www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3, Issue 10 October, 2014 Page No. 8920-8925

Design Recovery of Student Information Legacy System

[1]Bello AlhajiBuhari, [2]Abba Almu

[1]Usmanu Danfodiyo University/Department of Mathematics, Sokoto, 234, Nigeria
Email: buhari.bello@udusok.edu.ng

[2]Usmanu Danfodiyo University/Department of Mathematics, Sokoto, 234, Nigeria
Email: almu.abba@udusok.edu.ng

Abstract: *Reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural, and procedural design information from an existing program. This paper explores the application of reverse engineering in recovering the design of a legacy student information system developed using Dbase V atUsmanuDanfodiyo University Sokoto using UML based approach. Use case model is used in recovering the design specifications (i.e., functionalities)of the student information system. In addition,object oriented design model for the system is proposed using class diagramso that the system can be implemented using object oriented programming.*

Keywords:reverse engineering, software engineering, design recovery, use case diagram, class diagram, student information system, legacy system.

1 Introduction

"Reverse engineering" has its origins in the analysis of hardware for commercial or militaryadvantage [1].A company takes to pieces a competitive hardware product in an effort to recognize its competitor's design and manufacturing "secrets." These secrets could be easily understood if the competitor's design and manufacturing specifications were found. But these documents are exclusive and unavailable to the company doing the reverse engineering. In essence, successful reverse engineering develops one or more design and manufacturing specifications for a product by examining actual specimens of the product.

Reverse engineering for software is quite similar. In most cases, however, the program to be reverse engineered is not a competitor's. Rather, it is the company's possess work (often done many years earlier). The "secrets" to be understood are unclear because no specification was ever developed.

Therefore, reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery.

The Unified Modeling Language (UML) [2] hasestablished itself in software industry for describingsoftware models. UML-based software design process relying on two subprocesses: *reverse engineering* and *model analysis.*

Reverse engineering combines a top-down reverse engineeringtechnique with traditional bottom-up reverse engineeringactivities.

In this paper, a reverse engineering method called design recovery is employed to recover the design specification of Student Information legacy system of UsmanuDanfodiyo University, Sokoto. UML based approach is used as the basis for the recovery of the design. Use case model is the UML model used to represents the design graphically. After the use case model also a class diagram model is also used to represent the design using object oriented paradigm. Hence, the class diagram can be used to implement the system using object oriented programming.

2 Related Works

Reverse engineering should produce, first in an automatic way, documents that help software engineers in understanding the system. Over the last ten years, reverse engineering research has produced a number of abilities for analyzing code, including subsystem decomposition[3], concept synthesis [4], design, program and change pattern matching [5][6], analysis of static and dynamic dependencies [7], object-oriented metrics [8], documentation, maintenance,

and re-engineering [9], analysis (not modification) of an existing (software) system [10]and others. In general, these methodologies have been successful in treating the software at the syntactic level to address specific information needs and to span relatively narrow information gaps.

3 Methodology

Reverse engineering is the process of identifying software components, their interrelationships,and representing these entities at a higher level of abstraction. Reverseengineering by itself involves only analysis, not change [10]. Program comprehension and program understanding are terms often used interchangeably with reverse engineering. Four specializations of reverse engineering are offered, in increasing level of impact [11]:

- *Redocumentation:* Perhaps the weakest form of reverse engineering, this involves merely the creation (if none existed) or revision of system documentation at the same level of abstraction.
- *Design Rediscovery:* Redocuments, but uses domain knowledge and other externalinformation where possible to create a model of the system at a higher level ofabstraction.
- *Restructuring:* Lateral transformation of the system within the same level ofabstraction. Also maintains same level of functionality and semantics.
- *Reengineering:* The most radical and far reaching extension. Generallyinvolves a combination of reverse engineering for comprehension, and areapplication of forward engineering to reexamine which functionalities needto be retained, deleted or added.

This paper is using design rediscovery which both recover the design and redocments the system.

