韓國 OA學會 論文誌
第3卷 第4號, 1998. 12.

98-3-4-2-6

소프트웨어 성능공학과 소프트웨어 개발도구 및 방법과의 관계

변 진 식*

The Relationship of the Software Performance Engineering and Software Developement Tool and Method

Jin-Sik Byun*

요 약

시스템 성능평가의 대상은 하드웨어와 소프트웨어로 나누어 평가할 수 있다. 현재 하드웨어의 평가는 부문별로 많이 개발되어 왔지만, 소프트웨어의 성능은 그 대응책없이 개발되어 온 사례가 많다. 물론 외국에서는 소프트웨어 성능공학쪽으로 많은 관심을 기울러 소프트웨어 개발도구 및 방법론이 성능공학 문제와 연관하여 제안되고 있다.

그러나 우리나라에서는 거의 전무한 상태인 것은 사실이다. 그래서 본 연구에서는 소프트웨어개발 도구 및 개발방법론을 정의하여 그 내용을 비교하고자 한다.

Abstract

The performance evaluation of system can be divided into software and hardware. Recently, many hardware evaluations have been developed as many fields, however, performance of software without any solution have been also developed. As a matter of cource, in a foreign country, software development tools and methodologies related to problems of performance engineering are suggested by paying much attentions on the software performance engineering.

However, in Korea, researches on those issues are wholly lacking. Therefore, in this paper, a software development methodology will be discussed and its contents will be compared as well.

* 동명대학 사무자동화과 부교수
논문접수 : 98.11.13. 심사완료 : 98.12.18.

I. 서 론

컴퓨터를 기반으로 하는 시스템은 성능(performance) 요구를 잘 충족시켜야 한다. 성능 평가의 대상이 되는 컴퓨터 시스템은 하드웨어와 소프트웨어의 집합으로 볼 수 있다.

그러나 일반적으로 하드웨어 기술이 성능과 관련된 모든 문제를 해결해 주는 것 같이 생각해 왔으나, 현실은 소프트웨어 성능에 관한 대응책이 전혀 없이 개발이 되어 온 사례가 너무 흔하다. 그래서 소프트웨어 성능공학이 제기되고 이와 관련하여 개발도구 및 방법과의 관계가 정립될 필요성이 있다.

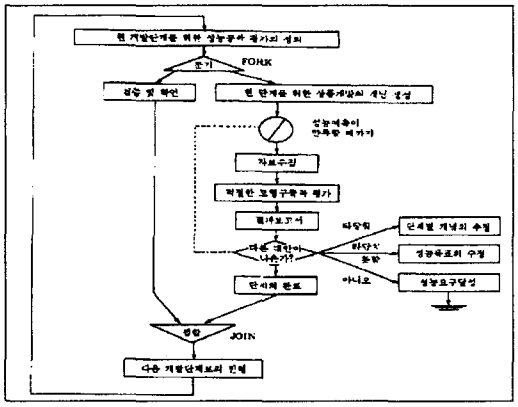

그림 1. 소프트웨어 성능공학의 모형

소프트웨어 생명주기의 전 단계에 걸쳐 성능 평가 방법을 제공하는 것이 성능공학의 목적이나, 본 연구에서는 소프트웨어 개발도구(CASE등)와 개발방법론(구조적 방법이나 자료구조 지향방법)과의 관계를 정리하여 그 내용을 평가, 분석해 보고자 한다. 또한 컴퓨터 시스템이나 설치환경 보다는 소프트웨어의 성능에 그 초

점을 맞추고 단순한 성능이나 개선책이 아닌 좋은 성능의 소프트웨어를 개발하는 공학적 방법을 알아보고자 한다.

II. 소프트웨어 성능공학의 이해

1. 정의

소프트웨어 성능공학(Software Performance Engineering : SPE)은 성능목표를 달성시키기 위한 소프트웨어 시스템 구축방법이다. 성능평가의 대상이 되는 컴퓨터시스템은 하드웨어와 소프트웨어의 집합으로 구성되며 이들 나름데로의 특성을 가지고 있다. 일반적으로 컴퓨터시스템의 성능공학이 힘든 이유는 이처럼 다양한 요소들이 복잡하게 상호작용함으로써 성능으로 나타나기 때문이다. 그러므로 소프트웨어 성능공학은 소프트웨어 공학을 보완시켜 주는 방법이다. 소프트웨어 성능공학을 그림으로 표현하면 [그림 1]과 같다.

