소프트웨어 결함 분석을 위한 트리거 설계

이 은 서[†] · 이 경 환[††]

요 약

본 연구에서는 소프트웨어 개발시 발생하는 결함을 찾아내고, 원인을 식별하고자 한다. 또한 검출된 결함 원인을 기반으로 하여 결함간의 연관성을 파악하여 트리거로 나타낸다. 따라서 결함 트리거를 이용하여 유사한 프로젝트를 수행시, 결함 예측하여 대비할 수 있게 된다.

Trigger design to software defect analysis

Eun Seo Lee[†] · Kyung Whan Lee[††]

ABSTRACT

This research introduces defect and its causes that happen on software development. Based on defect cause analysis, we understand associated relation between defects and them design defect trigger. So, when we achieve similar project, we can forecast defect and prepare to solve defect by using defect trigger.

키워드 : 소프트웨어 프로세스 개선(SPI), 결함 관리(Defect Management), 결함 분석(Defect Analysis), 결함 트리거(Defect Trigger), 프로젝트 관리(Project Management)

1. 연구 배경

소프트웨어 공학의 가장 중요한 목표는 고품질의 시스템과 어플리케이션 또는 제품을 생산해 내는 것이다. 이 목표를 달성하기 위해 소프트웨어 엔지니어는 소프트웨어 프로세스의 전후 관계에 현대적 도구들과 연관된 효율적인 방법을 적용해야 한다. 더욱이 소프트웨어 엔지니어는 고품질이 실현되었는지를 측정해야 한다. 컴퓨터 시스템의 고장은 하드웨어적인 것 보다 소프트웨어적인 원인이 더 많이 있다[1]. 따라서 소프트웨어적인 원인을 찾고 개발하기 위하여 소프트웨어 공학기술을 이용하여 신뢰성이 높은 소프트웨어를 개발하는 것이 중요한 부분이다.

그러므로 소프트웨어 공학에서 품질이라는 것이 중요한 요인이 되었다.

시스템, 어플리케이션 또는 제품의 품질은 문제를 기술하는 요구사항, 해결 방안을 모형화시키는 설계, 실행 가능한 프로그램을 이끌어 내는 코드, 소프트웨어의 오류를 발견하기 위해 소프트웨어를 조사하는 테스트 등을 기술하는 데 아주 좋다[6]. 훌륭한 소프트웨어 엔지니어는 분석과 설계 모형의 품질, 원시코드, 그리고 소프트웨어가 공학화되어 생성한 테스트 사례 등을 평가하는 측정을 한다. 이러한 실시간 품질평가를 달성하기 위해 엔지니어는 주관적인 방법보다 객관적인 방법으로 품질을 평가하는 테크니컬 측정(technical measures)을 사용해야 한다[1, 6].

품질은 프로젝트 관점에서도 살펴본다면 다음과 같다. 프로젝트 관리자는 프로젝트가 진행되면 품질을 평가해야 한다. 개별적으로 소프트웨어 엔지니어가 수집한 비공개 메트릭스는 프로젝트 수준의 결과들을 제공하기 위해 통합된다. 비록 많은 품질 측정이 수집될 수 있어도, 프로젝트 수준의 결과들을 제공하기 위해 통합된다[3]. 비록 많은 품질 측정이 수집될 수 있어도, 프로젝트 수준에서 주요 요점은 오류와 결함을 측정하는 것이다. 이들 측정으로부터 유도된 메트릭스는 개인과 그룹 소프트웨어의 품질 보증과 제어 활동의 효과성에 대한 지표를 제공해 준다[4].

검토 시간에 발견된 오류와 테스팅 시간에 발견된 오류는 메트릭스가 갖고 있는 각 활동의 효율성에 대한 통찰력을 제공해준다. 오류 데이터는 또한 각 프로세스 프레임워크 활동에 대한 결함 제거 효율성(DRE : Defect Removal Efficiency)을 계산 하는데 사용될 수 있다.

신뢰성 있는 소프트웨어를 개발하기 위해서는 소프트웨어와 개발과정에 존재하는 결함을 찾아내고 이를 제거하는 것이 중요한 요인이 된다. 이와 같은 요인은 품질로 귀결되게 되는데, 품질은 비용, 일정과 함께 프로젝트의 성공을 결정

※ 이 논문은 2002년도 중앙대학교 학술 연구비 지원에 의한 것임.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 정 회 원 : ISO/SC7/WG10 SPICE 한국 위원장
논문접수 : 2003년 2월 10일, 심사완료 : 2003년 4월 3일

하는 주요 요소이다. 소프트웨어 품질 개념은 쉽게 정의 할 수 있는 것이 아니다. 소프트웨어는 다양한 품질 관련 특성을 가지고 있는데 품질을 위한 국제 표준도 있다[2].

그러나 실제로 품질 관리는 종종 결함의 주변에서 맴돈다. 그러므로 인도된 결함 밀도, 즉 인도된 소프트웨어의 결함의 개수/단의 크기를 품질의 정의로 사용하는데, 이것은 사실 현재 업계에서 표준 정의가 되고 있다[3, 4]. 그러므로 결함의 의미를 소프트웨어 결함과 소프트웨어가 고객의 필요와 요구 사항과는 다르게 동작하게 하는 원인이라고 할 수 있다. 따라서 본 논문에서는 신뢰성 있는 소프트웨어를 생산하기 위하여 결함을 식별하고 연관성을 분석하게 된다. 또한 이를 기반으로 결함 트리거를 설계하고자 한다.

2. 기본 개념

2.1 결 함

결함 정보는 다른 유형의 원시 데이터이며, 소프트웨어 프로젝트에서 매우 중요한 것이다. 결함은 소프트웨어의 품질과 직접 관련이 있으므로 여러 의미에서 결함 데이터는 공수 데이터보다 더 중요하다[11, 12].

결함 데이터는 우선 프로젝트 관리를 위해 필요하다. 대규모 프로젝트는 수천개의 결함을 포함할 수 있는데, 이 결함은 다른 사람들이 프로젝트의 각각 다른 단계에서 발견하게 된다. 흔히 프로세스에서 결함을 고치는 사람과 결함을 발견하거나 보고하는 사람은 서로 다르다[13]. 보통 프로젝트는 소프트웨어가 최종 인도 전에 발견한 모든 또는 대부분의 결함을 제거하려고 할 것이다. 이런 시나리오에서 결함 보고와 해결은 비공식적으로 수행할 수 없다. 비공식적인 메커니즘의 사용은 발견한 결함에 대해 잊어버리는 결과를 가져올 수 있으므로 결함을 제거하지 않거나 다시 찾는데 추가 공수가 들어갈 수도 있다. 그러므로 적어도 결함을 기록해야 하고 해결할 때까지 추적해야 한다. 이런 절차를 위해서 결함의 징후, 의심되는 결함의 위치, 발견자, 해결자 등과 같은 정보가 필요할 것이다. 따라서 결함이란 프로젝트의 작업 산출물에서 발견되는 것으로 이 때문에 프로젝트의 목표를 달성하는데 부정적인 영향을 끼칠 수도 있다[14, 15].