A variety of approaches for automated assistance are available for the reverse engineer in programcomprehension. A full list of reverse engineering approaches is available in [12]. Some of the more prominent approaches include:

- *Textual, lexical and syntactic analysis* - these approaches focus on the source code itself and its representations. These include the use of UNIX's lex, lexical metrics (counting assignments, identifiers, etc.) outlined in [13], and even automated parsing of the code searching for cliches [14]. Cliches are standard approaches to problem solving that can extracted from the source code to give hints about design decisions. The unit of examination is the program source itself.
- *Graphing methods* - there are many graphing approaches for programunderstanding. These include, in increasing order of complexity and richness: graphingthe control flow of the program [15], the data flow of the program [15], and programdependence graphs [16]. The unit of examination is a graphical representation of theprogram source.
- *Execution and testing* - there are a variety of methods for profiling, testing, andobserving program behavior, including actual execution and inspection walkthroughs.Dynamic testing and debugging is well known and there are several tools availablefor this function. For large systems, a technique called "partial evaluation" is availableto identify and test isolate components of a system [17]. "Abstract interpretation" isa method for using denotational semantics to perform static testing through simulatingthe behavior of the actual system [18]. The unit of examination is a full, partial, orsimulated execution of the program.

In this paper graphical method is used in form of UML. The UML used are use case diagram and class diagram.

4 Recovering the System Design

The first real reverse engineering activity begins with an attempt to understand and then extract procedural abstractions represented by the source code. To understand procedural abstractions, the code is analyzed at varying levels of abstraction: system, program, component, pattern, and statement

The overall functionality of the entire application system must be understood before more detailed reverse engineering work occurs. This establishes a context for further analysis and provides insight into interoperability issues among applications within the system. Each of the programs that make up the application system represents a functional abstraction at a high level of detail.

4.1 Creating Use Case Diagram

A *use case,* a concept invented by IvarJocbson[19], is asequence of transactions performed by a system that yields an outwardly visible, measurable result of value for a particular actor. A use case typically represents a major piece of functionality that is complete from beginning to end [20].

In UML, a use case is represented as an ellipse, as shown in Figure 4.1. In a student information system, some use cases are: Register Student, Register Course, add exam result, Create Course Report, Create Grade Sheet, Create Senate Format Report, Create Transcript, etc.

An *actor* represents whoever or whatever (person, machine, or other) interacts with thesystem. The actor is not part of the system itself and represents anyone or anything that must interact with the system.

The total set of actors in a use case model reflects everything that needs to exchangeinformation with the system [21]. In UML, an actor isrepresented as a stickman, shown below in Figure 4.1. In the student information system, actorsare the admin and staff.

There are several different kinds of relationships between actors and use cases. The default relationship is the «communicates» relationship. The«communicates» relationship indicates that one of these entities initiated invoked a request of the other. An actor communicates with use cases because actors want measurable results.

There are two other kinds of relationships between use cases (not between actors and usecases) that you might find useful. These are «include» and «extend». You use the «include» relationship when a chunk of behavior is similar across more than one use case, and you don't want to keep copying the description of that behavior [21]. This is similar to breaking out re-used functionality in a program into its

own methods that other methods invoke for the functionality. For example, since many actions of a system require the user to login to the system before the functionalitycan be performed. These use cases would *include* the login use case. The admin use case diagram is shown in figure 4.1 and the staff use case diagram is shown in figure 4.2.

