역공학을 이용한 소프트웨어 재사용 시스템에 관한 연구

최 은 만[†]

요 약

소프트웨어 재사용 기법은 기존의 시스템을 개발하는데 사용된 다양한 형태의 정보와 지식을 다른 시스템 개발에 재적용함으로써 생산성을 향상할 수 있고 유지보수를 쉽게 할 수 있다. 본 논문에서는 C 및 C++로 개발된 원시 코드를 대상으로 역공학을 이용하여 재사용 가능한 부품을 추출하고 이 부품을 저장 및 검색, 합성하여 사용할 수 있는 재사용 시스템 CSORUS(C and C++ SOurce ReUse System)를 설계 및 구현하였다. 역공학을 이용하여 재사용 부품을 구축할 경우 실무 분야에서 적용되어 정확성을 검증받은 신뢰도가 높은 부품이므로 재사용 부품의 신뢰도와 새로운 시스템의 유지보수를 쉽게 할 수 있다는 장점이 있다.

A Study on Software Reuse System Using Reverse Engineering

Eun Man Choi[†]

ABSTRACT

Software reuse techniques make reapplication of various well-organized information and knowledge to system development so that improve productivity and make it easy to maintain software. This paper describes the design and implementation of CSORUS(C and C++ SOurce ReUse System) which can extract reuse components using reverse engineering, and store, retrieve, merge them written with C or C++ programming language. The construction of reuse components using reverse engineering has advantage in software quality assurance because they are reliable components already tested in real environments.

1. Introduction

The technical advances of the decades allowed for the development of larger and more complex software systems that was possible in the past. Application needs software advances. Software advances foster demand for more software. However, software supply is not satisfying explosively increasing demand for software. There have been many technical and managerial attempts to improve software productivity. For instance, standards for software components, CASE, version control technique, reverse engineering, software reuse, etc. Object-oriented paradigm has established the foundation of software reuse and has accelerated the application of software reuse techniques.

Software reuse is a technology that reapplies information and knowledge such as design, design components, source code, documents in already developed systems to a new application in same domain[1]. The reuse of proved software components makes new systems reliable and amplifies programmer's productiv-

† 정 회 원:동국대학교 컴퓨터공학과

논문접수:1996년 1월 25일, 심사완료:1996년 9월 17일

ity. It results in less time spent on analysis and design phase and in less efforts in maintenance. Reuse improves the software quality from developing software based on well designed, documented, tested, and certified components.

Even through software reuse promises less cost and fewer defects, this technology does not prevail in software industry. What makes reusing software artifacts difficult? Reuse workshop[2] concluded that non-technical reasons inhibit widespread reuse. There was no motivation to salvage and reuse software artifacts. Reuse is like a saving account. We have to put a lot in before we get anything out. However, many software organizations are hesitant to invest the time and money needed to convert old code and design information into reusable components. It is hard to manually investigate all possible reusable systems, find out the reusable component, and construct a reuse library. In this paper, we suggest a tool and method to extract, restore, retrieve and synthesize reusable components from the source code in C or C++ by using reverse engineering technique.

To make a reuse system, we studied a general characteristics of reusable components. The extracted candidate of reusable components is measured and assessed by reuse metrics including complexity, regularity, reuse frequency, class attributes. The reuse system also provides retrieval mechanism to reduce cognitive distance for passive reuse approach. The reuse components are generic and combined to produce new target systems. Therefore, individual programmers must be aware of existing components, where they are, and how to use them.

2. Software Reuse

Software reuse reapplies source code, design components, documents to new development with minimum changes. New target system inherits accuracy, maintainability, portability, reliability, and other attributes of executed system. It realize gains in productivity and improvements in quality.

Major topics of the research on software reuse are method to pick up reusable components, classify them, construct a reuse library, and retrieve appropriate reuse components. There have been other researches on characteristics of reusable components, guideline to design reusable components, and domain analysis for reusability.

Biggerstaff[1] classified the reuse technology into two major groups that depend on the nature of the components being reused. Two major approaches are composition technology with building blocks, and generation technology with patterns. The first approach is characterized by the fact that the components stored in library are largely atomic, and ideally are unchanged in their use. Therefore, this is a sort of passive way reuse system composed by an external agent. Deriving new systems from building blocks is a matter of understanding reusable components and composition of them. In generation approach, the reused components are often patterns of code and transformation rules. This active methods involve components which were executed to generate target systems. These generators are reused whenever they are executed. Since the general problem of program synthesis is very difficult, most of these systems must be specially tailored for a specific application area.

