International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 19 (2017) pp. 8518-8522
© Research India Publications. http://www.ripublication.com

# Detection a Design Pattern through Merge Static and Dynamic Analysis Using Altova and Lambdes Tools

**Hamed J. Fawareh**
*Department of Software Engineering ,Zarqa University,Jaber Highway, Zarqa, Jordan.*

*Orcid ID: 0000-0002-0853-3149*

**Mohammad Alshira'h**
*Al albayte university, Mafraq Jordan.*

**Abstract**

Understanding the legacy systems and its changed requirements is the main problem in software process. The legacy systems must be maintain to meet the needs of new computing environments or technology, and must be enhanced to implement new business request or to make it interoperable with more modern system or databases. Reverse engineering is the main idea in maintaining legacy systems throughout understanding the source code. This paper focused on developing an approach for merging static and dynamic analysis using Altova and LAMBDES tools. In addition to developed an automated tool for integrated the static and dynamic approach in one merged file. The approach used the new XMI file for modifying the legacy system requirement by extract the pattern. Then the tool allows the user to modify the requirement in graphical representation.

**Keyword**: Reverse Engineering, Legacy System, Design Pattern

## INTRODUCTION

Software systems have been applied in many difficult and complex applications, from different environments. Each software system may contain thousands of source code lines, a fact which makes it difficult to manually walk through these software without aids tools. This problem becomes even more complicated when the developer uses a large software system. Several approaches had been developed in design pattern for legacy system. Erdos and Sneed (Erdos, and Sneed) suggest partial comprehension of complex programs. The approach contends that maintenance tasks require the comprehension of a relatively small portion of the program. This is done by developing an automated tool. This tool answers a set of programmer question automatically. This approach permits unfamiliar programmers with the purpose and function of the programs during maintenance. The approach used Fan-in diagram which is used along with Low level Data Flow Diagrams. Decision Trees are used to model complex conditional series of statements.

The supposed ease of comprehension of object-oriented programs is squarely denounced by (Sneed and Dombovari). Their paper deals with an ongoing research project that aims at the difficult task of comprehending complex, distributed, object-oriented software systems by approaching in a formal disciplined manner. Citing contemporary work in initiatives, the paper goes on to explain that if modeled properly and if similar supported by automated tools, even complex, object oriented systems can be comprehended formally. This approach also places emphasis on reverse engineering required only to the extent of maintaining software.

Mayrhauser and Vans approach is used for large scale programs. The approach reports on a software understanding study during adaptation of large-scale software. The study was designed as an observational field study of professional maintenance programmers adapting software. The approach details the design of the study and discusses the results from the programmers. The goal was to answer several questions about how programmers approach software adaptations, their work process and their information needs. The programmers were found to work predominantly at the domain model level, adopting opportunistic and systematic understanding. A report on the general understanding process, the type of action programmers performed during the adaptation task, and the level of abstraction at which they work is included.

Antoniol et al present **a**n approach to recover object-oriented design patterns from the design and code (Antoniol et. al.). Design patterns are micro-architectures, high level building blocks. Design patterns are an emergent technology: they represent well-known solutions to common design problems in a given context. From the perspective of reverse engineering the discovery of patterns in software artifacts represents a step in the program understanding process. A pattern provides knowledge about the role of each class within the pattern, the reason for certain relationships among pattern constituents and/or the remaining parts of the system. Design patterns being a relatively young filed, there are currently few works that address design pattern recovery in the field of program understanding and design recovery.

A pattern description encompasses its static structure, in terms of classes and objects participating to the pattern and their relationships, but also behavioral pattern dynamics, in terms of participants exchanged messages. Five specific design patterns suggested in previous literature are chosen as samples for recovery.