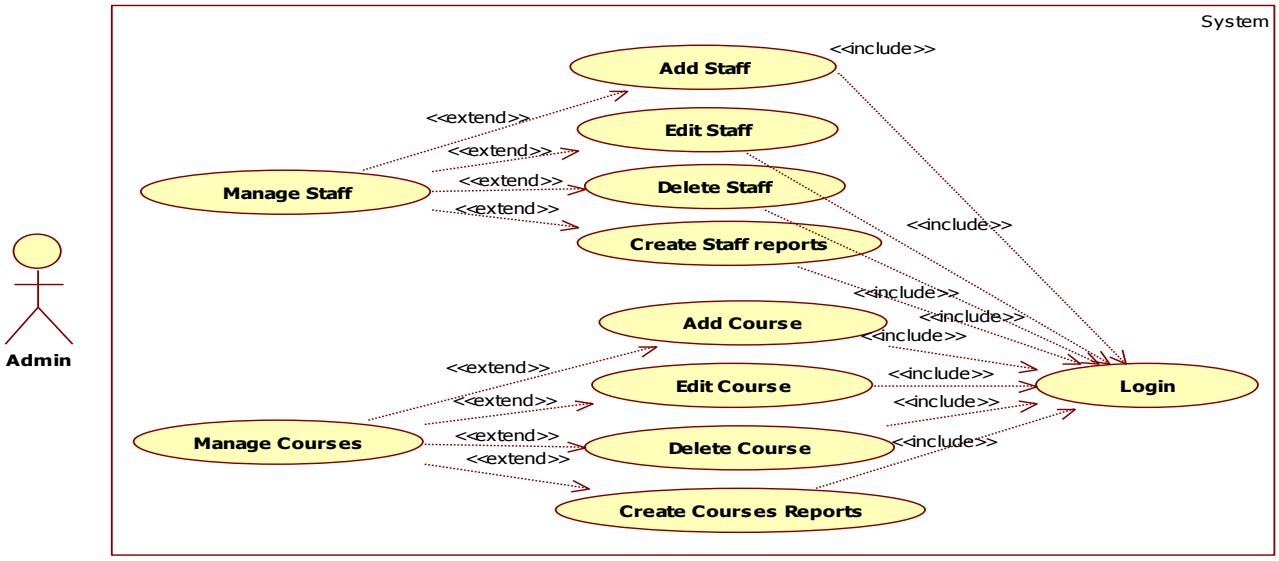

Figure 4.1: Use case diagram (admin actor) for the student information system.

