論文
4-4-6

Component Identification using Domain Analysis based on Clustering
(클러스터링에 기반 도메인 분석을 통한 컴포넌트 식별)

Haeng-Kon Kim(김 행 곤)[1] Jeon-Geun Kang(강 전 근)[2]

ABSTRACT

CBD is a software development approach based on reusable component and supports easy modification and evolution of software. For the success of this approach, a component must be developed with high cohesion and low coupling. In this paper, we propose the two types of clustering analysis technique based on affinity between use-cases and classes and propose component identification method applying to this technique. We also propose component reference model and CBD methodology framework and perform a case study to demonstrate how the affinity-based clustering technique is used in component identification method. Component identification method contains three tasks such as component extraction, component specification and component architecting. This method uses object-oriented concept for identifying component, which improves traceability from analysis to implementation and can automatically extract component. This method reflects the low coupling-high cohesion principle for good modularization about reusable component.

Keywords : Component Reference Model, Component Taxonomy, Affinity-based Clustering Analysis Technique, Component-based Development.

요 약

컴포넌트 기반 소프트웨어개발 (CBD: Component Based Development)은 재사용 부품을 기반하여 소프트웨어 개발, 수정, 유지보수를 용이하게 지원한다. 따라서 컴포넌트는 강한 응집력과 양한 결합력으로 개발되어야 한다. 본 논문에서는use case와 클래스를 간에 유사성을 통한 클러스터링 분석에 기반하여 컴포넌트 식별에 대해 연구한다. 컴포넌트 참조 모델과 프레임워크를 제시하여 사례를 통해 검증한다. 컴포넌트 식별 방법은 추출,명세 및 아키텍처를 지원한다. 이들 방법론은 기존의 객체지향 방법론을 참조하며 분석에서 구현까지의 추적성을 지원하며 재사용 컴포넌트의 모듈성 지원을 위해 강한 응집력과 약한 결합력을 반영한다.

1) 정회원 : 대구 카톨릭대학교 컴퓨터정보통신공학부 교수
2) 정회원 : 아산정보기능대학 컴퓨터애니메이션과 교수

논문심사 : 2003. 5. 3.
심사완료 : 2003. 5. 26.

1. Introduction

The trend in software is toward bigger, more complex systems. Component-based development is the most promising way of controlling the complexity and cost of software system, and has many potential advantages such as shorter time to market and lower costs. CBD is a software development approach based on reusable component and supports easy modification and evolution of software. [5] For the success of this approach, a component must be developed with high cohesion and low coupling. Even if CBD is a significant approach, it is still a methodology with lot of problems. One of the problems is the lack of established concept and procedure.

Component can be defined with various perspectives. One of perspectives is interface-oriented approach that concentrates on business services. Another perspective is integrity approach. [1] This emphasis on independence is important because interface-oriented components do not necessarily have implementation independence. Interface-oriented components are logical notion but reusability of component can be increased. Components of integrity approach are physical notion and these are can be easily realized and traced. We propose component identification method according to integrity perspective.

In this paper, we discuss the component identification issue: how a piece of software can be recognized as a reusable component, how to describe a component. And we propose component identification method by the affinity-based clustering analysis technique. We also propose component reference model and CBD methodology framework in section 2. Component reference model consists of component taxonomy and component element. In section 3, we propose the two types of affinity-based clustering analysis technique for identifying component with high cohesion. Finally, we propose component identification method and present case-study examples to demonstrate how affinity-based clustering technique is used in component extraction.

2. Related Work

2.1 Component Reference Model

Component is a self-contained piece of software with a well-defined interface or interface set. The component has interfaces that are accessible at run-time and the component can be independently delivered and installed. This component can be also easily combined and composed with other components to provide useful functionality. [12]

We propose the component reference model for component identification. Component reference model consists of component taxonomy and component element. Component taxonomy shows the component types that are classified by granularity, and component element shows the component internal structure.