결함에 대한 정보는 여러 가지 다양한 결함 검출 활동을 통해 발견한 결함의 개수도 포함 한다. 그래서 요구 사항 검토, 설계 검토, 코드 검토, 단위 테스트, 그리고 다른 단계에서 발견된 모든 결함 등을 기록한다. 프로젝트의 서로 다른 단계에서 발견된 결함 개수의 분포 데이터 역시 프로세스 역량 기준선(Process Capability Baseline) 생성에 사용한다[16]. 아울러 PDB(Process Data Base) 입력항목에 몇 가지 설명이 기록되는데, 예측에 대한 설명과 위험 관리에 대한 설명이 해당된다.

2.2 결함의 영향

발견된 결함이 기록되면 발견된 결함의 개수, 시작 상태의 결함의 비율, 그리고 프로젝트의 다른 쟁점 등을 집중적으로 분석할 수 있다. 이와 같은 결함 추적은 프로젝트 관리를 위한 최선의 실행 지침중의 하나이다[8, 9].

필요한 다른 유형으로 분석하려고 할 때 단순히 결함을 기록하는 것만으로는 충분하지 않다. 예를 들어 사용중인 프로세스의 품질 역량을 이해하려면, 테스트 중 발견한 결함과 인도 후에 발견한 결함을 결함의 다른 유형들과 분리해야 한다. 몇 퍼센트의 결함이 어디서 발견되었는지를 기록할 기준선을 만들기 위해, 그리고 실제 발견한 결함과 예측한 결함을 비교하기 위해 결함을 발견한 단계를 식별하면, 결함의 영향 분석도 가능해진다[17].

그러나 이런 정보만으로는 여전히 충분하지 못하다. 조직의 목표 중에 하나는 지속적인 프로세스를 개선해서 품질과 생산성을 개선하는 것이다. 품질과 생산성을 개선하는 접근 방법은 다양한 결함 검출 단계의 결함 제거 효율에 대해 연구하고, 어디에 개선의 여지가 있는지 알아내는 것이다. 결함 검출 단계의 결함 제거 효율은 단계를 수행할 때 나타난 결함의 전체 개수에 대한 그 단계에서 발견된 결함 개수의 비율이다. 결함 제거 효율을 높이면, 결함의 잠복 가능성이 낮다[3]. 이때 분명한 것은 생산성을 개선하기 위한 방법이 결함 제거 효율개선이라는 것이다. 결함 제거 효율을 계산하려면, 결함을 발견한 곳과 결함이 주입된 시점을 알아야 한다. 다시 말해서 기록되는 각 결함에 대해서 그 결함이 시스템에 주입된 단계에 대한 정보도 파악해야 한다.

〈표 1〉 결함 데이터

데 이 터	데이터의 설명	필수/선택
프로젝트 코드	결함이 발견된 프로젝트 코드	필수
설 명	결함에 대한 설명	필수
모듈 코드	모듈 코드	선택
프로그램명	결함이 발견된 프로그램명	선택
검출된 단계	결함이 검출된 단계	필수
주입된 단계	결함이 주입된/출처 단계	필수
유 형	결함의 분류	필수
심각도	결함의 심각도	필수
검토 유형	검토 유형	선택
상 태	결함의 현재 상태	필수
제출자	결함을 발견한 사람 이름	필수
소유자	결함을 소유한 사람의 이름	필수
제출일	결함이 소유자에게 제출된 날짜	필수
마감일	제출된 결함이 마감된 날짜	필수

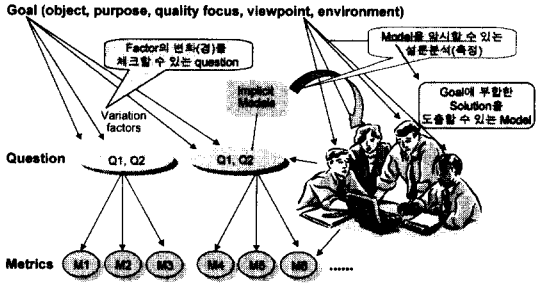
인포시스에서는 상용 결함 통제 시스템을 결함의 기록과 추적을 위해 프로젝트에서 사용했다. 결함 통제 시스템은

다양한 유형의 분석을 제공하며, 결함분석에도 사용한다. 또한 각 프로젝트에 대해 결함 통제 시스템을 설정하고, 모든 프로젝트 구성원이 프로젝트를 위한 결함 통제 시스템 데이터에 접근하는 것을 허용한다. 결함 통제 시스템을 설정하는 중에는 프로젝트에서 기록할 결함에 대한 정보의 유형을 상세히 기술하게 된다. 즉 필요에 따라서 프로젝트에 기록할 정보를 최적화한다. 인포시스는 모든 프로젝트에 필요한 몇 개의 필드를 필수 항목으로 명세 했다. <표 1>에서는 다양한 선택 또는 필수 항목을 제시하고 있다.

정량적인 품질관리를 위한 접근방법은 결함 예측을 이용하는 것이다. 이 접근 방법에서 인도된 결함 밀도의 관점에서 품질 목표가 설정된다. 품질 관리의 다른 측면, 즉 개발 프로세스 관리는 프로세스가 원하는 결함 밀도 목표를 달성할 수 있는 정량적인 방법으로 통제되어야 한다. 이 목표는 다양한 결함을 검출하는 활동에 의해 식별될 결함 예측, 그리고 실제 결함 개수와 예측된 결함 수준 비교 등을 통해 이루어진다. 일단 품질 목표를 설정했으면, 서로 다른 단계의 결함 수준을 예측하고, 예측을 맞춤으로써 목표 품질을 달성하게 된다. 이때 프로세스를 관리하기 위해 예측된 결함 수준은 실제 결함 수준과 비교하는 기준이 된다. 이것을 통해 개발 프로세스가 품질 목표를 달성하기 위한 방향으로 움직였는지의 여부를 평가하게 된다.

2.2 GQM(Goal/Question/Metrics)방법

GQM 접근 방법은 3단계로 정의되는데 구성은 (그림 1)과 같다.



(그림 1) GQM 방법

첫 번째 단계는 개념적인 단계로서 단계의 요소로는 대상(Object), 목적(Purpose), 관점(View point), 초점(Focus)등이 되고 이 단계는 목표(Goal)를 정의하는 단계가 된다[22, 23].