2. 성능평가 척도들

컴퓨터시스템의 성능의 평가 척도는 사용편의성(ease of use), 구조성(structuredness), 명령문의 강력성(power of an instruction set) 등 계량화시키기 힘든 질적(qualititive) 척도들이 있는가 하면, 컴퓨터시스템 성능평가의 주관심이 되는 계수(quantitative) 척도들도 있다. 계수척도들은 다음과 같다.

① 생산성(productivity) : 주어진 단위시간내에 시스템에 의해 처리되는 정보량을 뜻한다. 처리율, 생산률,용량 등으로 표현된다.

② 반응성((responsiveness) : 시스템에 입력이 주어진 시각으로부터 시스템이 이와 관련된 출력을 생성하기 시작한 시각까지의 시간을 뜻한다. 응답시간, 총처리 시간 및 반응시간 등으로 설명된다.

③ 활용성(utilization) : 주어진 시간 동안 시스템의 특정 요소가 이용되는 정도를 말한다.

3. 성능평가의 활용도와 효과

컴퓨터시스템의 성능평가는 시스템 개발·확장의 여러 단계에서 활용될 수 있는 중요한 개념이다. 그 대표적인 예는 다음과 같다.

① 선정(selection)
② 개선(improvement)
③ 설계(design)

또한 소프트웨어 성능공학은 소프트웨어 개발의 초기 단계에서부터 성능을 염두에 둔 공학적 접근방법을 진행시키자는 것이다. 소프트웨어 성능공학의 효과는 다음과 같다.

① 생산성 향상
② 소프트웨어 상품의 품질 향상
③ 하드웨어 및 소프트웨어 관리 통제
④ 개발과 초기 운용 비용의 감소

III. 개발도구

소프트웨어업계가 지향해야 할 목표는 소프트웨어 개발과정의 자동화이다. 이중 CASE(Computer Aided Software Engineering)는 노동 집약적인 특성을 갖는 소프트웨어 개발 과정을 자동화함으로써 소프트웨어 개발의 생산성을 증대시키고자하는 목적으로 탄생된 것이다.

1. CASE의 정의

소프트웨어 공학과 관련된 여러 작업중에서 하나의 작업을 자동화한 소프트웨어 패케지를 CASE 도구라 부른다.

IEEE의 공학 용어집에서의 정의를 보면 다음과 같다. "소프트웨어 생명주기 전체에 걸쳐 프로그래밍 지원 능력을 제공하는 도구의 통합된 집합이며 이는 하나의 명령어로 접근하도록 되어 있다. 환경은 일반적으로 설계, 편집, 컴파일, 로드, 테스트, 형상관리, 프로젝트 관리를 위한 도구를 포함한다."

2. CASE의 역사

〈표 1〉은 소프트웨어 개발 지원의 발전이라는 관점에서 본 CASE의 역사이다.

표 1. CASE의 역사

연대	개발지원	개발기법
1970	고급언어 컴파일러 대화형 프로그래밍 설계 분석.설계용 CASE	구조적 프로그래밍 구조적 설계 구조적 분석
1980	4세대 언어 정보저장소 코드생성기	프로토타이핑 정보공학
1990	역공학 CASE 자능형 CASE 통합 CASE	객체지향기법 통합방법론

3. CASE의 기능

CASE사용의 주된 목적은 생산성 향상, 비용절감, 소프트웨어 개발 과정 촉진이며 이런 완벽한 소프트웨어를 지원하기 위해서는 다음과 같은 기능을 가지고 있어야 한다.

① 그래픽 기능
② 오류 검사
③ CASE 정보 저장소
④ 통 합
⑤ 생명주기의 초기 부하
⑥ 원형화
⑦ 코드 생성
⑧ 구조적 방법론의 지원

4. CASE의 분류

CASE는 다음 세 가지 형태로 나눌 수 있다.

(1) 상위(Upper) CASE

상위 CASE는 소프트웨어 개발의 초반에 일어나는 일을 지원한다. 상위 CASE는 여러 가지 명세와 문서를 작성하는 분석가의 일에 도움을 주는데 중요한 기능을 보면 다음과 같다.