Abd-El- Hafiz evaluates knowledge-based approach to achieve program comprehension. The approach mechanically documents programs by generating first order predicate logic annotations of their loops. A family of analysis techniques has been developed to cover different levels of program complexity. The knowledge based approach exploits the fact that there are certain stereotyped programming concepts that are heavily used in programs and detecting these can be easy using this approach. An attempt is made to prove that the knowledge base built using a specific program can help in understanding similar stereotyped programming constructs in other programs. The approach can be greatly enhanced by trying to create knowledge bases that are sufficient for specific application domains.

DeBaud et al. contend that instead of the current reverse engineering techniques that takes a program and constructs a high level representation by analyzing the lexical, syntactic and semantic rules, an approach that utilizes the relationship between the application domain analysis and reverse engineering can be used. A domain is a problem area and domain analysis is an attempt to identify the objects, operators, and relationships between what domain experts perceive to be important about the domain. A domain description will give the reverse engineer a set of expected constructs to look for in the code.

Another research trend of reverse engineering is design pattern recognition from source code. A design pattern (Gamma et. al.) is a reusable object oriented software design artifacts that solves a problem in particular context. Design patterns in an architecture making faster the understanding the design considerations of a software system. There are several different approaches to identify patterns in source code, design patterns can be identified by among others inter-class relationship in method call, data-flow analysis, by fuzzy logic, graph matching or formal semantic (Hamed 2015).

Pattern recognition is also suitable for measuring software quality (Brown et. al.), because not only design patterns, but also anti-patterns (Beyer and Lewerentz) can be detected in the implementation, thus, bad design considerations or weakness of the code can be discovered. Similarly to design patterns, anti-patterns are piece of reusable code, but applying these kinds of patterns shuld be avoided. CrocoPat e. al. tool does graph search, it processes RSF (Rigi Standard Format) files that contains the graph of a system that uses own imperative language to find the predefined patterns between class inheritance relations and method calls. Columbus uses graph matching algorithms. Other methods are also available, such as PtideJ, which uses constraint solving or SPOOL (Nija and Olsson), which uses database query. PINOT (Hakjin et. al.) pattern inference and recovery tool reclassifies the GoF patterns and implements a lightweight static inter-class and data-flow analysis.

## RELATED WORK

In Jing Dong et al, present an approach to discover design patterns by defining the structural characteristics of each design pattern in terms of weight and matrix. The system structure is represented in a matrix with the columns and rows to cover classes in the system. The value of each cell represents the relationships among the classes. The structure of each design pattern is also represented in another matrix. The discovery of design patterns from source code becomes matching between the two matrices. If the pattern matrix matches the system matrix, a candidate instance of the pattern is found. Also, they use weight to represent the attributes and operations of each class and its relationships with other classes. In addition to the structural aspect, the approach investigates the behavioral and semantic aspects of pattern discovery. The approach consists of three phases: structural, behavioral, and semantic analyses. The structural analysis phase concentrates on the structural characteristics of the system, such as classes and their relationships. The results of the structural analysis may include the detected instances that are actually not a design pattern. Although such instances satisfy the structural characteristics of a design pattern, they may not be the instances of such design pattern due to missing behavioral characteristics. Behavioral analysis checks the results from the structural analysis for false positives. In the semantic analysis there are certain closely related design patterns which are similar with respect to their structural and behavioral aspects but just different by their intent with which they were created. The approach is includes several analysis phases and based on matrix and weight to discover design patterns from source code. They need many steps and some time they repeat the same steps in behavioral analysis and need to execute the code, which is a time consuming step. And the behavioral patterns can't be detected in structural and behavioral phase, also, semantic analysis.

In Hakjin Lee et al proposed taxonomy of GoF design patterns that can guide the reverse engineering process. The approach applies a number of existing applications, such as PURE toolkit, JINI based home application system. The approach shows that using a static analysis only is very difficult to distinguish pattern among similar structure with high false-positive rate. Furthermore the approach shows that using dynamic analysis only needs too many searching space to read source code and requires the well-arranged testing environment. According to the inputs of the reverse process used in this approach source code is read first, and the next step is detect a static analysis of source code to generate the structure candidate instances. The next step is detects a dynamic analysis. These steps require more time because firstly, they apply static analysis, after that both static and

dynamic analysis is applied to detect the behavioral patterns. Also, this process make the approach is difficult to integrate with other tools.