두 번째 단계는 운용단계로서 설정된 목표에 각각의 관점에서 정의 된 모델을 사용하여 합당한 질문(Question)을 하는 단계이다[23].

세 번째 단계는 정량적인 방법(Metrics)으로 질문에 답하는 단계를 말한다. 위의 3단계를 통해서 메트릭의 체계가 만들어 진다. 이러한 메트릭은 프로세스 개선을 위한 측정 도구로서 사용될 수도 있고 또한 EF의 기반이 되는 프로젝트 정보가 된다[21, 23].

2.4 결함 트리 분석

결함 트리 분석은 위험한 사건 또는 시스템 상태를 이끌어 낼 수 있는 사건들의 순차적이고 동시 결합인 그래픽 모형을 만들어 준다. 잘 개발된 결함 트리를 사용하면 다른 시스템 요소들에서 발생하는 일련의 상호 관련된 결함의 결과를 관측할 수 있다. 실시간 논리는 지정된 사건들과 대응하는 행동들로 시스템 모형을 만든다. 사건-행동 모형은 시스템 요소들과 그들의 시간에 관한 안전성 주장들을 시험하려고 논리 연산을 사용해서 분석한다. 페트리 넷 모형은 가장 위험한 결함을 발견하는데 사용될 수 있다[19].

위험이 식별되고 분석되었으면, 안전성에 관련된 요구사항이 소프트웨어에 명시될 수 있다. 즉, 명세는 바라지 않는 사건들의 목록과 희망하는 시스템 응답을 포함시킬 수 있다. 희망하지 않는 사건을 관리하는 소프트웨어의 역할은 이때 지적된다.

소프트웨어 신뢰성과 소프트웨어 안전성이 아주 밀접한 관계를 갖고 있다해도 그들간의 미묘한 차이를 이해하는 것은 중요하다. 소프트웨어 신뢰성은 소프트웨어 결함이 발생할 수 있는 확률을 구하기 위해서 통계적 분석을 사용한다. 그러나 결함의 발생은 필연적으로 위험이나 재난으로 귀결되지 않는다[20].

소프트웨어 안전성은 실패가 재난을 유발시킬 수 있는 조건에서 귀결되는 방법으로 시험한다. 즉, 실패는 공허한 상태로 고려되지 않고 전체 컴퓨터-기저 시스템에서 평가된다.

3. 본론 및 사례연구

본 장에서는 회사에서 실제 프로젝트를 수행하는 과정에서 발생하는 결함을 찾기 위하여, 설문서를 작성하게 된다. 따라서 결함을 검출하기 위한 설문서의 구조와 내용을 설명하고, 결과를 계층적 의사 결정법에 의하여 분석하게 된다. 그리고 분석된 결과를 결함 트리거화하기 위하여 결함 트리거를 제시한다.

설문서 대상 회사 및 프로젝트의 도메인으로는 디지털 시스템을 대상으로 수행하였으며, 네 개 회사의 21개 프로젝트를 대상으로 하였다.

3.1 결함 검출을 위한 GQM의 설문서

결함 검출 설문서의 목적은 회사에서 수행되는 프로젝트를 대상으로 하여 결함을 검출하기 위하여 작성하였다. 또한 결함을 찾아서 각 결함간의 원인을 분석하는데 목적이 있다.

설문서에서 조사하고자 하는 설문서 구조는 다음과 같이

각 생명주기에 존재하는 결함을 상세 범주로 나누어서 구분하였다. 각각의 생명주기별 설문서의 구조는 다음과 같다.

요구 사항 분석단계에서는 결함을 검출하기 위하여 상세 범주를 다섯 개로 나누어서 설문서를 작성하였다. 요구사항에서의 결함 식별을 위한 상세 범주들은 고객의 요구사항을 잘 반영하여 기능화하고, 얼마나 요구사항이 만족되고 있음을 확인하게 된다. 또한 요구사항이 일의성을 확인하는 관점에서 나누게 되었다. 따라서 상세 범주가 완전성, 정확성, 품질속성, 추적성으로 나누어지게 되었고, 기타사항에서는 네개의 상세범주를 지원하는 명세의 정형화된 형식과 일정에 의하여 명세가 이루어지는가를 조사하였다. 각 상세 범주의 설명은 <표 2>와 같다.

〈표 2〉 요구사항 분석 단계의 설문서 구조

생명주기	상세 범주	설 문 목 적
요구사항 분석	완전성	요구사항의 타당성과 이를 만족하기 위하여 알고리즘이 정의되었는가에 대한질의
	정확성	요구사항을 기능화 하기 위하여 구현 관점에서 오류가 있는가에 대한질의
	품질속성	요구사항 만족도를 확인하기 위하여 품질속성과 연관된 항목들에 관하여 질의
	추적성	요구사항의 일의성을 확인하기 위하여 질의
	기 타	상세 범주에 해당되지 않은 기타 사항을 질의

설계 단계의 결함 검출 설문서 구조는 <표 3>과 같다. 상세 범주의 구분은 구현단계까지 요구사항이 올바르게 반영되기 위하여 설계단계에서도 요구사항을 잘 반영하는가를 확인하는데 목적이 있다. 따라서 설계단계의 모듈 명세가 내·외부적으로 얼마나 명확히 정의되는가의 관점으로 나누게 되었다. 그 범주는 구조, 데이터, 정확성, 표준과 추적성, 논리, 인터페이스, 명료성 및 강건성으로 구분하였다.

〈표 3〉 설계 단계의 설문서 구조

생명주기	상세 범주	설 문 목 적
설 계	구 조	설계 단계의 구조가 명확하며, 통합 및 시험이 용이한가에 대한 질의
	데이터	모듈(컴포넌트)사이의 관계가 모호성 없게 정의하기 위하여 데이터가 초기화 되었는가에 대한 질의
	정확성	요구사항을 만족시키기 위하여 설계작업을 수행하는가에 대한 질의
	표준과 추적성	설계시 표준 프로세스의 이용과 요구사항단계에 대한 추적이 가능한가에 대한 질의
	논 리	논리적인 오류가 없는가에 대한 질의
	인터페이스	인터페이스가 명확하게 정의되었는가에 대한 질의
	명료성	설계단계의 산출물 표현시 일의성을 가지며, 명확히 표현 되었는가에 대한 질의
	강건성	예외상황에 대한 처리를 고려했는가에 대한 질의

코딩 단계의 설문서 구조는 <표 4>와 같다. 상세 범주의 구분은 추출된 요구사항이 설계단계를 거치면서 구현단계에서 얼마나 신뢰성 있게 기능으로 구현되는가를 확인한다. 그리고 코드 내부와 외부의 연관성을 분석하여 오류가 발생되었을 때, 이를 파악하고 대처하는 방안을 확인하는 관점에서 범주를 나누게 되었다.