- 여러 가지 방법론을 지원하는 다이어그래밍 도구
- 모델의 정확성, 일관성을 확인하기 위한 오류 검증 기능
- 프로토타이핑을 지원하는 도구

• 여러 가지 다른 모델 사이에 모순이 있는지 검사

(2) 하위(Lower) CASE

하위 CASE는 소프트웨어 생명 주기의 나중 단계들을 지원한다. 프로그래밍에 사용되는 도구는 컴파일러, 링커, 로더, 디버거 등이 있다. 이런 도구들은 CASE 도구라기 보다는 프로그래밍을 위한 환경이라 할 수 있다. 최근에는 보다 적극적으로 프로그래밍을 지원하는 도구가 출현하고 있다. 프로그래밍단계의 특별한 도구로 코드생성기(code generator)가 있다. 또한 구문중심편집기(syntax-directed editor), 전문가 시스템을 이용한 컴파일러도 사용되고 있으며 다양한 종류의 테스트, 디버깅 도구가 개발되어 보급되고 있다.

(3) 통합(Integrated) CASE

통합 CASE는 소프트웨어 개발 주기 전체를 지원하는 도구로 구성된다. 또한 구성하는 도구들이 공통의 정보저장소를 이용하며 사용자를 위한 인터페이스도 일관성을 갖고 있다. 통합 CASE는 특정 개발 모형과 방법론을 지원하는 제품도 있고 여러 가지 방법론을 지원하므로 선택하여 쓰는 것도 있다. 통합 CASE의 다섯 개의 기본적인 구성 요소가 [그림 2]에 나타내었다.

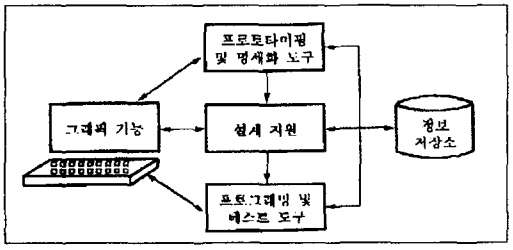

그림 2. 통합 CASE의 구성요소

통합 CASE의 장점으로는 다음과 같다.

① 소프트웨어 개발의 모든 단계에 걸친 표준의 확립
② 방법론을 지원하는 도구들의 사용으로 개발 기법의 실용화
③ 시스템 개발 기간의 단축 및 개발 속도의 증진
④ 소프트웨어 품질 향상 및 용이한 프로젝트 관리
⑤ 소프트웨어 모듈의 재사용성 증대
⑥ 시스템 수정 및 유지 보수의 용이성 향상
⑦ 문서화의 용이성 제공

5. CASE의 종류

CASE의 종류는 다양하게 많으나 몇가지만 소개한다.

(1) SREM

SREM(Software Requirements Engineering Methodology)은 요소들, 속성들, 관계들, 그리고 구조들을 기술하기 위해서 RSL(Requirements Statement Language)을 사용해서 만든 자동화 요구사항 분석 도구이다.

SREM도 분석가가 요구사항 단계에서 진행해야 하는 절차들의 집합을 통합시켜 놓은 것이다.

① 번역(Translation) : 시스템 명세서에 기술되어 있는 초기의 소프트웨어 요구사항들을 보다 상세한 자료 설명과 처리단계들의 집합으로 바꾸어 주는 활동.
② 분해(Decomposition) : 소프트웨어 요송애 대해 인터페이스에서 정보를 평가하고 완전한 계산적인(기능적) 요구사항들의 집합으로 만들어내는 활동.
③ 할당(Allocation) : 설정된 요구사항들에 대해 대체 접근방법들을 고려해 주는 활동
④ 분석적인 타당성 조사(Analytical feasibility demonstration) : 타당성을 결정하기 위해 중요한 처리, 요구사항들을 모의 실험하는 활동

(2) PSL/PSA(Problemstatement Language/Problem Statement Analyzer)

원래 미시간대학교에서 개발된 것이지만 지금은 CADSAT(Computer Aided Design and Specification analysis)라 불리는 대형 시스템의 한 부분이 되었다.PSL/PSA는 분석가에게 : ①응용분야에 관계없이 정보시스템들의 기술 ②정보시스템에 대한 기술어를 포함하는 데이터베이스의 생성 ③기술어의 추가, 삭제, 정정 ④명세서에 형식화된 문서와 다양한 보고서 생성들을 포함하는 능력들을 제공해 준다.