Considering two approaches based on user intervention, building block method needs a user to retrieve reuse library, identify proper components, and synthesize them into a new target system. Generation method makes a minimum user intervention in parameterizing reuse patterns. Most current reuse systems are based on building blocks which have function encapsulated and well-defined interfaces. RSL[3] is an example of reuse system based on building blocks.

2.1 Classification of reuse components

Prieto-Diaz[4] introduced two types of classification schemes for reusability: enumerative approach and facet approach. Enumerative method postulates a universe of knowledge divided into successively narrow classes. These classes include all the possible subclasses and compound classes arranged in hierarchical relationships. When the reuse components have clear-cut relationship, they can be retrieved fast. But an inherent problem with enumerative schemes is traversing the hierarchical tree to find the appropriate class. Selecting the most appropriate class is a difficult task because more than one class may be applicable. Moreover, enumerative approach has difficulty in adding new class because it makes reuse hierarchical tree reconstructed.

In facet approach new classes can be easily added without reconstruction. Reuse components are represented with collection of related common terms, called facets. Each reuse component has several types of characteristics which are identified to facets. Facets can be considered as function, objects, system-type, function area, etc. Each facet has items to be ordered according to how closely related they are each other. This idea has been extended to provide an even more precise measurement of similarity among items. If no match of a description can be made during the retrieval, the conceptual closeness is measured and other closely related items are provided. The facet approach uses a basic class to represent reuse components so that it makes easy classification and understandable. However, adding a new item increases the number of facets which means more difficult in representing relationship of items and dealing with synonym.

Weiping[5] suggested an interesting reusability approach which is based on software engineering and information engineering concepts. Software engineering concepts includes commonality, portability, modularity, maintainability, accessibility, understandability, reusability. Information engineering concepts are technical feasibility, generality, modifiability. Reuse components are classified in these concepts. It is not easy to provide objective measurement of reuse components.

2.2 Retrieval of reuse components

Reuse components library is retrieved through several methods. The user provides keywords which represent information and knowledge about needed reuse components. The keywords are used in case that enough information and knowledge are known. There is another retrieval method by enumeration of behavior. This is effective when the function of reuse components is already known. The third method is interactive retrieval with a user. The user gives the information incrementally until the system finds out the exact proper components. Browsing the reuse code, explanation, or documents help understanding reuse components. That will make cognitive distance short[6].

3. Related research

Most reuse libraries are constructed forward for the future reuse in software development. Another approach is a technique which applies reverse engineering to extraction and abstraction of reuse components. It takes too much time and effort to develop reuse libraries from scratch by forward engineering. Reverse engineering gives a clue to build reuse libraries and saves time to code repeated trivial stuff.

There have been three main research stream:(1) characteristics of reuse components to be used in evaluating reusability, (2) reuse system development including extraction, classification, retrieval, and modification of reuse components, (3) methods to shorten cognitive distance. Arnold[7] defines reuse component as a module that is referenced frequently, has low coupling, and high cohesion. Dunn and Knight[8] suggested an algorithm to find abstract data type by analysing call relationships and use of global variables among modules. They also developed the code miner which finds a place of reusable components. The code miner use inference engine of Prolog to know the place of reusable components.

We can classify reuse system into two types. The first type of reuse system is reverse-oriented. CIA(C

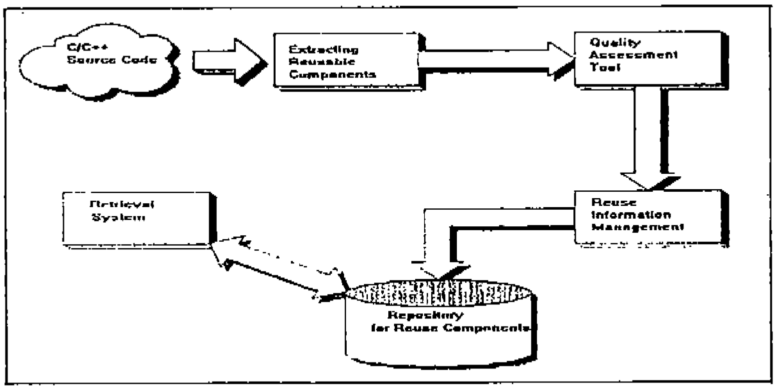
Information Abstractor) defines five types of data objects within C source code. File, macro, data type, global variable, function are extracted, abstracted, and stored in relational database. Subsystem can be extracted by analysing source code related with specific data objects[9]. Another type of reuse system is forward-oriented. Caldiera, Basili[10] suggested a reuse metric based on cyclomatic complexity, regularity, reuse frequency, and volume and appled it to building CARE(Computer Aided Reuse Engineering) system. CIA abstracts relationship of data and function from the source code. However, it is not effective to find reusable components. CARE also needs expert involvement to assess components and give test cases.