[Fig. 1] shows component taxonomy model including component types. IT component and business component can be comprised to an application component. Multiplicity of IT

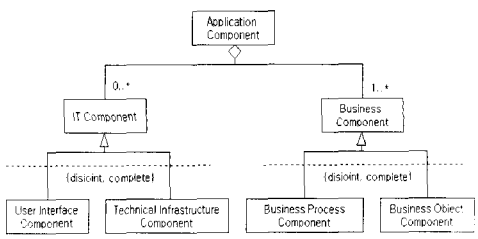
component is zero or more. IT component can be separated to user interface component and technical infrastructure component. Multiplicity of business component is one or more. Business component can be separated to business process component and business object component. They are as follows:

Application Component

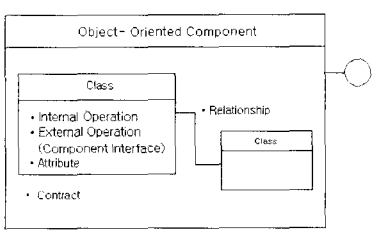
An application component corresponds to an information system or application in CBD technology. This is a set of cooperating business components assembled together to deliver a solution to a business problem.

Business Process Component

A business process component implements collaborations among the number of components. An example of this is a sales-component, that is collaboration between an order component and a customer component. Business process component is the same as business collaboration component.



[Fig. 1] Component Taxonomy Model



[Fig. 2] Component Element Model

Business Object Component

A business object component implements the data and operations of a concept from the business domain. The state of these should be persistent over the life of the application. Examples of this are a customer component and order component.

User Interface Component

A user interface component includes things like dialog-box, button, list-box, edit-box, and so on. This is client-side component.

Technical Infrastructure Component

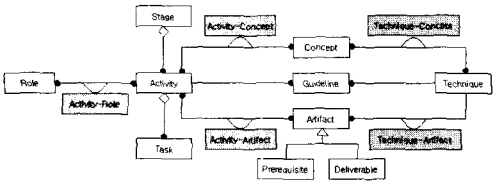
A technical infrastructure component offers technical services that are used by other components of the system. This includes things like an event-handler, a database management, authorization management, and so on.

Component contains interfaces that contain collection of private operations and public operations. The component provides at least one interface. This interface is the user's entry point to operations of the component. Component also contains attributes, internal operations, constraints, and relationships. Internal operations include the actions on data stores and set post-conditions when pre-conditions are met. If a component is implemented by object-oriented technology, the component contains objects, which consists of attributes, private/public method and their relationships.

2.2 Component-Based Development (CBD)

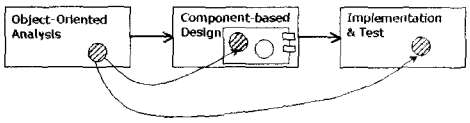
Component-based development is a broad term that describes a generic approach to

delivering software applications. It encompasses a broad range of activities, standards and technologies which, when deployed appropriately, deliver applications that are genuinely component-based. [4] We propose general methodology framework.[Fig. 3] shows the development & management methodology framework. CBD methodology must contain role, component related concept, technique, artifact and process such as stage, activity and task. We propose component development and management methodology framework to define component identification method. Component identification method presented in this paper consists of activity, technique, prerequisite, deliverable, and guideline according to methodology framework. Component identification is an activity of analysis stage for CBD. Component identification delivers clustered class diagram, component description and initial component architecture, and uses affinity-based clustering analysis technique for extracting component.



[Fig. 3] Methodology Framework

We have categorized component-based software development into three approaches. [Fig 4] shows the hybrid development approach. Initial analysis stage uses object-oriented concept. In the final analysis stage, component is extracted for defining design scope. This approach can easily use object-oriented technology such as OOP and produce object-oriented component. It also has strong traceability from analysis stage to implementation stage. CBD96 is a representative CBD methodology using this approach.

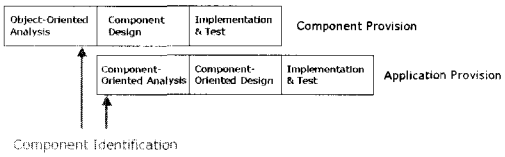


[Fig. 4] Object-Oriented & Component-Based Development Approach

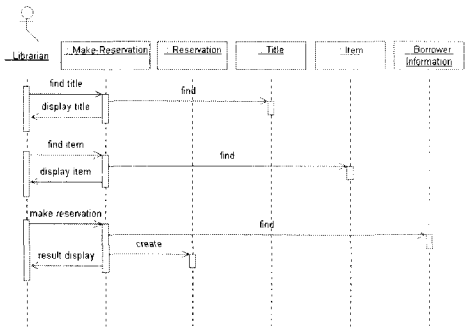
[Fig. 5] shows the component-oriented development approach. This approach can easily produce non object-oriented component. But, it has weak traceability if components are implemented by OOP such as C++ and Java.