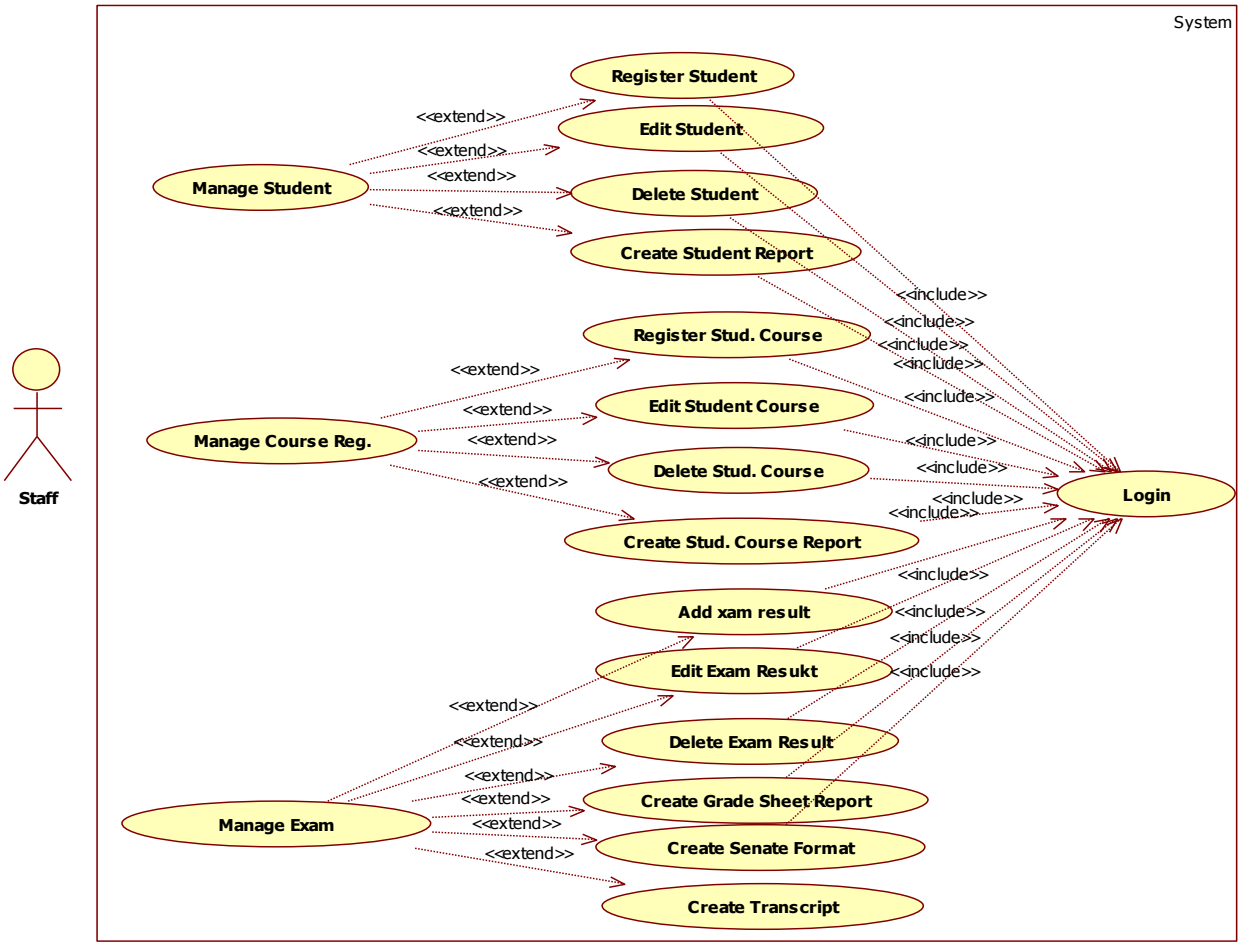

Figure 4.2: Use case diagram (staffactor) for the student information system.

The admin here is the Director MIS or Deputy Director MIS. He/she creates staff to use the system. Each staff is going to handle one faculty. In addition, admin is in charge of adding the entire courses in each department. This involves course code, course title, course unit, semester, etc.

Staffs are the MIS staff. Each staff manage students from his/her faculty, register course offered by each student per session, and enter and process their results.

5 Proposed Design

In addition to recovering the design of the legacy system, object oriented design specification is also proposed so that the system can be implemented using object oriented programming. Class diagram is used in the new design.

5.1 Creating Class Diagram

Class diagrams are used in both the analysis and the design phases. During the analysisphase, a very high-level conceptual design is created. At this time, a class diagram might be created with only the class names shown or possibly some pseudo code-like phrases may be added to describe the responsibilities of the class. The class diagram created during the analysis phase is used to describe the classes and relationships in the problem domain, but it does not suggest how the system is implemented. By the end of the design phase, class diagrams that describe how the system to be implemented should be developed. The class diagram created after the design phase has detailed implementation information, including the class names, the methods and attributes of the classes, and the relationships among classes.

The class diagram describes the types of objects in a system and the various kinds ofstatic relationships that exist among them [20]. In UML, a class is represented by a rectangle with one or more horizontal compartments. The upper compartment holds the name of the class. The name of the class is the only required field in a class diagram. By convention, the class name starts with a capital letter. The (optional) center compartment of the class rectangle holds the list of the class attributes/data members, and the (optional) lower compartment holds the list of operations/methods.

There are two principle types of static relationships between classes: inheritance andassociation. The relationships between classes are drawn on class diagram by various lines and arrows.

Inheritance (termed "*generalization*" for class diagrams) is represented with an emptyarrow, pointing from the subclass to the superclass, as shown in Figure 4.3. In this figure, StudCourse inherits from Cell (i.eStudCourse "is-a" specialized version of a Student). The subclass (StudCourse) inherits all the methods and attributes of the superclass (Student) and may override inherited methods.

An association represents a relationship between two instances of classes. An associationbetween two classes is shown by a line joining the two classes. Association indicates that one class utilizes an attribute or methods of another class. If there is no arrow on the line, the association is taken to be bi-directional, that is, both classes hold information about the other class. A unidirectional association is indicated by an arrow pointing from theobject which holds to the object that is held. There are two different specialized types ofassociation relationships: aggregation, and composition.

If the association conveys the information that one object is part of another object, buttheir lifetimes are independent (they could exist independently), this relationship is called *aggregation*. For example, we may say that "a Course contains a set of ExamResult." Where generalization can be though of as an "is-a" relationship, aggregation is often thought of as a "has-a" relationship – "a Course 'has-a' ExamResult." Aggregation is implemented by means of one class having an attribute whose type is in included class (the ExamResult class has an attribute whose type is Course).

Aggregation is stronger than association due to the special nature of the "has-a"relationship. Aggregation is unidirectional: there is a container and one or more contained objects. An aggregation relationship is indicated by placing a white diamond at the end of the association next to the aggregate class, as shown between StudCourse and ExamResult in Figure 4.3.

Even stronger than aggregation is *composition*. There is composition when an object is contained in another object, and it can exist only as long as the container exists and it only exists for the benefit of the container. Examples of composition are the relationship StudCourse, and ExamResult. An exam result can exist only for student course. Any deletion of the whole (student course) is considered to cascade to all the parts (the exam results are deleted). Composition is shown by a black diamond on the end of association next to the composite class, as shown between Student and ExamResultin Figure 4.3.