After retrieving reuse library, we need a lot of effort to understand meaning and i/o parameters of reuse library components. We call it a cognitive distance, intellectual effort to understand reuse component. It could not be measured because it depends on prior knowledge about reuse domain. Reuse system helps to make cognitive distance short. Fischer[6] developed a code finder guiding a path to reuse component and explainer displaying related part of documents and executing reusable source code. Code finder and explainer support Lisp program comprehension by reverse engineering graphics library.

4. A Design of Reuse System

A reuse system should support several useful functions such as extracting, classifying, and retrieving C and C++ source components. The design of CSORUS starts with extracting reusable components from C and C++ source code. The source code should not have any syntactic error. However, the reuse system verifies C and C++ syntax due to the file crash or modification in reuse process. Lint function in Unix validates source code with syntactic rules and identifies various relationships of functions and classes such as inheritance, coupling, dependency. Those relationships will be used in selection of reusable components. Candidates of reusable components are assessed by reusability metrics. The reuse system selects reusable components and stores them with keywords and explanation to reduce cognitive distance. In addition, the system has function to search a proper components through several different retrieval methods.

The reuse system has a work flow to make reusable components as shown Fig. 1. It includes (1) extraction step, (2) quality assessment step, (3) storing reusable components in repository, (4) retrieval step, (5) components comprehension, and (6) composition of components. First half has been used in reverse engin-



(Fig. 1) Reuse system work flow

eering techniques with information abstracted from source code. Cognitive techniques are applied to second half.

We have designed and implemented six reverse engineering tools to extract reusable components. Syntax error checker has the same function as Lint in Unix. It covers ANSI C and AT&T C++ 3.0 grammars. Pretty printer helps code reading by indentation and formatting source code with good style. Function prototype extractor reads C and C++ source code excluding class definition to extract function signature. Class information extractor provides information about class definition, member data, and member functions. Function dependency checker accepts function prototypes and makes function call graphs which will be used in assessing reusability.

5. An Implementation of Reuse System

5.1 Extraction of Reusable Components

The key idea of extraction is based on the class and strong cohesive functions. Class is a real good candidate for future reuse because it has been encapsulated and hidden with cohesive objects. Another criterion is function call frequency which represents possibility of reuse. The first step of extraction is an error checking to assure error-free reusable components. Simple noise or static error does not allow extraction step to go on. If the reuser modify reusable components and store them in the reusable repository again, it should be checked syntactically first. Pretty printer gets error-free source code and make it reformatted such as indentation and transformation for consistency of coding style. For example, K&R type function declaration in C program should be converted to ANSI C style. Next step for extraction is finding out function prototype and class information. Function call graphs are generated by function dependency checker.

Reusable components are extracted according to the following criterion. functions are referred frequently. In function call graph, a component with high in-degree and 0 out-degree might be a good candidate for reusable component. Reuse system generates cross-reference table for function prototype in first pass scanning code. The second pass scanning determines whether each function call other functions in table or not. Another criterion for extraction is AND-OR tree. Structured C program has simple control flow including sequence, selection, and repetition. We can construct AND-OR tree in which main routine is a root node, and, functions are children. Functions running in sequence are connected by AND node. Other types control structure can be represented as OR node which means running selectively. The nodes connected with AND have strong coupling with other node. Because the result of a function affects on the next function executing sequentially. Functions connecting with OR node have low coupling due to running alternatively. The reuse system traverses nodes from the bottom node to the top. If OR node is reached, subtree from the bottom to the OR node is recognized as candidates of reusable components.