(3) TAGS(Technoligy for Automated Generation of System)

시스템 공학방법들의 응용을 위한 자동화 접근방법으로서 Teledyne Brown Engineering회사가 개발하였다.

IORL(Input/Output Reqrirements Language)라 불리는 명세서 언어, 요구사항 분석과 IORL 처리를 위한 소프트웨어 도구들의 집합, 그리고 주요 TAGS 방법론이다.

IV. 개발방법

소프트웨어의 개발 방법의 유형은 구조적 분석 방법, 프로토타입방법, 객체지향 기법 등으로 나눌 수 있다.

1. 구조적 분석 방법

초기의 구조적 기법은 문제 해결 방법과 사용자의 관련된 사항들에 대해서는 크게 고려하지 않았으나 시스템의 요구사항을 충분히 이해하지 않는 상태에서는 결국 실패한다는 것을 알고 사용자의 요구 분석 단계를 중요시하였다.

구조적 분석에서는 시스템의 요구 사항을 정의하기 위하여 주로 하향식(Top-down) 기법을 사용한다.

여러 가지 구조적 설계 기법이 있는데 이들 모두가 기능의 분해과정을 설계 방법으로 응용하고 있다.

(1) 하향식 설계기법

하향식 설계 기법은 문제를 세분화해 나가는 비정형적 설계기법이다.

설계단계는 4단계로 나누는데 [그림 3]과 같다.

1단계 : 데이터 흐름도 작성

2단계 : 구조도 작성

3단계 : 평가

4단계 : 구현을 위한 준비

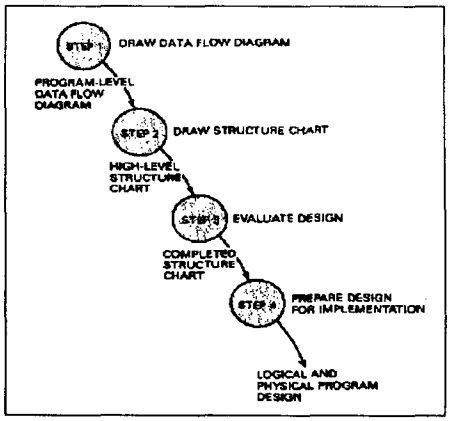

그림 3. 구조적 설계 기법 4단계

(2) Jackson 설계기법

Jackson 설계기법 역시 하향식 설계 기법을 개량해 있는데 정형화를 시키는데 중점을 두었다.

설계 방법은 다음 다섯 가지 단계로 구성된다. 처음 두 단계는 프로그램의 구조를 만드는 일이며, 나머지 세 단계는 프로그램의 구조가 정확한가를 점검한다.

1단계 : 시스템의 입력과 출력 구조를 나타내는 구조 다이어그램을 그린다.

2단계 : 자료 구조 다이어그램을 합하여 하나의 프로그램 구조 다이어 그램을 그린다.

3단계 : 사용되는 프로그래밍 언어를 고려하여 프로그램이 수행해야 할 기본 명령을 모두 적는다.

4단계 : 프로그램 구조도에서 각 기본 명령의 적당한 위치를 잡는다.

5단계 : 그림으로 그려진 프로그램 구조를 텍스트 형식(구조적 프로그램 설계 언어)으로 바꾼다.

Jackson 방법의 단점은 간단한 모듈의 상세설계여서 실제 문제에는 이 기법을 직접 적용할 수 없다는 것이다. 즉 이것은 일괄처리 중심의 설계기법이지 온라인이나 데이터베이스에서는 효과적이 될 수 없다.

(3) Warnier-Orr 설계 기법

이 기법 역시 데이터 중심의 설계 기법이다.

프로그램의 출력이 완전하게 데이터 구조를 결정짓고 그것이 다시 프로그램 구조를 결정짓는 것이다.

설계 단계는 6단계로 나눈다.

1단계 : 출력을 정의

2단계 : 논리적 데이터 베이스를 정의

3단계 : 사건분석 (Entity와 Attribut 사이에 미치는 영향 조사)
4단계 : 물리적 데이터 베이스를 구축
5단계 : 논리적 프로세스 설계
6단계 : 물리적 프로세스 설계

2. 프로토타이핑 설계 기법

프로토타이핑(Prototyping) 개발 방법론의 목적은 시스템의 일부나 전부를, 프로토타입이라 불리는 '소규모 시스템 또는 시험용 시스템'으로 개발하는 것이다.