〈표 4〉 코딩 단계의 설문서 구조

생명주기	상세 범주	설 문 목 적
코 딩	매 소 드	구현상에서 기능이 올바르게 수행되는가에 대한 질의
	정적 연관성	코드의 내부적인 내용에 대하여 오류가 있는지에 대한 질의
	동적 연관성	외부 장치와 연관되어 오류를 파악하고 이와 연관된 파일을 관리하는가에 대한 질의

시험 단계의 설문서 구조는 <표 5>와 같다. 설문서의 상세 범주는 시험이 계획된 일정과 환경 하에서 수행이 되고 있으며, 시험의 기준을 제시하여 다른 사람이 시험을 수행해도 일의성 있는 결과를 산출하고 있는가를 확인하기 위하여 구분을 하였다. 그리고 시험 항목도 고객의 요구사항을 만족시키기 위하여 수행을 하고 있는지와 일정에 따라서 자원을 할당하여 책임과 역할이 식별되는가를 확인하는데 목적이 있다.

〈표 5〉 시험 단계의 설문서 구조

생명주기	상세 범주	설 문 목 적
시 험	시험계획	시험 환경, 목적등이 일정에 의하여 계획되었는가에 대한 질의
	정확성과 완전성	시험을 정확하고 완전하게 수행하기 위하여 지침 및 기준이 정의 되었는가에 대한 질의
	표준과 추적성	시험이 요구사항과 연관되어서 수행되고 있는가에 대한 질의
	회귀 시험	코드의 변화와 버전에 대한 구별이 잘 식별되고 수정되는가에 대한 질의
	자원과 일정	자원할당이 일정계획에 기반하여 책임과 역할이 할당되는가에 대한 질의

실제 설문서는 요구사항 17문항, 설계 28문항, 코딩 11문항, 시험 26문항으로 작성되었다. 그리고 설문서의 응답은 기능이 수행되는가를 "예"와 "아니오"로 개발자가 혼동하지 않고 답변할 수 있도록 작성되었다. 또한 예라고 답을 한 경우, 질문에 해당되는 기능이나 기준이 존재하더라도 향후 문제가 발생할 수 있다. 따라서 이러한 문제가 발생하는 경우를 결함으로 구분하여 해당 문항의 결함이 전체 결함에서 차지하는 비중을 각 문항별로 조사하였다.

본 논문에서 결함 검출 설문서의 내용은 결함으로 검출된 설문서 문항만을 제시하고자 한다. 세부적인 설문서의 내용은 3.2절에서 분석시에 제시한다.

3.2 결함 검출 설문자료 분석

여러 개발자가 응답한 결함 검출 설문서에서 공통적으로 결함이 있다고 설문한 문항을 식별하였다. 식별된 설문서 문항을 생명주기와 범주별로 정리하면 다음과 같다.

요구사항 분석단계에서 결함으로 검출된 설문서를 분석하면 다음과 같다. 각 회사에서는 추출된 요구사항을 명세하여서 설계, 구현단계로 진행을 하고 있다.

그러나 요구사항이 변화하고 요구사항을 기능화 한다고 해도 의도했던 기능을 만족하는가를 확인하는 측면에서 미흡한 점이 있는 것으로 분석되었다.

조사 대상인 각 사는 대부분 한번 추출한 요구사항을 변화 없이 그대로 재사용하고 있었다. 따라서 고객의 요구사항을 추출하기 위한 방안과 향후 변화 발생 유무 및 요구사항이 만족되는가에 대한 확인 방안이 요구되었다. 요구사항 단계에서 검출된 결함은 다른 개발 단계에 비하여 많는 비중을 차지했다. 그 비중은 35%로서, 세부 비중으로는 완전성 25%, 정확성 10%의 결함 비율을 나타내었다. 이후에도 설계, 구현 및 시험 단계의 결함이 제시되지만, 요구사항에서 이후 단계의 결함이 원인이 된 것으로 분석되었다. 따라서 결함을 프로젝트 초기에 요구사항단계의 예상하여 대책을 세우기 위하여 노력을 하는것이 전체 프로젝트가 일정과 비용, 품질을 높이는 방안이 될 것이다.

〈표 6〉 요구사항 분석 단계에서 검출된 결함 문항

범 주	설 문 내 용
완전성	1. 작성된 요구사항이 타당성 있게 상세히 기술되었는가?
	2. 기능적인 요구사항을 만족시키기 위하여 기본적인 알고리즘이 정의 되었는가?
	3. 소프트웨어 요구사항 명세는 고객의 모든 요구사항 및 시스템 요구를 포함하는가?
	4. 다른 요구사항과의 일치성을 확인하기 위해 상호적으로 확인하는가?
정확성	1. 명세화된 오류 메시지가 일의성 있는 의미로 전달될 수 있는가?

설계 단계의 결함 검출 설문서의 내용을 살펴보면 다음과 같다. 설계단계는 추출된 요구사항이 시스템에서 기능화 되기 위하여 요구사항을 시스템화하는 단계이다.

응답자료에 의하여 설문서를 분석하면, 설계단계에서 데이터를 사용하기 위한 기반구조에 결함을 나타내었고, 모듈 상호간의 관계를 예측하기 위하여 인터페이스가 명확하게 정의될 필요가 있는 것으로 분석되었다. 그리고 전체 결함 중에서 설계단계가 차지하는 결함이 25%로 분석되었다. 또한 세부비중은 정확성 10%, 데이터, 논리, 인터페이스에서 각각 5%씩 검출되었다. 설계단계에서 검출된 설문서는 <표 7>과 같다.

코딩 단계의 결함은 주로 기능과 연관되어 코드상에서 발생하는 오류가 주를 이루었다. 그러나 각사에는 코딩 단계 이후로 진행될수록 소비자에게 생산품이 인도되는 단계에 근접하게 되므로, 코딩이후의 단계를 세밀하게 분석하고 대안을 마련하고 있었다.

〈표 7〉 설계 단계에서 검출된 결함 문항

범 주	설 문 내 용
데이터	1. 시스템 데이터의 기술에서 missing된 상세한 내용을 설명하였는가?
	2. 모든 데이터는 적절히 정의되고 초기화 되었는가?
정확성	1. 신뢰성과 성능 요구사항이 명시되었는가?
	2. 정의되지 않았거나, 불필요한 데이터 구조가 있는가?
	3. 모든 제약사항을 고려하였는가?
	4. 요구된 생산량, 반응시간, 정확성을 결정하기 위해서 분석하며, 이를 설계단계에서 구현할 수 있는가?
	5. 설계단계의 검증이 가능한가?
논리	1. missing 또는 불완전한 로직인가?
인터페이스	1. 모든 인터페이스는 명확하게 정의되었는가?