Class is considered as an independent reusable unit. It is encapsulated, self-contained and hides information so that the class status is changed only by message-passing. However, every class can not be a reusable component by itself. For example, pure virtual function which is defined in derived class. That should be included in final derived concrete subclass. For another example, inheritance from other classes and class reference by using friend declaration. That should be combined with corresponding classes, recognized as one reusable component. For those considerations, class information extractor gets information about class name, file names in which classes are defined, start and finish line number of class definition, class classification representing the possibility of independent reuse, super class or subclass, member function names. Table 1 shows class information extracted from a sample C++ program in Fig. 2. CSORUS displays these information as shown in Fig. 3.

```
 9:
10: class stack {
11:         private:
12:                 int         top;
13:                 char        data[10];
14:         public:
15:                 stack();
16:                 ~stack();
17:                 void        push(char* );
18:                 char*       pop();
19: };
20:
```

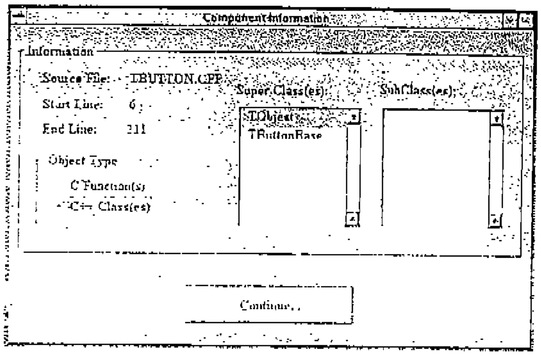
(Fig. 2) Source code for Stack

〈Table 1〉 Extracted information for Stack

Type of information	Extracted information
Class name	stack
File name	stack.cpp
Start line number	10
End line number	18
Type of Object	class
Superclass	false
Subclass	false
Member function	push pop

Identifying super or sub class needs not only inheritance relationship but also Gen-Spec and Whole-part relationship in OOA/OOD which is suggested by Coad and Yourdon[11]. Member functions of specific class will be popped up by clicking the button of 'C functions' in object type. Table 2 described the mapping C++ code to OOA/OOD in class relationship.

After conversion of class structure to OOA/OOD style, the system applies the following classification algorithm to instance of relationships and stores class relationship in data structure shown as Fig. 4.


Component Information

Information

Source File: TBUTTON.CPP

Start Line: 6

End Line: 211

Super Class(es): TObject, TButtonBase

SubClass(es):

(Fig. 3) Displaying information for extracted class

〈Table 2〉 Relationship of C++ and OOA/OOD

C++		OOA/OOD
Base-Derived Class	→	Gen-Spec Structure
Nested Class	→	Whole-Part Structure

Step 1: perform the following operations for every type of class in OOA/OOD style structure.

```
① Gen type class in Gen-Spec relationship
        Search Spec node.
        while(found) {
                Link LowerLink with Spec node.
                Search next Spec node.
        }
② Spec type class in Gen-Spec relationship
        Search Gen node.
        while(found) {
                Link UpperLink with Gen node.
                Search next Gen node.
        }
③ Whole type class in Whole-Part relationship
        Search Part node.
        while(found) {
                Link LowerLink with Part node.
                Search next Part node.
        }
④ Part type class in Whole-Part relationship
        Search Whole node.
```

```
    while(found) {
          Link UpperLink with Whole node.
          Search next Whole node.
    }
```

Step 2: store identified relationships into data structure named ClassRel.

```
typedef struct {
        char       *Name;              // Class Name
        LIST       *AKOlist;           // Gen-Sepc List
        LIST       *APOlist;           // Whole-Part List
        void  *UpperLink[MAX];         // Link to
                                       // Upper Level
        void  *LowerLink[MAX];         // Link to
                                       // Lower Level
} ClassRel;
```

(Fig. 4) Data structure for class relationships

5.2 Quality assessment of Reusable Components

The ultimate purpose of reuse is construction of high reliable reuse components and improvement of productivity. Quality assessment measures various characteristics of reuse components and decides reusable library. Reuse components might be considered in terms of cost, usefulness, quality. All functions and classes are assessed by different types of quantitative metrics for candidates of use components.

The cost of reuse components is sum of extraction cost, repacking effort, retrieval cost, and integration cost. That means reuse cost can not be calculated in construction phase. We decided to measure the reuse cost indirectly by using complexity, readability, volume. The usefulness of reuse component is affected by commonality and variety. Commonality is how similar extracted component and application domain are. That can be measured in ratio of calling frequency of reusable component to calling frequency of standard library. Variety can be represented by measuring parts of reuse components to be modifiable. The measurement of variety is more difficult than that of commonality. We apply a principle to measurement of variety. The more the complex system is, the more functionality the system have. Too much functionality means that the system can be implemented in various ways.