Associations have a cardinality that indicates how manyobjects of each class can legitimately be involved in a given relationship. Cardinality is expressed by the "*n..m*" symbol put near to the association line, close to the class whose cardinality in the association we want to show. Here "*n*" refers to the minimum number of class instances that may be involved in the association, and "*m*" to the maximum number of such instances. If $n = m$, only an "*n*" is shown. An optional relationship is expressed by writing "0" as the minimum number.

6 Conclusion and Future Work

Reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery.

Reverse engineering method called design recovery was employed to recover the design specification of Student Information legacy system of UsmanuDanfodiyo University, Sokoto using UML based approach. Use case model is the UML model used to represents the design graphically. In

addition to design recovery an objected object oriented design was proposed using class diagram. Hence, the class diagram can be used to implement the system using object oriented programming.

We also intended to model this student information systemusingother UML models like sequence diagram,

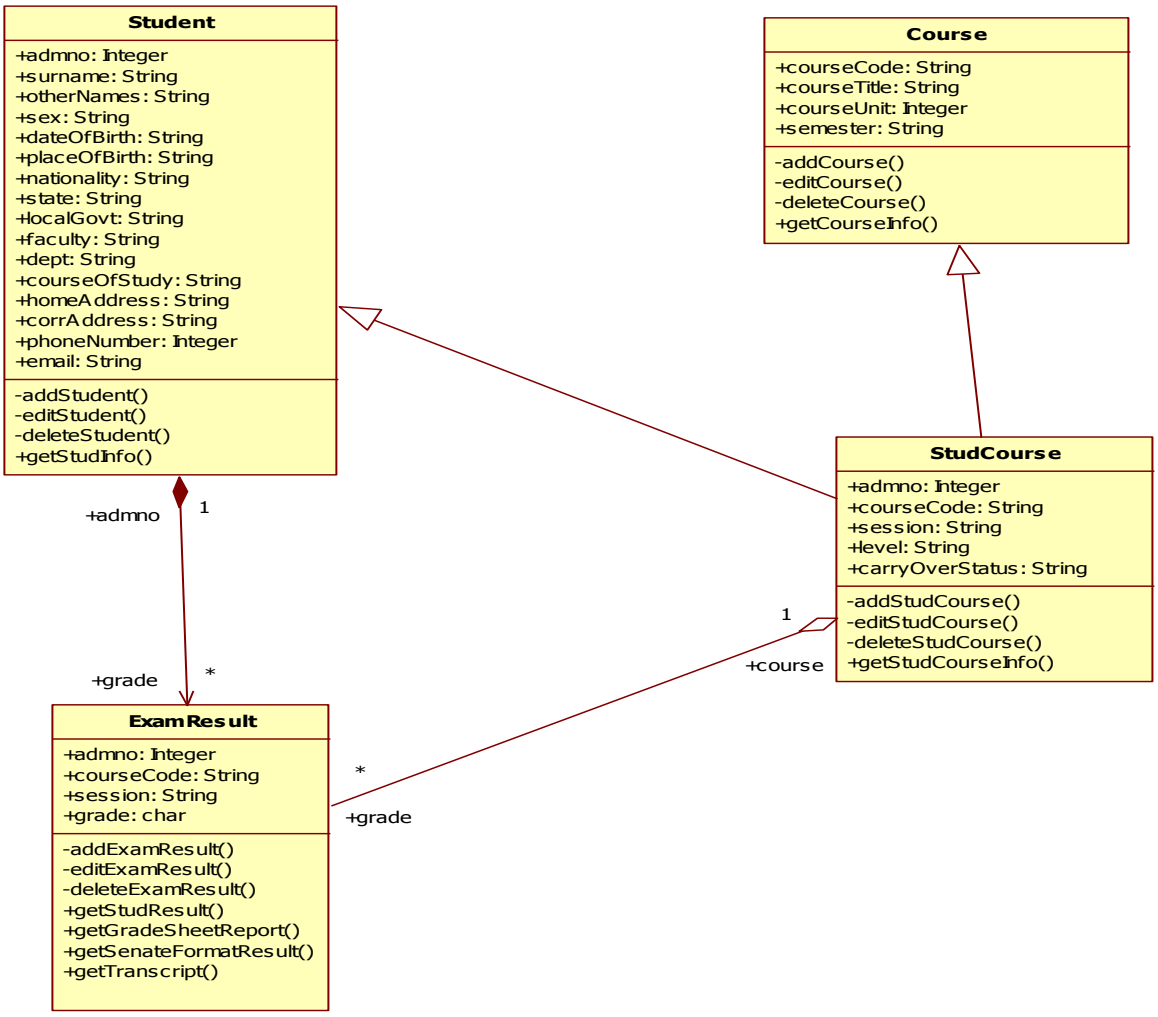

Figure 4.2: Class diagram for the proposed student information system design.