그러나 코딩 단계에서는 발견된 결함인 외부 스펙과의 연계성, 함수 호출등의 결함은 코딩 자체의 결함인 것도 존재했지만, 요구사항 분석과 설계단계의 작업이 미흡하여 발생한 경우가 많이 존재하였다. 코딩단계의 결함 비중은 전체에서 15%로 분석되었다. 세부 비중은 메소드 7%, 정적 연관성 2%, 동적 연관성 각각 6%로 결함이 검출되었다. 코딩 단계에서 결함으로 검출된 설문서는 <표 8>과 같다.

〈표 8〉 코딩 단계에서 검출된 결함 문항

범 주	설 문 내 용
메소드	1. 외부스펙과 연관된 기능이 올바르게 수행되는가?
	2. 코드가 외부의 재사용 컴포넌트나 라이브러리 함수를 호출하여 재사용 할 수 있는가?
	3. 반복적인 코드 블록을 요약해서 하나의 프로시져로 만들 수 있는가?
정적 연관성	1. 코드가 유지보수하기 쉽도록 명확한 주석형태를 갖추어서 문서화 되어 있는가?
동적 연관성	1. 파일접근을 시도하기 전에 파일의 존재 여부를 확인하였는가?

시험 단계의 결함은 시험시 결함의 허용한도와 출력기준이 명확히 적용되어 있지 않았다. 그리고 시험 계획시 자원과 인력 할당 및 역할의 계획이 미흡하였다. 시험 단계의 결함 비중은 전체 중에서 25%로 분석되었다. 세부 비중으로는 정확성 및 완전성 10%, 표준과 추적성 6%, 회귀시험 4%, 자원과 일정이 5%로 결함이 검출되었다.

시험 단계에서 결함으로 검출된 설문서는 <표 9>와 같다. 개발과정에서 존재하는 각 단계의 결함간의 연관성은 3.3절에서 제시한다.

〈표 9〉 시험 단계에서 검출된 결함 문항

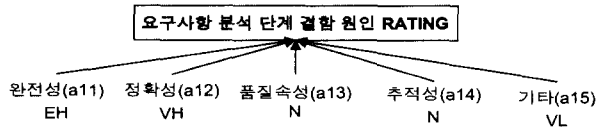
범 주	설 문 내 용
정확성 및 완전성	1. 요구사항의 목표 레벨과 코드 적용범위가 양적으로 명세화 되었는가?
	2. 시험계획에서 결함 허용한도와 통과/실패의 출력기준을 정했는가?
	3. 통합 시험 절차가 설계문서의 인터페이스에 설명되었는가?
	4. 신뢰성을 제시할 수 있는 시험 적용범위가 충분하가?
	5. 기능의 설명이 시험계획의 문서와 완전하고 정확하게 대응되었는가?
	6. 모든 시험 입구와 출구 조건은 충분하고 현실적인가?
	7. 시험문서에 없는 항목이 있는가?
	8. 시험결과의 인도가 정의되었는가?
	9. 시험계획이 완전하고, 정확하며 모호하지 않는가?
표준과 추적성	1. 시험계획은 모든 명세, 표준과 개발 문서를 작성하였는가?
회귀 시험	1. 충분하고 적절한 시험의 사례들을 이전의 시험과 구별하여 실행할 수 있도록 식별하는가?
	2. 모든 코드 변화는 충분한 연습과, 부분적인 인터페이스의 수정이 되는가?
자원과 일정	1. 모든 자원은 인간과 하드웨어, 소프트웨어를 고려하였는가?
	2. 개발 또는 메소드의 획득과 도구가 충분하게 스케쥴링이 되었는가?
	3. 시험 활동에서 관련된 사람의 역할과 책임을 식별하는가?

3.3 결함 원인 분석

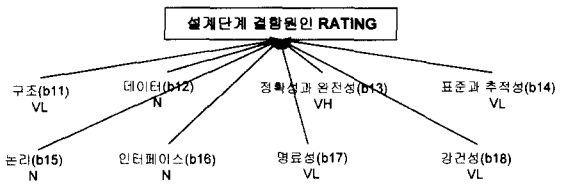
각 단계별 결함이 3.2절에서 식별되었다. 따라서 본 절에서는 식별된 결함원인을 분석하여 계량화하기 위하여 의사결정을 위한 계층구조를 도식화하여, 각 단계에서 항목이 차지하는 비중을 6개의 범위로 등급화 하였다. 등급화된 항목은 비교 메트릭스[21]를 기반으로 분석이 되며, 각 항목의 기하평균을 구하여 항목이 결함원인과 의 연관성을 식별하게 된다.

3.3.1 계층적 의사결정 구조도

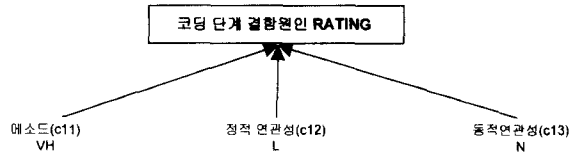
각 단계별 의사결정 구조도와 등급화된 항목은 다음과 같다.



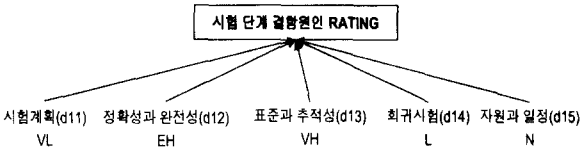
(그림 2) 요구사항 분석단계의 결함 원인 등급 구조도



(그림 3) 설계 단계의 결함 원인 등급 구조도



(그림 4) 코딩 단계의 결함 원인 등급 구조도



(그림 5) 시험 단계의 결함 원인 등급 구조도

결함 원인 등급 구조도는 네 개(요구사항 분석, 설계, 코딩, 시험)의 단계로 나누어서 각 단계 항목을 등급화 하였다. 등급은 6개 범위로 나누었으며, 내용은 다음과 같다. 6개의 범위는 Very Low(VL), Low(L), Norminal(N), High(H), Very High(VH), Extra High(EH)로 구분하였다. 6개 범위에서 EH로 갈수록 우선순위가 높아지며, VL로 갈수록 낮아짐을 의미한다.