In Grose et al, they detect design patterns in legacy code combining static and dynamic analyses with required method. This approach analyses distinguishes between static and dynamic pattern restrictions or rules. The former restrict the code structure the latter the runtime behavior. Analyzing with the static restrictions, results in a set of candidate occurrences in the code. In practice this set is large and programmers hardly want to screen all of them to detect the actual instances. Therefore, they execute the program under investigation and monitor the executions of the candidate instances found by the static analysis with respect to the dynamic restrictions. The results of dynamic analyses depend on an execution of the candidate instances. The static analysis computes potential program parts playing a certain role in a design pattern. The dynamic analysis further examines those candidates. In this approach the detection process is separated into two step , static analysis detection , and dynamic analysis detection with the need to executing the source code to detect the behavioral aspects of the source code, which is time consuming process because the two steps of detection, other limitation appears after the detecting of design patterns. They don't use the detected result in reengineering cycle, or benefits from these results.

## APPROACH TAKEN

Reverse engineering aims to provide program descriptions on higher levels of abstractions, such an abstract level could be a program description using UML diagrams. These program descriptions facilitate the understanding of program structures and program behavior. This paper presents the Detection of Design Pattern through merge static and dynamic analysis (D2Pattern) as a proposed approach. The approach is shown in figure 1. The approach organized into five phases; the first two phases are generating a static and dynamic analysis. Altova Umodel 2010 is used in the first two phase to extract class diagrams and sequence diagrams of the design patterns respectively. Then, the third phase combines the static and dynamic furthermore the tool automatic generate an XMI file. In addition in this phase the tool verify the design pattern candidates that found during the static and dynamic analysis. The fourth phase is design pattern detection and classification using automated software tool called LAMBDES. In this phase the XMI file which contains the class and the sequence is used to process the necessary information for detecting the design pattern. The design pattern is used during understanding a legacy system requirement. The fifth phase is integrated result with legacy requirement which supported maintenance phase. In addition, the fifth phase recovers requirements from the reversed design patterns and integrating the patterns in requirements phase.