activity diagram, state chart diagram, etc to recover the design in order to have in-depth documentation of the system. Also in this system the students does not directly interact with system so another design can be made such that student did both student registration and course registration them selves. That the system to be an online system.

References

[1] E. J. Chikofsky and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*,vol. 7, no. 1, pp. 13-17, January 1990.

[2] The Object Management Group, Unified ModelingLanguage Specification (Action Semantics) – UML 1.4 withAction Semantics, Final Adopted Specification, January 2002.On-line at http://www.omg.org/uml.

[3] Umar, A.: "Application (Re) Engineering: Building Web-Based Applications and Dealing with Legacies". *Prentice Hall*, Upper Saddle River,NJ, 1997.

[4] Biggerstaff, T. J. et al.: "Program understanding and the concept assignment problem". In: Proceedings of the15nd International Conference on Software Engineering (ICSE), pp. 482-498.ACM Press, 1993.

[5] Gamma, E. et al.: "Design Patterns - Elements of Reusable Object Oriented Software". Addison WesleyProfessional Computing Series.Addison-Wesley, 1995.

[6] Stevens, P. and Pooley, R.: "Systems reengineering patterns". In: *Proceedings of* the ACM SIGSOFT 6th International Symposium on the Foundations of Software Engineering (FSE), Vol. 23, No. 06, Software Engineering Notes, pp. 17–23, November, 1998.

[7] Systa, T.: "The relationships between static and dynamic models in reverse engineering java software". In: *Proceedings of the 6th Working*Conference on Reverse

Engineering (WCRE). IEEE Computer Society Press,October 1999.
[8] Chidamber, S. R. and Kemerer, C. F.: "A metrics suite for object Oriented design". In: IEEE Transaction on Software Engineering, Vol.20, No. 06, June, 1994, pp. 476–493.
[9] E. Stroulia, M. El-ramly, P. I. & Sorenson, P.: User interface reverse engineering in support of interface migration to the web, *Automated Software Engineering.* 2003
[10] Müller, H. A. &Kienle, H. M.: *Encyclopedia of Software Engineering*, Taylor &Francis, chapter Reverse Engineering, pp. 1016–1030.http://www.tandfonline.com/doi/abs/10.1081/E-ESE-120044308.2010
[11] Michael L. Nelson, "A Survey of Reverse Engineering and Program Comprehension", 1996. (re-issued in 2005 as arxiv.org technical report cs/0503068).
[12] S. Rugaber, "Program Comprehension," *Encyclopedia of Computer Science and Technology*, Draft -- to appear,April, 1995.
[13] Maurice H. Halstead, "Elements of Software Science," Elsevier, 1977.
[14] Linda M. Wills, "Using Attributed Flow Graph Parsing to Recognize Programs," Workshop on Graph Grammarsand Their Application to Computer Science, Williamsburg, Virginia, November 1994.
[15] M. S. Hecht, "Flow Analysis of Computer Programs," North Holland, 1977.
[16] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The Program Dependence Graph and its Use in Optimization,"ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, July 1987, pp. 319-349.
[17] F. G. Pagan, "Partial Computation and the Construction of Language Processors," Prentice Hall, 1991.
[18] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs byConstruction of Appropximation of Fixpoints," Fourth Annual ACM Symposium on Principles of Programming Languages,Los Angeles, CA, January, 1977, pp. 238-252.
[19] Jacobson, I., M. Christerson, et al. (1992). Object-Oriented Software Engineering: A UseCase Driven Approach. Wokingham, England, Addison-Wesley.
[20] Bruegge, B. and A. H. Dutoit (2000). Object-Oriented Software Engineering:Conquering Complex and Changing Systems. Upper Saddle River, NJ, PrenticeHall.
[21] Rosenberg, D. and K. Scott (1999). Use Case Driven Object Modeling with UML: APractical Approach. Reading, Massachusetts, Addison-Wesley.
[22]