[그림 4]에서와 같이 프로토타입의 개발은 아래 4단계로 구성되어 있다.

1단계 : 사용자의 기본적인 요구 사항을 규명
2단계 : 초기 프로토타입을 개발
3단계 : 프로토타입을 사용
4단계 : 프로토타입을 수정 및 기능 향상

프로토타입 개발방법론은 SDLC 개발방법론보다 훨씬 신속하고 반복적이지만 비공식적이다.

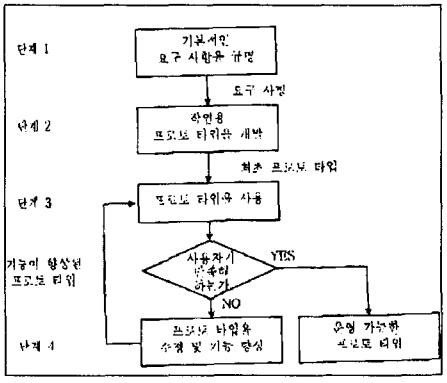

그림 4. 프로토타입 개발 4단계

3. 객체 지향 설계 기법

객체지향 설계로 프로세스나 자료 중심의 설계와는 달리 주어진 문제 영역을 그대로 표현하고 이를 프로그램으로 매핑하는 방법이다.

설계 과정을 정리하면 다음과 같다.

① 문제를 정의한다.
② 객체와 클래스를 찾아낸다.
③ 각 클래스의 속성과 필요한 연산을 정의한다.
④ 연관된 클래스들의 상속 구조를 찾아낸다.
⑤ 각 클래스의 연산을 구체화한다.

객체지향의 개발단계는 다음 [그림 5]와 같다.

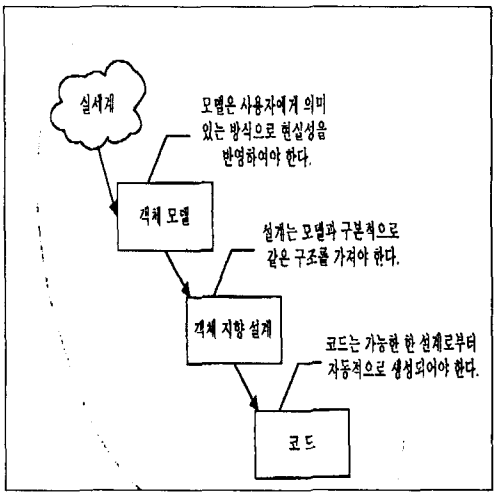

그림 5. 객체지향 개발 단계

객체지향 분석 중 대표적인 것으로는 CoadYourdon과 J.Rumbaugh가 제안한 방법이 있다.

(1) Coad와 Yourdon의 객체 지향 분석 및 설계

Coad와 Yourdon의 객체 지향 분석 및 설계 방법은 E-R 다이어그램을 사용하여 엔티티의 활동을 데이터 모델링하는 것을 기본 초점으로 하고 있다. 이 방법의 장점으로는 객체 및 클래스 식별의 우수한 기준과, 객체 지향 개념에 가장 근접한 특징들을 충족시키는 데 있다. 그러나 객체간의 통신 지원 개념이 취약하고 대규모 프로젝트보다는 작은 시스템 개발 적용에 적합하다는 단점이 있다.

분석 단계에서 아이디어를 형성한 후에 설계는아래와 같은 4가지 설계 요소를 추가하여 하향식 방법을 프로토타입(prototype)으로 구현된다.

① 문제 영역 요소
② 사람과 상호 작용 요소
③ 타스크 관리 요소
④ 데이터 관리 요소

(2) Rumbaugh의 객체 지향 분석 및 설계

Rumbaugh의 객체 모형화 기법(OMT ; Object-Modeling Technique)은 모든 소프트웨어 구성 요소들을 그래픽 표기법을 이용하여 객체를 모형화하는 방법으로 시스템의 분석, 설계, 계층화(hierarchy) 등의 일관된 객체 지향 개념을 적용한다.

Rumbaugh의 객체 모형화 기법은 객체모형화(object

modeling),동적모형화(dynamic modeling), 기능모형화(functional modeling) 등의 3가지의 모형화 방법을 적용하여 분석 모델을 설정한다.