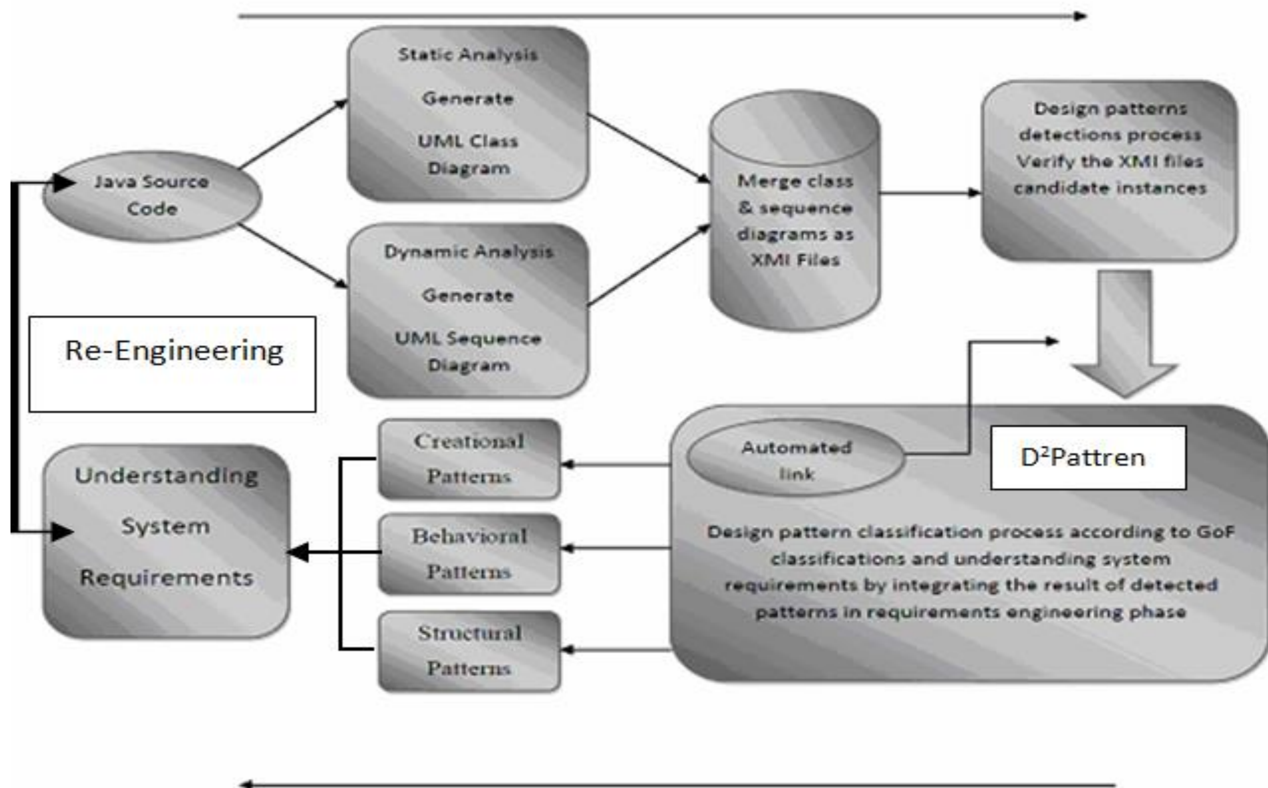


**Figure 1:** $D^2$Pattern approach

The result of the static analysis and dynamic analysis is merged and stored as XMI file, to be input in the detection phase, design pattern detected using LAMBDES. The tool build a pattern based on descriptive semantics a number of generators and a repository of design pattern specifications. It takes XMI file produced by Altova Umodel 2010 as input to perform logical analysis of the model and/or Metamodels. LAMBDES system translates the UML diagrams into their descriptive semantics and to decide whether the design conforms to a pattern. In addition facilitates reasoning about models through logical inference.

When the results of detect pattern is fully detected, we classify it according to GoF classification to creational patterns, behavioral patterns, and structural patterns as shown in figure 1. The classification output allows maintainer to add any new requirements easily by insert the code content into a class model. The enhancement based on the new requirements. For example, the new requirements concern on enhancing the graphical user interface, then, the requirements engineer must do his/her modification in the structural patterns.

## RESULT AND DISCUSSION

To evaluate the proposed approach we build a prototype system as shown in figure 1. the proposed system integrated Altova Umodel 2010 and LAMBDES tools in one system. The prototype firstly read the source code, also it perform a static and dynamic analysis; the result is used in detecting design patterns. The new system helps in understanding the legacy requirements.

The adapter design pattern used to demonstrate each stage of our approach. As a case study the new approach reads the a java code as a first step then generate class and sequence diagrams using Altova Umodel 2010 LAMBDES tools. During static analysis the source code is analyzed. Then the system extracts a class diagram and its relationships among the components were visually represented by a dependency relationship between them. Major packages were also identified in these diagrams; furthermore the graphical view is represented for simplify the viewer's time and effort to understand the architectural layout of the software.