(그림 2)의 경우, 요구사항 분석 단계에서는 3.2절에서 분석된 각 항목의 비중을 기반으로 등급화 하였으며, 그중에서 완전성이 가장 많은 결함을 내포하고 있었다. 따라서 완전성과 정확성은 다른 품질속성이나 추적성에 비하여 높은 결함과 진행되는 후행 단계들의 결함을 발생시킬 원인으로 분석되었다.

(그림 3)의 설계단계는 정확성과 완전성이 결함원인으로 식별되었고, (그림 4)의 코딩단계는 메소드, (그림 5)의 시험단계는 정확성과 완전성, 표준과 추적성이 결함원인으로 분석되었다.

3.3.2 결함 분석을 위한 비교 메트릭스

결함의 원인을 비교하기 위하여 메트릭스를 이용한다. 메트릭스는 각 중요도를 1부터 9까지 나열하여 관계를 규명하고자 한다[19]. 비교 메트릭스의 측정 기준표는 <표 10>과 같다.

〈표 10〉 비교 메트릭스의 측정 기준표

(a_{ij})	정 의
1	약 중요도가 같음
3	중요도
5	중요도
7	중요도
9	강 가장 중요도가 강함
2, 4, 6, 8	중간값
상호적인 관계	if $a_{ij} = w_i / w_j$, then $a_{ji} = 1 / a_{ij} = w_j / w_i$

<표 10>을 기준으로 각 단계별 항목간의 중요도를 계량화해서 나타낸다.

(그림 2)~(그림 5)는 <표 10>을 기준으로 분석하였다. 이때 각 단계의 항목 등급의 차이를 비교하게 된다. 예를 들면, 한 항목이 EH이고 다른 항목이 N이면 두 항목 사이의 중요도는 두 등급 차이가 존재하기 때문에 중요도 차이는 3배가 발생하게 된다. 이와 같은 관계는 <표 11>과 같다.

〈표 11〉 등급 대비 중요도 차이 비율 산출표

항목간의 등급차이	등급 대비 중요도 차이 비율
1	2배
2	3배
3	4배
4	5배
5	6배

<표 11>에 의하여 (그림 1)의 비교 메트릭스를 산출할 수 있다. 그 내용은 <표 12>와 같다.

〈표 12〉 요구사항 분석 단계의 항목간 비교 메트릭스

	a11	a12	a13	a14	a15
a11	1	2	4	4	6
a12	2	1	2	2	4
a13	1/4	1/2	1	1	2
a14	1/4	1/2	1	1	2
a15	1/6	1/4	1/2	1/2	1

<표 12>에서는 a11, a12, a13, a14, a15 열을 기준으로 다른 항목과 중요도를 비교하게 된다. 예를 들면, a11과 a12는 a11이 결함 항목으로서 2배가 많게 분석이 되었다. 그 이유는 a11은 EH이고, a12는 VH이므로, <표 11>에 의하여 항목간의 등급이 1차이가 발생하였다. 따라서 그에 따른 등급 대비 중요도 차이 비율은 2배가 된다. 결론적으로 a11은 a12보다 결함을 발생시킬 확률이 2배 많다는 의미가 된다. 이외에 설계와 코딩, 시험단계의 비교 메트릭스도 같은 방법에 의하여 산출하였으며 그 내용은 <표 13>, <표14>, <표 15>에 나타나 있다.

〈표 13〉 설계 단계의 항목간 비교 메트릭스

	b11	b12	b13	b14	b15	b16	b17	b18
b11	1	1/3	1/5	1	1/3	1/3	1	1
b12	3	1	1/3	3	1	1	3	3
b13	5	3	1	5	3	3	5	5
b14	1	1/3	1/5	1	1/3	1/3	1	1
b15	1/3	1	1/3	3	1	1	3	3
b16	1/3	1	1/3	3	1	1	3	3
b17	1	1/3	1/5	1	1/3	1/3	1	1
b18	1	1/3	1/5	1	1/3	1/3	1	1

〈표 14〉 코딩 단계의 항목간 비교 메트릭스

	c11	c12	c13
c11	1	4	3
c12	1/4	1	1/2
c13	1/3	1	1

〈표 15〉 시험 단계의 항목간 비교 메트릭스

	d11	d12	d13	d14	d15
d11	1	1/6	1/5	1/2	1/3
d12	6	1	2	5	3
d13	5	1/2	1	4	3
d14	2	1/5	1/4	1	1/2
d15	3	1/3	1/3	2	1

3.3.3 결함 원인의 중요도 산출

3.3.2에서 산출된 비교 메트릭스를 기반으로 각 단계별 항목의 중요도를 산출하고자 한다. 각 항목의 중요도가 높을수록 결함의 원인이 될 확률이 높으며, 산출은 기하 평균을 이용하여 산출하였다.

각 단계별 산출표는 다음과 같다.

〈표 16〉 요구사항 분석 단계의 결함 원인 중요도 산출표

항 목	산 출 값
a11	$(1x2x4x4x6)^{1/5} = 192^{1/5} = \mathbf{2.862}$
a12	$(2x1x2x2x4)^{1/5} = 32^{1/5} = \mathbf{2}$
a13	$(1/4\ x1/2x1x1x2)^{1/5} = 0.25^{1/5} = 0.758$
a14	$(1/4x1/2x1x1x2)^{1/5} = 0.25^{1/5} = 0.758$
a15	$(1/6x1/4x1/2x1/2x1)^{1/5} = 0.10^{1/5} = 0.631$

〈표 17〉 설계 단계의 결함 원인 중요도 산출표

항 목	산 출 값
b11	$(1x1/3x1/5x1x1/3x1/3x1x1)^{1/8} = 0.007^{1/8} = 0.538$
b12	$(3x1x1/3x3x1x1x3x3)^{1/8} = 27^{1/8} = \mathbf{1.510}$
b13	$(5x3x1x5x3x3x5x5)^{1/8} = 16875^{1/8} = \mathbf{3.376}$
b14	$(1x1/3x1/5x1x1/3x1/3x1x1)^{1/8} = 0.007^{1/8} = 0.538$
b15	$(1/3x1x1/3x3x1x1x3x3)^{1/8} = 3^{1/8} = \mathbf{1.147}$
b16	$(1/3x1x1/3x3x1x1x3x3)^{1/8} = 3^{1/8} = \mathbf{1.147}$
b17	$(1x1/3x1/5x1x1/3x1/3x1x1)^{1/8} = 0.007^{1/8} = 0.538$
b18	$(1x1/3x1/5x1x1/3x1/3x1x1)^{1/8} = 0.007^{1/8} = 0.538$

〈표 18〉 코딩 단계의 결함 원인 중요도 산출표

항 목	산 출 값
c11	$(1x4x3)^{1/3} = 12^{1/3} = \mathbf{2.288}$
c12	$(1/4x1x1/2)^{1/3} = 0.125^{1/3} = 0.5$
c13	$(1/3x2x1)^{1/3} = 0.667^{1/3} = \mathbf{0.874}$

