객체지향 메트릭을 이용한 변경 발생에 대한 예측 모형

(A Prediction Model for Software Change using Object-oriented Metrics)

이 미 정 † 채 흥 석 †† 김 태 연 †††
(Mi Jung Lee) (Heung Seok Chae) (Tae Yeon Kim)

요 약 다양한 이유로 소프트웨어는 변경이 될 수 있으며 이는 유지보수 비용의 상승을 초래한다. 소프트웨어 메트릭은 클래스의 특성에 대한 정량적인 값으로서 유지보수 비용, 결함의 가능성 여부 등을 예측하는데 사용되고 있다. 본 논문에서는 대표적인 객체지향 메트릭과 산업체의 실제 소프트웨어 개발 과정에서 발생하는 변경 발생 횟수와의 관계를 제시한다. 규모, 복잡도, 결합도, 상속과 다형성 측면에서 7개의 메트릭이 사용되었으며, .NET 플랫폼 기반의 정보 시스템의 개발 과정에서 변경 발생 횟수에 대한 자료를 수집하였다. 본 논문에서는 다중회귀분석 기법을 이용하여 사용된 객체지향 메트릭으로부터 변경 발생 횟수를 예측하는 모형을 제시한다.

키워드 : 객체지향 메트릭, 변경 발생 예측

Abstract Software changes for various kinds of reasons and they increase maintenance cost. Software metrics, as quantitative values about attributes of software, have been adopted for predicting maintenance cost and fault-proneness. This paper proposes relationship between some typical object-oriented metrics and software changes in industrial settings. We used seven metrics which are concerned with size, complexity, coupling, inheritance and polymorphism, and collected data about the number of changes during the development of an information system on .NET platform. Based on them, this paper proposes a model for predicting the number of changes from the object-oriented metrics using multiple regression analysis technique.

Key words : Object-oriented metrics, software change prediction

1. 서 론

소프트웨어의 규모가 커지고 개발 기술이 발전하면서 소프트웨어 유지보수에 대한 관심이 고조되고 있다. 소프트웨어는 다양한 이유로 소스에 대한 변경이 발생한다. 사용자가 결함을 발견하여 그것을 제거하기 위해 변경이 일어나거나 사용자의 새로운 기능 요구 때문에 변경이 가해지기도 하며 또는 소프트웨어와 관계된 하드웨어 실행환경이 변하게 된 경우나, 때로는 소프트웨어 성능의 향상과 품질의 향상을 위해 코드 리팩토링(refactoring)에 의해서도 변경이 발생하게 된다. 이와 같이 소프트웨어가 개발된 후 사용자의 사용이 시작되면서부터 폐기될 때 까지 소프트웨어를 변경시키는 모든 활동을 소프트웨어 유지보수라고 한다[1].

체계적인 소프트웨어 유지보수 활동의 수행 및 관리를 위해서는 소프트웨어 유지보수에 소요되는 비용에 대한 예측이 필요하다. 현재는 유지 보수에 소요될 비용은 구체적인 근거 보다는 과거에 수행된 유지보수에 대한 경험을 바탕으로 산정하고 있다. 만약 유지보수에 대비한 변경발생 예측모델이 있다면 합리적인 소프트웨어 유지보수 비용을 산정하고 이를 바탕으로 체계적인 유지보수 활동에 대한 관리가 가능할 것이다.

소프트웨어 품질을 정량적으로 정확히 측정하기 위하여 객체 지향 메트릭을 사용하고 있으며, 이러한 소프트웨어 메트릭을 사용한 정량적인 방법은 잠재해 있는 설계 결점을 발견하고 결점을 정정하기 위한 변형을 찾는

· 이 논문은 부산대학교 자유과제 학술연구비(2년)에 의하여 연구되었음
† 정 회 원 : 대양전기공업(주) 전산팀
mjung007@naver.com
†† 정 회 원 : 부산대학교 컴퓨터공학과 교수
hschae@pusan.ac.kr
††† 학생회원 : 부산대학교 컴퓨터공학과
tykim@pusan.ac.kr
논문접수 : 2006년 11월 28일
심사완료 : 2007년 4월 16일

데 도움을 제공한다[2]. 그러므로 소프트웨어 메트릭은 오류 발생성이나 재사용성, 생산성을 포함한 변화요구에 따른 유지보수비용을 예측하는데 유용하다.