① 객체 모형화(object modelling) : 객체들과 그 특성을 식별하여 객체 다이어그램을 작성한다.

객체 모델을 도출하는 절차는 8단계로 나눌 수 있다.

- 객체와 클래스 식별
- 클래스에 대한 데이터 사전 준비
- 클래스 간의 관계 파악
- 객체 속성 및 연결 관계 파악
- 상속 관계 구성
- 모델 검증
- 모델 반복 정제
- 클래스들의 모듈화

② 동적 모형화(dynamic modelling) : 시스템이 시간 흐름에 따라 변화하는 것을 보여주는 상태 다이어그램을 작성한다.

동적 모델을 도출하는 절차는 5단계로 요약할 수 있다.

- 상호 작용 순서의 시나리오 준비
- 객체 간의 사건 식별
- 각 시나리오를 위해 사건 추적 준비
- 상태 다이어그램 구축
- 일관성 검증을 위해 객체들의 사건 대조

③ 기능 모형화(function modelling) : 시스템 내에서 데이터가 변하는 과정을 DFD로 나타낸다.

기능 모델을 도출하는 절차는 5단계로 요약할 수 있다.

- 외부와 시스템 사이의 입·출력 값 식별
- 시스템의 데이터 흐름 다이어그램 생성
- 프로세스에 대한 기능 명세서 작성
- 제약 조건 파악
- 최적화 기준 명세화

Rumbaugh의 객체지향 설계순서는 다음과 같다.

① 시스템 설계 : 시스템 설계 단계는 이러한 분석 모델을 구성한 후 개발할 시스템을 개괄적으로 설계하는 과정을 말한다.

② 객체 설계 : 객체 설계 단계에서는 분석 단계에서 생성된 분석 모델과 시스템 설계 단계에서의 내부 구현 사항을 가지고서 구현시에 사용할 클래스 추가, 모듈 간의 인터페이스, 알고리즘 정의 등의 작업을 수행한다. 분석 모델을 보강하여 설계 산출물로 발전시키는 것이 객체 설계의 주 목적이다.

③ 구현 : 구현 단계에서는 객체 설계 단계의 세부적인 객체, 동적, 기능 모델 및 기타 문서 등을 이용하여 시스템을 구현한다.

④ 다이어그래밍 방법 : 객체 모형화 기법은 데이터 베이스의 개념적 설계의 대상으로 클래스를 먼저 정의한 후, 일반화, 집단화 개념을 적용하여 이들의 관계성을 기존의 E-R 모델을 개선하여 표시한다.

V. 결론

전통적인 소프트웨어 개발에서는 정형적인, 절차적인 개발방법론을 이용하여 개발되어 왔으나, 현재는 4GL, 객체지향 언어등을 이용하여 개발되고 다양한 CASE도구를 활용하여 구조화 기법으로 개발해 나가는 추세이다. 이중에서 특히 프로토타입 개발 방법이 비정형적 업무 개발에 많이 활용되고 있다.

각 개발도구 및 개발방법마다의 장단점은 있으나 종합해 보면 정형적인 업무의 형태로 추진되고 있는 실정이다.

본 연구에서는 각 모형의 성격을 정의해 봤지만 앞으로 이 개발도구 및 방법을 이용하여 소프트웨어 성능공학적 평가(생산성, 반응성, 활용성등)를 분석해 소프트웨어의 품질보증에 더 기여할 수 있어야 할 것이다.

참고문헌

[1] 이영욱, "MVS컴퓨터 시스템의 성능관리", 태성출 판사, 1993

[2] 유해영, "소프트웨어 공학", 희중당, 1995

[3] 이주헌, "실용 소프트웨어 공학론", 법영사, 1993
[4] 이진우, "교육 목적 컴퓨터 시스템의 성능평가 및 분석", 인하대학교 대학원, 1987
[5] 이윤미, "소프트웨어 모니터링을 이용한 시스템의 성능평가에 관한 연구", 동국대학교 대학원, 1990
[6] Allen. k.s. and a. I. Lery, "A Case Study in Software Performance Engineering", Proceedings of the International Conference on Computer Capacity Management, 1983
[7] Beizer, "Software Performance", Handbook of Software Engineering, 1984
[8] Smith. c.u., "Performance Engineering of Software System", Addison-Wesley, 1990

저자소개

변진식
한국 OA학회 논문지 제2권 제3호
참조