〈표 19〉 시험 단계의 결함 원인 중요도 산출표

항 목	산 출 값
d11	$(1x1/6x1/5x1/2x1/3)^{1/5} = 0.006^{1/5} = 0.306$
d12	$(6x1x2x5x3)^{1/5} = 180^{1/5} = \mathbf{2.825}$
d13	$(5x1/2x1x4x3)^{1/5} = 30^{1/5} = \mathbf{1.974}$
d14	$(2x1/5x1/4x1x1/2)^{1/5} = 0.05^{1/5} = 0.549$
d15	$(3x1/3x1/3x2x1)^{1/5} = 0.667^{1/5} = \mathbf{0.922}$

각 단계의 항목을 기하 평균에 의하여 산출하였다. 산출 결과를 분석하면 요구사항 분석 단계의 경우, a11(완전성)과 a12(정확성) 항목이 결함 원인이 될 수 있다고 분석되었다.

설계 단계에서는 b12(데이터), 13(정확성과 완전성), 15(논리), 16(인터페이스) 항목이 결함 원인이 될 수 있다고 분석되었다.

코딩 단계에서는 c11(메소드)이 가장 높고, c13(동적연관성)이 높은 수치는 아니지만 결함 원인으로 분석되었다.

마지막으로 시험 단계에서는 d12(정확성과 완전성), 13(표준과 추적성)이 가장 높은 수치를 나타내었고, d15(자원과 일정)항목도 결함 원인으로 분석되었다.

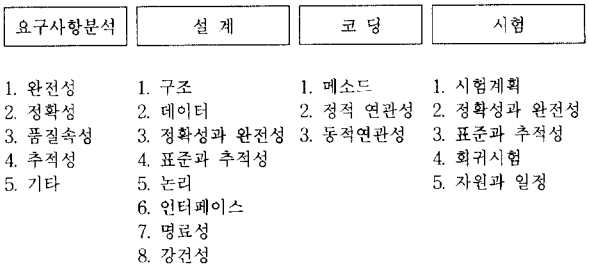
이와 같은 분석자료를 이용하여 결함간의 트리거를 3.4절에서 도식화 하겠다.

3.4 결함 트리거

결함을 분석하기 위하여 검출 설문서를 작성하여 설문을 실시하였다. 따라서 각 개발 단계별 결함이 검출되었으며, 결과를 분석하였다. 또한 검출된 결함을 활용하여 결함을 예방하기 위하여 결함 트리거를 제시한다.

결함 트리거는 결함 발생원인을 식별하여 결함간의 연관성을 찾아서 결함의 원인을 제시하기 위하여 사용하게 된다. 따라서 결함 데이터를 이용하여 결함 트리거를 완성하게 된다. 완성된 결함 트리거는 유사 프로젝트의 수행시 결함간의 연관성을 파악하여 발생할 결함을 예측하게 되고, 이를 통하여 결함을 예방하는 자료로 활용될 수 있게 된다.

본 절에서는 결함 트리거를 설계하기 위하여 각 개발 단계의 트리거 항목을 3.1절에서 설명한 상세범주를 기준으로 나타내었다. 결함 트리거의 구조는 (그림 6)과 같다.



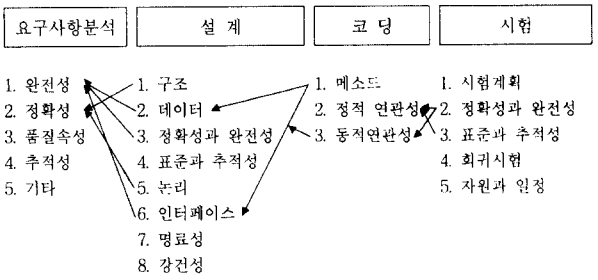
(그림 6) 결함트리거의 구조

결함트리거는 요구사항 분석, 설계, 코딩 및 시험 단계에서 전 단계의 원인을 분석하기 위한 것이다.

분석내용을 3.3절 내용에 의하여 종합하면, 요구사항 단계의 완전성과 정확성이 대부분의 결함 원인으로 분석되었다.

결함간의 연관관계에 대한 트리거는 (그림 7)과 같다.

설계단계의 데이터범주의 결함은 데이터 내용의 기술과 정의가 요구사항분석 단계에서 상세히 분석되지 않았다. 그리고 정확성과 논리 범주의 경우, 데이터의 상세한 내용이 부족하여 오류를 정확히 전달할 수 없었다. 정확성 범주는 요구사항 분석단계의 완전성, 논리범주는 정확성 범주가 원인으로 분석되었다.



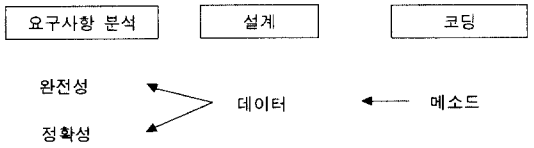
(그림 7) 결함간의 연관관계 트리거

인터페이스 범주는 요구사항간의 일치성의 확인이 미흡하여 인터페이스가 명확히 정의될 수 없었다. 따라서 요구사항의 완전성 범주가 원인으로 분석되었다.

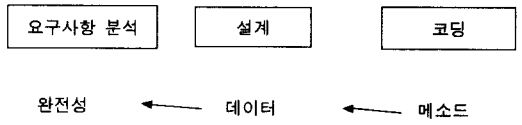
코딩 단계의 메소드 범주는 외부스펙과의 기능과 반복코드 문제가 데이터의 상세한 기술 부족과 인터페이스에 의한 상호 연관관계의 식별이 부족하여 설계단계의 데이터와 인터페이스 범주가 원인으로 분석되었다. 그리고 동적 연관성 범주는 제약사항의 고려가 부족한 것이 원인이 되어서 설계단계의 정확성 범주가 원인으로 분석되었다.

시험 단계의 정확성과 완전성 범주는 문서화와 제약사항의 고려가 부족하여 결함이 발생되었다. 따라서 코딩 단계의 정적 연관성과 동적 연관성 범주가 원인으로 분석되었다. 표준과 추적성 범주는 문서화의 부족으로 시험단계의 정적 연관성 범주가 원인이 되었다.

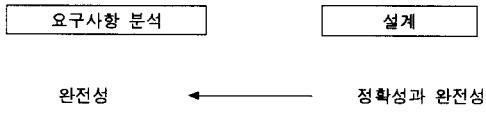
결함 트리거의 내용을 결론적으로 제시하면 요구사항 단계의 완전성 및 정확성 범주에 해당하는 결함을 제거하면 설계단계의 데이터, 정확성, 논리 및 인터페이스의 결함을 예방할 수 있는 것으로 분석되었다. 이와 같은 개념으로 트리거에서 인과관계를 개별적으로 도식화하면 다음과 같다.