[Fig. 5] Component-Oriented Development Approach

[Fig 6] shows the twin-track component-based development approach. This approach uses object-oriented & component-oriented concept. Component provision needs object-oriented technology and application provision needs CBD technology such as component customization and adaptation for composition. It is difficult to manage process because development approach is parallel. But, it is useful that we can manage component development separately. By this approach, we can easily reuse component later and develop COTS (Commercial-Off-The-Shelf) component.



[Fig. 6] Twin-track Component-Based Development Approach



[Fig. 7] Example of Sequence Diagram

3. Affinity-Based Clustering Analysis Technique

Affinity-based clustering analysis technique is described with two sections. The first section is prerequisites for clustering analysis. The second section is affinity-based clustering analysis technique for component identification. The third section is guidelines for good component identification.

3.1 Prerequisites

Business model and software architecture are necessary for component identification. The business model consists of business need description, business type model and business use-case model. The application architecture consists of use-case model, class model, interaction model, and involvement matrix such as use-case/class matrix and class/class matrix.

[Fig 7] shows an example of sequence diagram for Make reservation use-case. This prerequisite can be used for component specification and architecting tasks to extract component interface.

3.2 Domain based Clustering Analysis Technique

Affinity-based clustering analysis investigates affinities among the classes and use-cases and identifies those that need to be dealt with as a component. This is used to define cohesive groups of objects and use-cases. Affinity-based clustering analysis starts with use-case model, class diagram and use-case/class involvement matrix. Use-case/class matrix shows the actions (Create, Read, Update, Delete) of each use-case on each class. Class has 3 stereotypes such as *boundary, control* and *entity* according to UML notation. Class's stereotypes determine component's type. If a component consists of entity classes only, this component type is business object component. Class/Class affinity analysis is a measure of similarity of one class to another based on the ratio of the number of use-case's actions they have in common to the total number of actions of the first class. This determines class's group according to affinity level.

The two types of affinity-based clustering analysis

A. Class/Class affinity analysis

This technique uses the use-case/class matrix, and class/class matrix including only entity class. This is also a three-step process. First, calculate affinities; second, sort the affinities in descending size order; third, form new groups and merge existing groups based on cut-off criteria that you determine based on an examination of the affinity values.

Step 1 : Calculate affinities for each class pair. In a class/class matrix, the affinity for one row class Ci for another Cj is:

$$Affinity\ of\ Ci\ for\ Cj = \frac{Number\ of\ column\ use-cases\ in\ common\ to\ Ci\ and\ Cj}{Number\ of\ column\ use-cases\ in\ Ci}$$

The average affinity of a pair of class is 1/2 the sum of each affinity.

Step 2 : Sort affinities in descending order: Determine approximate natural breaks between affinity levels. Set tentative values for creating new groups, merging groups, adding classes to an existing group, and ignoring affinities.

Step 3 : Form Groups: Examine all pairs with the same affinity. If neither class of a pair is in a group and the affinity is at or above the create level, the pair creates a new group. If one of the pair is already grouped, if the add level affinity is high enough add the other class to the group. If each class has already been placed in its own group and their affinity is at or above in the merge level, then merge the two groups.[7]

B. Use-case/Class affinity analysis

This technique uses the use-case/class matrix.

Step 1 : Remove Boundary class from class set

Step 2 : Sort the matrix so that use-cases appear in a logical sequence. That is, use-cases that create a class appear before use-cases that update or read the class, which are followed by use-cases that delete the object.

Step 3 : Draw lines on the matrix to separate the logical use-case groupings and the logical class groupings. The intersections of the lines give the clusters. This cluster determines component and component type.

3.3 Guideline

Component is identified according to component reference model and stereotype of Class. First, boundary class is extracted to UI component. Control class and entity class is clustered by affinity-based clustering analysis for business component identification. If cluster include control classes, we define the component as business collaboration component. If cluster include only entity classes, we define the component as business object component.

- **Component Cluster Characteristics**
  - Classes and use-cases grouped together should be related to the same business need, objective, or benefit.
  - All classes that are needed to support high-priority use-cases should be included.