Reliability of reuse library can be represented by preciseness, readability, auditability, modifiability. The volume of reuse component plays an important role of preciseness of reuse component. A small component can be expected as a precise functionality. The auditability is measured by the number of independent paths. A large volume and complex system has high possibility of error and difficult audit. Volume and complexity can be applied to measurement of preciseness and auditability.

The system uses the Halstead Software Science metric[12] in measurement of volume, McCabe's cyclomatic measurement in complexity. Component regularity measures the readability and the nonredundancy of a component's implementation. Again using the Halstead Software Science Indicators, we have the actual length of the component, $N=N_1+N_2$ and the estimated length, $\hat{N}=n_1\log_2 n_1+n_2\log_2 n_2$. The closeness of the estimate is a measure for the regularity of the component's coding: $r=1-N-\frac{\hat{N}}{N}=\frac{\hat{N}}{N}$. We can estimate reuse frequency by comparing the number of static calls addressed to a component with the number of calls addressed to a class of components that we assume.

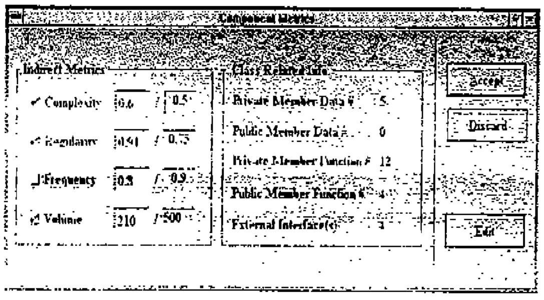
Reuse candidates including class have some problem to apply above metrics. Because class is encapsulated and hides information. Class reuse components can be measured by checking references from the external module through member function and member data. Accessibility of member function and member data is classified as private and public. Private class does not affect reusability due to the preservation of all internal information. Public class has an effect on reusability. Through an empirical study, we can define reusability of class as the following. Reusability of class is in reverse proportional to the number of interface, that is, the sum of the number of member function and the number of member data. Fig. 5 shows the result of components quality metrics. Indirect metrics represent comparison of measured value and designated thresholds. The user can select reusable components with proper degree of quality characteristics.

```
typedef struct tagCOMPONENT {
        bool                    singleComponent;
        bool                    classComponent;
        LINK                    *relatedTopic;
        LINK                    *prevComponent,
        LINK                    *nextComponent;
        COMPONENTINFO           info;
        char                    *relatedKeyword[];
        char                    *description;
} Component;
```

(Fig. 5) Results of software metrics for reusability

5.3 Storing Reusable Components

Selected reusable components are stored in information repository with retrieval keyword and documentation. The user can add extra information such as function name, formal parameters, keywords extracted from comments. The explanation of reusable components makes cognitive distance short. Reusable components are stored in repository with following data structure in Fig. 6.



Component Metrics

Indirect Metrics

✓ Complexity	0.6	/ 0.5
✓ Regularity	0.91	/ 0.75
Frequency	0.8	/ 0.9
✓ Volume	210	/ 500

Class Related Info

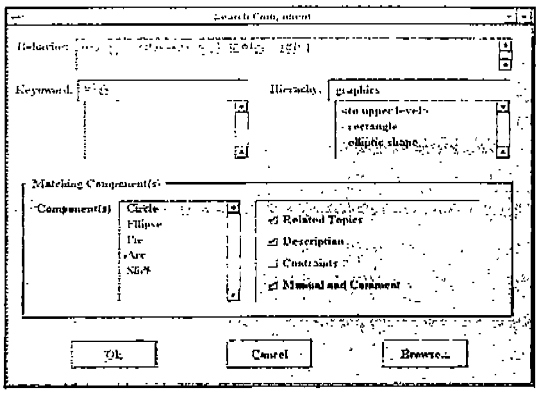
Private Member Data #	5
Public Member Data #	0
Private Member Function #	12
Public Member Function #	4
External Interface(s)	3

Accept Discard Edit

(Fig. 6) Data structure for storing reusable components

5.4 Retrieving Reusable Components

Our reuse system supports a function for retrieval of reusable components. The user provides requirements of reusable components to be used in new application and gets source code from the reuse library. The reuse system has facility to describe properties of required components. Keyword search and hierarchical search can be selected by the user for searching reusable components. The system searches matching components only by keywords, not by run-time behavior.