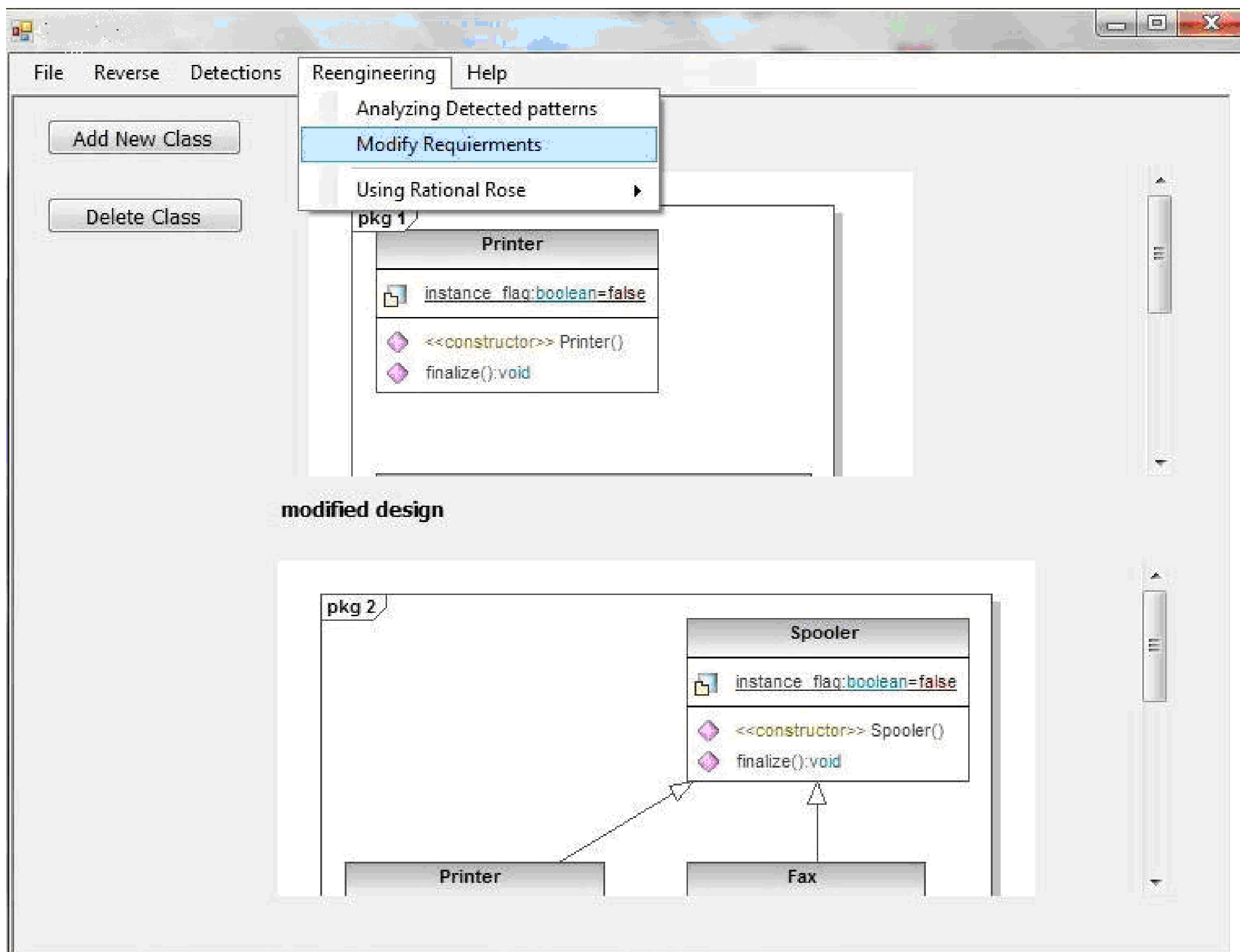


**Figure 2:** System Prototype

The proposed approach recovers requirements from the reversed design patterns. This done by integrated the new modification and enhancement requirement with the legacy code. Integration process is automatically done by this approach. The classification result by the prototype shown in figure 2. The result helps maintainer to answer question such as: what is redundant, what must be retained and what can be re-used. The new approach shows rationale of the necessity of eliciting requirements and proposed a modified model of existing model.

## REFERENCES


[1] Erdos K., H.M. Sneed, (1998), Partial Comprehension of Complex Programs (enough to perform maintenance)," *IEEE Proceedings - Sixth International Workshop on Program Comprehension, J*une 24 – 26,.