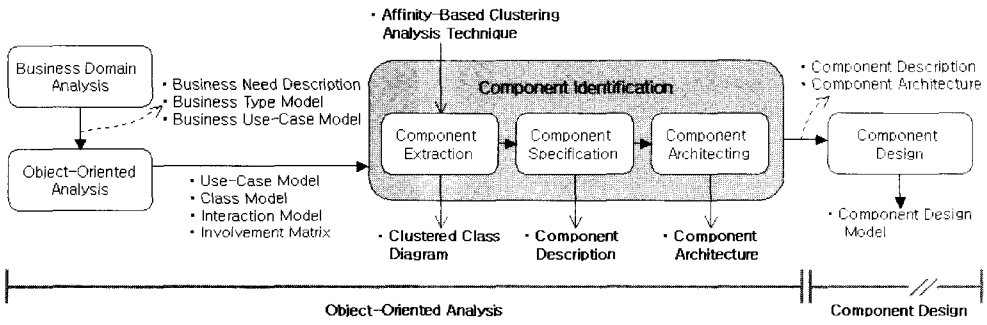
- All use-cases that are needed to support high-priority classes should be included.
- Classes and use-cases grouped together should form a consistent, coherent whole.
- Clusters should be as self-contained as possible, but it is never feasible to completely remove dependencies between clusters. We can identify the dependencies among the clusters as component interfaces.

4. Component Identification Method

Component identification method, illustrated in [Fig. 8], contains 3 tasks such as component extraction, component specification and component architecting. Component identification method uses the affinity-based clustering analysis technique, and delivers clustered class diagram, component description, and component architecture. By the incremental and iterative CBD development approach, artifacts such as component description and architecture are refined and rearranged in the later stage.

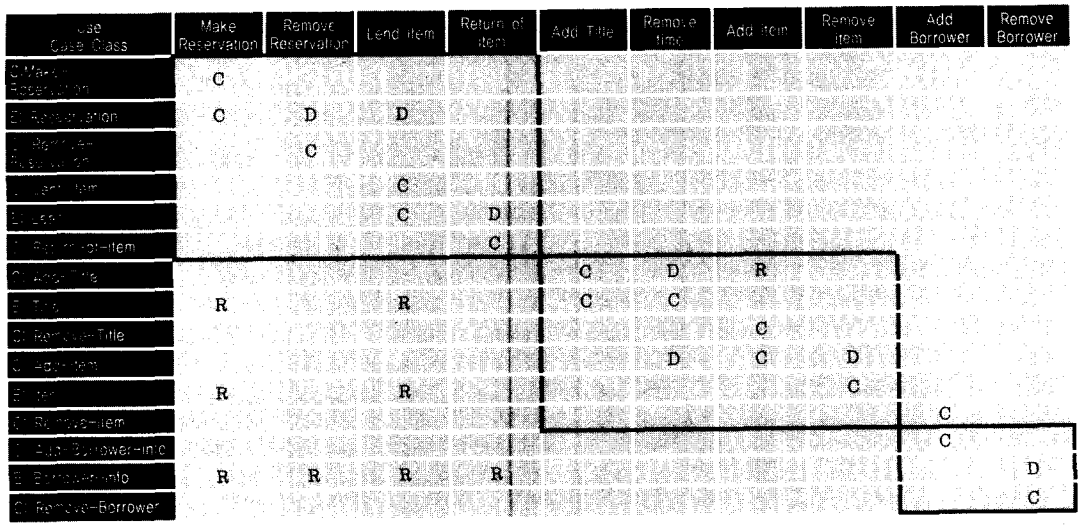
4.1 Component Extraction by the Affinity-based Clustering Analysis Technique

Component extraction is the first task of component identification. This task uses affinity-based clustering analysis technique for identifying component and delivers clustered class diagram. <Table 1> shows an example of use-case/class involvement matrix in the library system. The use cases in the library system will be Lend Item, Return Item, Make Reservation, Remove Reservation, Add Title, Update Title, and so on. Also, the library system includes control classes such as Make-Reservation, Remove-Reservation, Lend-Item, Return-of-Item, Add Title, etc., and entity classes such as Item, Title, Loan, Borrower Information, Reservation. <Table 1> shows a clustered matrix after use-case/class affinity analysis.