본 논문에서는 우리나라 한 제조업체의 영업 및 구매 관리 시스템에 유지 보수노력 예측을 실제적으로 적용해 봄으로써 단위 소프트웨어의 유지보수 변경 발생을 예측하고자 한다. 소프트웨어 사이클 동안 비용의 증가를 가져오는 변경 발생 횟수를 수집하여, 이에 영향을 미치는 객체지향 메트릭 중 어떤 것이 원인이고 어떤 것이 더 영향력이 큰 가를 분석한다면, 적절하게 비용을 사용 할 수 있다. 변경 발생 횟수의 가능한 요인으로 소프트웨어의 규모, 복잡도, 결합도, 상속성, 다형성을 설명하는 객체지향 메트릭을 사용하여 정량적인 수치를 산출한 후 상관관계를 파악한다. 그리고 널리 이용되는 통계적 기법인 다중회귀분석을 통하여 변경발생 횟수에 대한 적절한 예측 모형을 제시한다.

2. 관련 연구

소프트웨어 메트릭은 소프트웨어 공정과 소프트웨어 품질를 예측, 평가, 통제하기 위한 정량적인 수단으로 사용되고 있다. 소프트웨어 메트릭으로 소프트웨어 품질을 측정하기 위한 연구가 다양하게 제안되고 연구되었다[3-18]. 소프트웨어 메트릭의 효과성은 단순한 메트릭의 제시가 아니라 대상으로 하고 있는 소프트웨어 품질과의 관계를 얼마나 정확하게 표현하고 있는 지에 달려 있다. 즉 메트릭과 품질에 대한 명백하고 신뢰할 수 있는 관계가 있을 때만이 그 메트릭을 품질을 예측하는데 적용할 수 있을 것이다[19-21].

이렇게 메트릭과 품질과의 관계에 대한 연구의 중요성은 인식되고 있지만, 현실적으로는 자료 수집의 어려움으로 인해서 그다지 많은 실험적 연구가 수행되진 못하고 있다. 이제까지 소프트웨어 유지보수와 소프트웨어 메트릭 간에 관계를 파악하기 위한 몇몇 실험적 연구가 있었다. 표 1은 유지보수 노력 예측에 관한 실험적 연구를 요약하고 있다. 각 열은 유지보수 노력 예측과 관련된 대표적인 논문들이며 각 행은 이들 실험적 연구에 대한 비교 기준들이다. 각 실험적 연구에 대한 비교 기준으로는 실험에서 사용된 구체적인 유지보수 비용 산정 방법, 예측을 위하여 사용된 메트릭의 종류,. 통계적 분석 기법, 분석을 통하여 파악된 영향력있는 메트릭, 실험 대상 시스템의 특성을 사용하였다.

• 종속변수(유지보수 노력)

일반적으로 메트릭을 통하여 유지보수 노력을 예측하려고 할 때 유지보수 노력 자체에 대한 정의가 필요하다. 기존 실험적 연구에서는 유지보수 노력을 산정하기 위하여 변경된 행의 수, 변경 횟수, 유지보수에 소요된 시간 등을 사용하였다. Li와 Henry는 클래스당 변경된 라인의 수를 유지보수 노력으로 정의하였다[22]. Polo와 Piattini은 유지보수 노력을 긴급을 필요로 하는 수정요구와 긴급을 필요로 하지 않는 수정요구로 구분한 뒤

