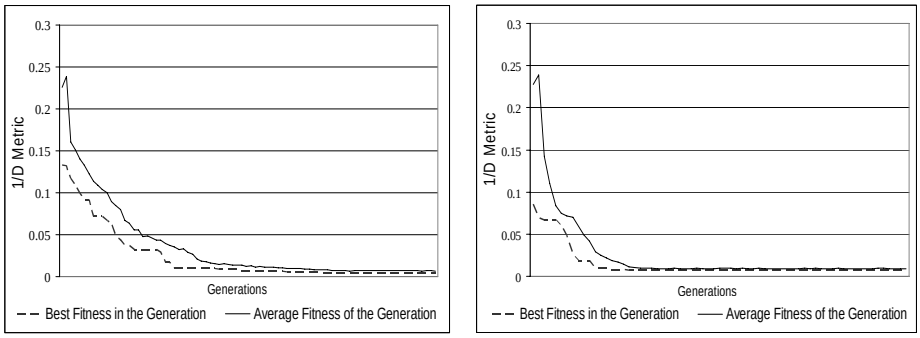

Fig. 4. Automatic Design Enhancer (ADE) Framework Schema

From software design perspective, the transformed design of the best chromosomes are evolved such that abstract packages become more abstract and concrete packages

turn to become more concrete. This is the nature of D metric and reduces the coupling between concrete packages.

Table 1. Different initial test conditions for GOLD case study

	Number of Generations/Searches	Average Population Size	Best Metric Value	Average Metric Value	Percentage of Valid Transformations
Random Search	10000	N/A	0.06	0.41	44.5%
GA - 1 Crossover	100	82	0.02	0.04	94.0%
GA - 2 Crossovers	100	105	0.01	0.03	95.1%

Conclusion and Future Works

This research introduces the possibility of completely automated design evolution using search-based methods especially genetic algorithms. It uses the pool of high level transformations to achieve the required design quality improvement.

Investigation on the results of the whole evolution process make us conclude that search based algorithms, particularly genetic algorithm, are helpful in improving special design metrics.

Further researches may include developing other and more complex crossover, mutation and selection algorithms. Also more intelligent crossover functions with memory can be implemented using reinforcement learning approaches. Besides, different pattern and meta-pattern pools may be developed to support transformations. Also further improvements may be considered to ensure the overall quality of software by using data fusion approaches on design metrics.

Finally, the whole process might be applied on design models with different applications. It is clear that different metrics are appropriate for different applications according to the nature of those programs.

References

1. Coleman D. M., Ash D., Lowther B., Oman P. W.: Using Metrics to Evaluate Software System Maintainability: IEEE Computer, Vol. 27, No. 8, August (1994) 44–49,
2. Guimaraes T., Managing Application Program Maintenance Expenditure Comm. ACM, Vol. 26, no. 10, (1983) 739–746
3. Lientz B. P., Swanson E. B.: Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations, Addison-Wesley, (1980)
4. Men T., Tourwe T., A Survay of Software Refactoring: IEEE Transaction on Software Engineering, Vol 30, No. 2, Feb (2004) 126-139
5. Fenton N., Pfleeger S. L.: A Rigorous and Practical Approach: International Thomson Computer Press, London, UK, second edition, (1997)
6. Chidamber S. R., Kemerer. C. F.: A Metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering, 20(6), June (1994) 476–493

7. Lorenz M., Kidd J., Object-Oriented Software Metrics: A Practical Approach. Prentice-Hall, (1994)
8. Marinescu R.: Using Object-Oriented Metrics for Automatic Design Flaws in Large Scale Systems: In S. Demeyer and J. Bosch, editors, Object-Oriented Technology (ECOOP'98 Workshop Reader), LNCS 1543,. Springer-Verlag, (1998) 252–253
9. Nesi P.: Managing OO Projects Better. IEEE Software, July (1988)
10. Opdyke W. : Refactoring Object-Oriented Frameworks. PhD thesis, University of Illinois at Urbana-Champaign, (1992)
11. Fowler M., Refactoring: Improving the Design of Existing Programs, Addison-Wesley, (1999)
12. Tokuda L., Batory D. S.: Evolving Object-Oriented Designs with Refactorings," Automated Software Engineering, Vol. 8, No. 1, (2001) 89–120
13. Tahvildari L., Kontogiannis K.: Improving Design Quality Using Meta-pattern Transformations: A Metric-based Approach: Journal of Software Maintenance and Evolution, Vol.16, (2004) 331-361
14. Fatiregun D., Harman M., Hierons R. M.: Evolving Transformation Sequences using Genetic Algorithms: Source Code Analysis and Manipulation, Fourth IEEE International Workshop on (SCAM'04) 66-75
15. Gamma E, Helm R., Johnson R., Vlissides, J. Design Patterns:Elements of Reusable Object-Oriented Software, Addison-Wesley: Reading MA, 1995
16. T. Mens and S. Demeyer. Evolution metrics. In Proc. Int. Workshop on Principles of SoftwareEvolution, 2001.
17. Martin, R.C. Design Principles and Design Patterns, http://www.objectmentor.com, 2000.
18. Clarke J., Harman M., et al: Reformulating Software Engineering as a Search Problem: Journal of IEE Proceedings - Software, (2003) 161–175
19. Design Pattern Automation Toolkit: http://sourceforge.net/projects/dpatoolkit/
20. refactorIT: www.refactorit.com/
21. JGAP: http://jgap.sourcefoge.net/
22. Gold Parser: http://www.devincook.com/goldparser/engine/dot-net.htm