[Fig. 8] Component Identification Method

〈Table 1〉 Use-cases/class involvement matrix in the library system



By use-case/class affinity analysis, we defined 3 groups. Each clustered group will be identified business process components: *Reservation_ oan_Mgr, Title_Item_Mgr, Borwer_gr.*

Good candidates for business components are those business concepts that are real and independent in the business domain. [9][12] The following is the guidelines for business component identification.

- The components must be perceived as highly reusable.
- A business component realizes a business concept.
- A business component should be at a level of granularity that will support autonomous development by two to three people through the development lifecycle. [12] Two use-cases can be considered directly related if they operate on the same class or classes.

〈Table 2〉 shows the class/class matrix after class/class affinity analysis. This technique uses the use-case/class matrix, and delivers class/class affinity matrix including only entity classes. In 〈Table 2〉, affinity of Reservation for Loan is 0.33, which is calculated by one-thirds. According to Table1, number of use-cases in common to Reservation and Loan is *one*, and number of use-cases in Reservation is *three*.

〈Table 2〉 Class/Class Affinity Matrix

	Reservation	Loan	Title	Item	Borrower Info
Reservation	1	0.33	0.67	0.67	1
Loan	0.5	1	0.5	0.5	1
Title	0.4	0.2	1	0.8	0.4
Item	0.4	0.2	0.8	1	0.4
Borrower Info	0.5	0.33	0.33	0.33	1

〈Table 3〉 Class/Class Average Affinity Matrix

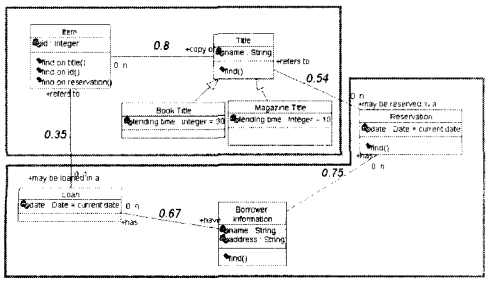
	Reservation	Loan	Title	Item	Borrower Info
Reservation	1	0.42	0.54	0.54	0.75
Loan		1	0.35	0.35	0.67
Title			1	0.8	0.37
Item				1	0.37
Borrower Info					1

The average affinity of a pair of Reservation and Loan is 1/2 the sum of each affinity, which is 0.42. 〈Table 3〉 shows the average affinity values and 〈Table 4〉 shows the class average affinity pair in descending order.

〈Table 4〉 Class Affinity Pair

Affinity Pair		Average Affinity
Title	Item	0.8
Reservation	Borrower Info	0.75
Loan	Borrower Info	0.67
Reservation	Title	0.54
Reservation	Item	0.54
Reservation	Loan	0.42
Title	Borrower Info	0.37
Item	Borrower info	0.37
Loan	Item	0.35
Loan	Title	0.35

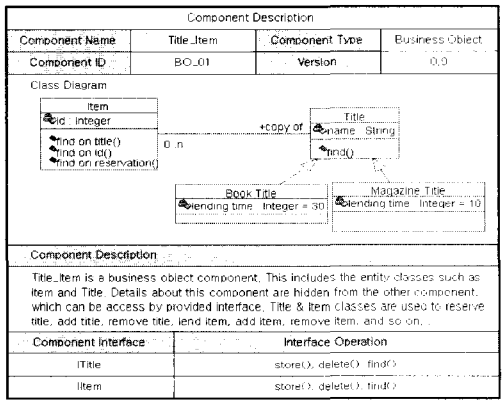
According to class average affinity pair, we formed 2 groups on the class diagram. [Fig. 9] shows the clustered class diagram. Each clustered group will be identified business object component: *Title_Item* and *Reservation _Loan_BorrowerInfo*. It also may form 3 groups by affinity level value: *Title_Item*, *Reservation _BorrowerInfo*, and Loan. Affinity level can be defined component manager such as component architect and designer, who consider on organization and project characteristics.



[Fig. 9] Clustered Class Diagram

4.2 Component Specification

Component specification task delivers component description, which contains component name, component type, class diagram, description and interface. A component interface is a set of operations. Each interface operation defines some service or function that component will perform. Component description is refined and rearranged in the later stage.