[2] Sneed H.M., T. Donbovari, (1999), Comprehending a Complex, Distributed, Object oriented, *IWPC '99 Proceedings of the 7th International Workshop on Program Comprehension IEEE* Computer Society Washington, DC, USA

[3] Mayrhauser, A. M. Vans, (1998), Program Understanding Behavior During the Adaptation of Large Scale Software, *IEEE Proceedings - Sixth International Workshop on Program Comprehension,* pp. 164-172, June 24 – 26,.

[4] Antoniol G, R. Fiutem, L. Cristoforetti, (1998), Design Pattern Recovery in Object Oriented Software, *IEEE Proceedings - Sixth International Workshop on Program Comprehension*, June 24 – 26, pp. 153-160,.

[5] Abd-El-Hafiz S.K., (1996), Evaluation of a Knowledge based approach to Program Understanding, *IEEE Proceedings – Working Conference in Reverse Engineering*, '96," pp. 259 – 269,.

[6] Burnstein I., F. Saner, (1999), An Application of Fuzzy Reasoning to Support Automated Program Comprehension, IEEE Proceedings - Seventh International Workshop on Program Comprehension, pp. 66-73, 5-7 May.

[7] DeBaud J.M., B. Moopen, S. Rugaber, Domain Analysis and Reverse Engineering," http://www.cc.gatech.edu/reverse/papers.html, College of Computing, Georgia Institute of Technology.

[8] Gamma E, R. Helm, R. Johnson, and J. Vlissides, (1994), Design Patterns: Elements of Reusable Object-Oriented Software, *Addison Wesley, 1st editions*

[9] Brown W. J., R. C. Malveau, H. W. McCormick III, T. J. Mowbray: (1998), AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis", *New York,John Wiley* and Sons, Inc.,

[10] Beyer D., C. Lewerentz: CrocoPat, (2003), Efficient pattern analysis in object-oriented programs, *In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC 2003)*, pp. 294-295, IEEE Computer Society,

[11] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, Narendra Jussien, (2001), Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together, *16th IEEE conference on Automated Software Engineering (ASE'01),*

[12] Keller R. K., R. Schauer, S. Robitaille, P. Page, (1999), *Pattern-based Reverse-Engineering of Design Component*s, In Proc. ICSE, pp. 226-235, ACM.

[13] Nija Shi, Ronald A. Olsson, (2006), Reverse Engineering of Design Patterns from Java Source Code, *ase, pp. 123-134, 21st IEEE International Conference on Automated Software Engineering (ASE'06).*

[14] Hakjin Lee, Hyunsang Youn, Eunseok Lee, (2008), A Design Pattern Detection Technique that Aids Reverse Engineering, *International Journal of Security and its Applications Vol. 2, No. 1,* January, 2008

[15] Altova UModel (2010), UML tool for software modeling and application development http://www.altova.com/umodel.html

[16] Hong Zhu Bayley, I. Lijun Shan Amphlett, R., (2009), tool support for design pattern recognition at model level, *33rd Annual IEEE International Computer Software and Applications Conference*, Volume: 1,

[17] Jing Dong, Dushyant S. Lad, Yajing Zhao], (2007), DP-Miner: Design Pattern Discovery Using Matrix, *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)* 0-7695-2772-8/07.

[18] Hakjin Lee, Hyunsang Youn, Eunseok Lee,(2008), A Design Pattern Detection Technique that Aids Reverse Engineering, *International Journal of Security and its Applications* Vol. 2, No. 1,

[19] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, Welf Lowe, (2003), Automatic Design Pattern Detection, *Proceedings of the 11 th IEEE International Workshop on Program Comprehension (IWPC’03)*1092-8138/03,2003

[20] Hamed Fawareh, Reverse Engineering Model from Object-Oriented Programs Using Concept Lattice Egyptian Computer Science Journal Vol. 39 No. 4 September 2015, ISSN-1110-2586