Search Component

Behavior: [illegible]

Keyword: [illegible]

Hierarchy: graphics
- to upper levels
- rectangle
- elliptic shape

Matching Component(s)

Component(s): Circle, Ellipse, Pie, Arc, Slice

✓ Related Topics
✓ Description
Constraints
✓ Manual and Comment

Ok Cancel Browse...

(Fig. 7) Searching for reusable components

Keyword search uses combination of logical operators such as AND, OR. Incomplete keywords can include meta characters such as '?', '*'. If the system selects more than one component by searching phase, it displays all possible components in priority order. After searching phase, the system browses source code, related keywords and manuals to make the user understood. If the user selects related topics in Fig. 7, the system searches other components by using related topic field in data structure.

6. Conclusion

To make sure of the efficiency of developed system we applied reuse system to three different application programs. One application is written in only C programming language and has 30 modules for membership management. The other sample is written in C++ programming language. It has 135 classes and 113 modules of YACL 1.2 for user interface management. Last sample application is multimedia editor in C++ programming language, which has 95 modules and 34 classes. We extracted 21 reusable components from the first sample program, 178 from the second, 45 from the third. When we construct a reverse engineering tools for the new application, we can reuse a large part of the implementation.

From the experiment, we can conclude that reuse library is constructed more efficiently by reverse engineering. The paper suggests a principle of extracting reusable components and indirect measurement method to select reusable components. We also provides retrieval method dependent on the user understandibility. The reuse system reduces the size of retargeted source code by optimizing useless code.

For the future research, we define various software metrics which make more effective on selecting reusable components. The retrieval function of the reuse system should be improved for the novice user. One way to improve usability is extracting constraints or properties of components for retrieval in reusable code extraction. CSORUS supports only source code level reuse. We believe that it can be extended to be reused in analysis or design phase by design recovery technology.

References

[1] T. Biggerstaff, C. Richter, "Reusability Framework, Assessment, and Directions," IEEE Software, Vol. 4, No. 2, pp.41-49, Mar. 1987.

[2] W. Tracz, "RMISE Workshop on Software Reuse Meeting Summary," Tutorial on Software Reuse: Emerging Technology, pp.41-53, 1990.

[3] M. Lenz, H. Schmid, P. Wolf, "Software Reuse through Building Blocks," IEEE Software, pp. 34-42, July. 1986.

[4] R. Prieto-Diaz, P. Freeman, "Classifying Software for Reusability," IEEE Software, pp.6-16, Jan. 1987.

[5] Y. Weiping, M. Tanik, D. Yun, T. Lee, and A. Dale, "Software Reusability: A Survey and Experiment," Proceedings of the Fifth Annual Joint Conference on Ada Technology and Washington Ada Symposium, pp.65-72, Oct. 1987.

[6] G. Fischer, S. Henninger, D. Remiles, "Cognitive Tools for Locating and Comprehending Software Object for Reuse," Proceedings of the 13th International Conference on Software Engineering, pp.318-329, 1991.

[7] R. S. Arnold, "Heuristic for Salvaging Reusable Parts From Ada Source Code," SPC Technical Report, ADA_REUSE_HEURISTIC-90011-N, Mar. 1991.

[8] M. F. Dunn, J. C. Knight, "Automating the Detection of Reusable Parts in Existing Software," Proceedings of the 15th International Conference on Software Engineering, pp.381-390, 1993.

[9] Y. F. Chen, M. Y. Nishimoto, C. V. Ramamoorthy, "The C Information Abstraction System," IEEE Trans. on Software Engineering, Vol. 16, No. 3, pp.325-334, Mar. 1990.

[10] G. Caldiera, V. R. Basili, "Identifying and Qualifying Reusable Software Components," IEEE Computer, pp.61-70, Feb. 1991.

[11] P. Coad, E. Yourdon, Object-Oriented Design, Prentice-Hall, 1991.

[12] M. H. Halstead, Elements of Software Science, Elsevier North Holland Inc., 1977.

최 은 만

1982년 동국대학교 전산학과 졸업(학사)
1985년 한국과학기술원 전산학과 졸업(석사)
1993년 미국 IIT 전산학과 졸업(박사)
1985년 한국표준연구소 연구원
1988년 데이콤 수임연구원
1993년~현재 동국대학교 컴퓨터공학과 조교수
관심분야: 객체지향 소프트웨어공학, 소프트웨어 유지보수, 소프트웨어 재사용, 역공학