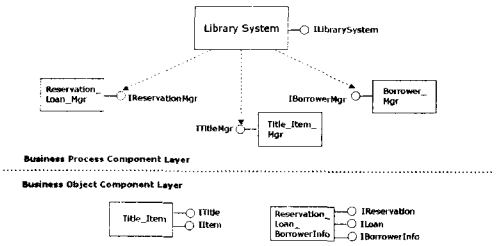
[Fig. 10] Component Description

4.3 Component Architecting

Component architecting is the final task of component identification. In this activity, we create an initial set of component architecture and form an idea of how they might fit together. [Fig 11] shows the initial component architecture for library system. In [fig. 11], library system is application component including business process component such as Reservation_ Loan_Mgr, Title_Item _Mgr, and Borrower_ Mgr and business object component such as Title_Item and Reservation_Loan _Borrower- Info. In [fig 7], it is can be extracted component

interface from sequence diagram. In this use-case, there are 3 events such as find title, find item, and make reservation. In component specification task, event of find title can be defined 'Ititle' interface for title component and event of make reservation can be defined 'IReservationMgr' interface for Reservation_Loan_Mgr com- ponent.

In our example, component architecture has two layers: process layer and object layer. Initial component architecture can be used for component interaction specification in the later. Component identification uses incremental and iterative approach and can be refined and rearranged for adding detail information.



[Fig. 11] Component Architecture

5. Conclusion

The purpose of this paper is to describe the ideas for component identification. We started the description of component reference model and various component-based development process framework. Also, we proposed the affinity-based clustering analysis technique, prerequisites, activity and guidelines with case study of library system. This method helps us to identify component with object-oriented concept and to extract automatically, and support strong traceability from analysis to implementation. This method reflects the low coupling-high cohesion principles for good modularization of reusable business component and supports reengineering from legacy system.

In our future study, we will be to put our ideas into practice, verify component reference model and affinity-based clustering analysis, and compare it with other component identification method. We also want to research component based system integration methodology including management and development processes.

※ Reference

[1] Alan W. Brown, Large-Scale, Component-Based Development, Prentice Hall, 2000

[2] Clemens Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998

[3] David M. Weiss, Chi Tau Robert Lai, Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley, 1999

[4] David Sprott, Lawrence Wilkes, "Component-Based Development", Butler Group, September 1999

[5] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999

[6] Ivica Crnkovic, Magnus Larsson, Frank Luders, "State of the practice: Component-based Software Engineering Course", ICSE, 2000

[7] James Martin, Information Engineering: Planning and analysis, Prentice-Hall, 1990

[8] John Cheesman, John Daniels, UML Components, Addison-Wesley, 2001

[9] Leonor Barroca, Jon Hall and Patrick Hall, Software Architecture, Springer, 1999

[10] Michael Siff, Thomas Reps, "Identifying Modules via Concept Analysis", IEEE Transaction on Software Engineering, VOL. 25, No.6, 1999

[11] N.H. Lassing, D.B.B. Rijsenbrij, J.C. van Vliet, "A view on Components", DEXA workshop, 1998

[12] Peter Herzum, Oliver Sims, Business Component Factory, Wiley, 2000

[13] William D. Burg, Scott Hawker, etc., "Exploring a Comprehensive CBD Method: Use of CBD/e in Practice", The third International Workshop on Component-Based Software Engineering, March 3 2000

김 행 곤

1985년 중앙대학교 전자계산학과 졸업(공학사)
1987년 중앙대학교 대학원 전자계산학과 졸업(공학석사)
1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1978~1979년 미 항공우주국 객원연구원
1987~1989년 AT&T 객원 연구원
2000.12~2001. 현재 미 Central Michigan University 교환교수
1990~현재 대구가톨릭대학교 컴퓨터공학부 부교수
관심분야 : 객체지향 시스템 설계, 사용자 인터페이스, 소프트웨어 재공학, 유지보수 자동화 툴, CASE, 소프트웨어 컴퍼넌트 공학

강 전 근

1977년 동국대학교 전자공학과 졸업(공학사)
1985년 한양대학교 대학원 전자계산학과 졸업(공학석사)
1997년 대구가톨릭대학교 대학원전자계산과졸업(이학박사)
1979년-1985년 한국전력공사 정보처리처리
1985년-2003년 영진전문대학 컴퓨터계열부 교수
2003년 -현재 : 아산정보기능대학 컴퓨터애니메이션과 교수
관심분야 : 컴퓨터애니메이션, 데이터베이스, 소프트웨어공학, 인공지능