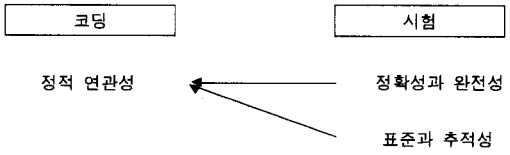
(그림 8) 상세 범주의 결함 트리거 1



(그림 9) 상세 범주의 결함 트리거 2



(그림 10) 상세 범주의 결함 트리거 3



(그림 11) 상세 범주의 결함 트리거 4

4. 결론 및 향후 연구 방향

3장에서는 결함에 관한 원인을 분석한 후에, 원인을 트리거화 하였다. 이와 같은 트리거를 이용하여 유사 프로젝트를 수행하는 경우에, 결함이 발생하는 단계와 결함 내용을 예측하여 전체 프로젝트에서 비용과 품질 및 일정에서 신뢰성 있는 결과를 산출할 수 있게 된다. 결함의 연관성이 분석되면, 이를 이용하여 유사 프로젝트 수행시 결함을 예측할 수 있게 된다. 따라서 프로젝트 계획과 일정 계획 시에 결함 발생원인을 고려하여 계획을 세울 수 있게 된다. 그러므로 전반적인 생산품의 품질이 향상되어 질 수 있게 된다.

향후 연구 방향으로는 결함의 세부적인 사항을 제시하여 결함 트리거를 프레임워크화 하고, 이를 기반으로 웹상에서 주요 항목을 입력하여 결함을 예측할 수 있는 시스템의 구현이 필요하다고 본다.

참 고 문 헌

[1] 이경환, HDC를 위한 모델링, 제 5회 한국정보과학회 소프트웨어공학 연구회, Feb., 2003.

[2] Annual Research Review, USC/CSE workshop reports, oct., 2002.

[3] Software Process Improvement Forum, KASPA SPI-7, dec. 2002.

[4] George Eckes, The Six Sigma Revolution, John Willey & Sons, 2001.

[5] SPICE Assessment in korea, The korea SPICE, May, 2002.

[6] 신경애, "결함중요도 단계를 고려한 소프트웨어 신뢰도 성장 모델에 관한 연구", 컴퓨터산업교육기술학회논문지, Vol.3 No.7 pp.837-844, 2000.

[7] Robert H. Dunn, "Software defect removal," McGraw-hill, 1984.

[8] Ram Chillarrege, Kothanda R. Prasad, "Test and Development Process Retrospective? a Case study using ODC Triggers," IEEE computer Society, 2002.

[9] N. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," IEEE Trans. Software Eng., 26, pp.797-814, 2000.

[10] McCall, P. K. Richards and G. F. Walters, "Factors in software quality," Vol.1, 2 and 3, Springfield VA., NTIS, AD/A-049-014/015/055, 1997.

[11] Vouk, Mladen, A., "Software Reliability Engineering," Tutorial Notes? Topics in Reliability & Maintainability & Statistics, 2000 Annual Reliability and Maintainability Symposium, Los Angeles, CA, January, 2000.

[12] Padberg, "A Fast Algorithm to Component Maximum Likelihood Estimates for the Hypergeometric Software Reliability Model," Asia-Pacific Conference on Quality Software APAQS 2, pp.40-49, 2001.

[13] Vamder Wiel, Votta, "Assessing Software Designs Using Capture-Recapture Methods," IEEE Transaction on Software Engineering, pp.1045-1054, 1993.

[14] Wohlin, Runeson, "Defect Content Estimations from Review Data," Proceedings International Conference on Software Engineering ICSE, pp.400-409, 1998.

[15] Gaffney, John, "Some Models for Software Defect Analysis," Lockheed Martin, Nov., 1996.

[16] L. Hatton, "Is Modularization Always Good Idea," Information and Software Technology, Vol.38, pp.719-721. 1996.

[17] B. Compton and C. withrow, "Prediction and Control of Ada Software Defects," J. Systems and Software, Vol.12, pp. 199-207, 1990.

[18] N. Fenton and M. Neil, "Software Metrics : Successes, Failures, and New Directions," J. Systems and Software, Vol.47, pp.149-157, 1999.

[19] Roger S. Pressman, "Software Engineering," Mcgraw-Hill International edition, 1997.

[20] Tim Kasse, Actin Focused Assessment for Software Process Improvement Artech House, 2002.

[21] THOMAS L. SAATY, "Analytical Planning," RWS PUBLICATIONS, 1985.

[22] V. Basili, G. Caldiera and D. Rombach, "The Experience Factory," Encyclopedia of Software Engineering, Wiley, 1994.

[23] V. Basili, G. Caldiera and D. Rombach, "The Goal Question Metric Approach," Encyclopedia of Software Engineering, Wiley, 1994.

[24] Victor R. Basili, "The Experience Factory and its Relationship to Other Improvement Paradigms," Lecture Notes in Computer Science 717, Software Engineering ESEC '93, 4th European Software Engineering Conference Garmish-Partenkirchen, Germany, September, 1993. December, 1991.

이 은 서

e-mail : esleee@object.cau.ac.kr

1999년 중앙대학교 컴퓨터공학과(석사)

2001년~현재 중앙대학교 컴퓨터공학과 (박사과정)

관심분야 : CBD, Formal method, Quality model, SPI(Defect Analysis)

이 경 환

e-mail : kwlee@object.cau.ac.kr

1964년 중앙대학교 이과대학 수학과(학사)

1966년 중앙대학교 이과대학 수학과(석사)

1980년 중앙대학교 대학원 수학과(박사)

1992년~JCSE '92 Conference Chairman 서울

1993년~1995년 중앙대학교 공과대학 학장

1994년~1995년 중앙대학교 정보산업대학원 원장

1998년~2000년 한국 온라인 가상 대학 운영 위원장

1998년~2000년 중앙대학교 정보통신연구소 소장

1999년~2000년 한국 정보과학회 회장

2000년~2001년 중앙대학교 총무처장

1982년~1983년 미국 AUBURN 대학교 객원교수

1986년~1986년 독일 BONN 대학교 객원교수

2001년~2002년 미국 USC 대학교 객원교수

1992년~현재 ISO/SC7/WG10 SPICE 한국 위원장

1992년~현재 한국표준 소프트웨어 공학(SC7) 위원

관심분야 : CBD Architecture, SPI(Defect Analysis), MBASE/CeBASE