표 1 유지보수 노력 예측에 관한 실험적 연구

	Object Oriented Metrics that Predict Maintainability (Li and Henry:1993) [22]	Using Code Metrics to Predict Maintenance of Legacy Programs : A Case Study (Polo and Piattini:2001) [23]	Predicting Maintainability with Object-Oriented Metrics~An Empirical Comparison (Dagpinar and Jahnke:2003) [24]	Maintainability Prediction : A Regression Analysis of Measures of Evolving Systems (Hayes and Zhao:2005) [25]
종속변수 (유지보수 노력)	변경된 라인의 수	월별 변경요청 횟수가 평균값을 초과 하는지 유무	개인별 유지보수에 소요된 시간	유지보수시간 를 10단계로 분류한 PM지수
독립변수	DIT, NOC, RFC, LCOM, WMC, MPC, DAC, NOM SIZE1, SIZE2	LOC, 모듈의 개수	NIM, TNOS, NOC, AID, LCC, Direct/Indirect 결합도 메트릭	변경된 모듈의 비율, RDCRatio
분석기법	다중회귀 분석	로지스틱 회귀 분석	단계적 다중회귀 분석	다중회귀 분석
파악된 영향력 있는 메트릭	DIT, NOC, RFC, LCOM, WMC, MPC, DAC, NOM	LOC, 모듈의 개수	TNOS, NICMIC, NIMMIC, NIIC	변경된 모듈의 비율, RDCRatio
시스템 환경	Ada, 상용 소프트웨어, UIMS (User Interface System), QUES (Quality Evaluation System)	COBOL/CICS, 상용 소프트웨어, 수백만 행	JAVA, DSD(Distributed Software Development)	OO PROJECT, 5개의 수업에서 작성된 소프트웨어

월별 변경요구건수의 평균값을 기준으로 문제가 있는 소프트웨어와 문제가 없는 소프트웨어로 분류하였다[23]. Dagpinar와 Jahnke는 유지보수 활동을 perfective/adaptive, corrective로 분류하여 각각의 유지보수에 소요되는 개인별 시간을 유지보수 노력으로 정의하였다[24]. Hayes와 Zhao은 개인별 유지보수에 소요되는 시간을 이용하여 유지보수 노력을 10단계로 분류한 PM (perceived maintainability)을 유지보수노력으로 정의하였다[25].

• 독립변수(사용 메트릭)

유지보수 노력과의 관계 및 추정을 위하여 다양한 객체지향 메트릭이 사용되었다. 동일한 메트릭들이 사용된 것이 아니며 각 실험에서는 자체적으로 판단하여 유지보수 비용에 영향을 미칠 것으로 예상되는 메트릭들을 선택하여 실험을 수행하였다. 이는 유지보수 노력과 관계가 있다고 모든 연구자들이 인정하고 있는 메트릭이 아직 정의가 되지 않았으며 실험적 환경에 따라서 수집 가능한 메트릭을 선택한 것에 기인한다.

Li와 Henry는 객체지향 소프트웨어의 변경된 라인의 수를 예측하기 위하여 Chidamber와 Kemerer[26]가 제안한 메트릭의 일부를 포함하여 결합도를 측정하기 위한 메트릭 4가지(DIT, NOC, MPC, DAC), 인터페이스 증가를 측정하기 위한 NOM 메트릭, 그리고 소프트웨어의 크기와 관련된 메트릭 2가지(SIZE1, SIZE2)를 사용하였다[22]. Polo와 Piattini은 월별 변경요청 횟수가 평균값을 초과 하는지 유무를 예측하기 위하여 LOC와 모듈의 개수를 사용하였다[23]. Dagpinar와 Jahnke는 객체지향 메트릭를 사용하여 개인별 유지보수 시간을 예측하기 위하여 소프트웨어의 크기(NIM, TNOS), 상속(AID, NOC), 응집도(LCC), 결합도(Direct/Indirect Coupling Metrics)를 사용하였다[24]. Hayes와 Zhao은 소프트웨어의 유지보수 노력을 쉽게 예측하기 위한 방안으로 RDCRatio 메트릭을 제안하였다[25]. RADRatio값은 요구사항 변경에 소요된 시간에 디자인 변경에 소요된 시간을 더하여 계산된 총합에 코딩 변경에 소요된 시간을 나누어서 구하였다. 또한, 변경된 모듈의 비율을 사용하였다.

• 분석 기법

메트릭과 소프트웨어 품질과의 관계 및 예측 모형을 도출하기 위하여 회귀분석 기법이 일반적으로 사용되고 있다. 사용된 메트릭의 개수에 따라서 단일회귀분석과 다중회귀분석이 사용되고 있으며, 메트릭의 값의 분포에 따라서 로지스틱회귀분석이 사용되기도 한다. 본 논문에서 비교 분석한 논문 4편은 모두 다중회귀분석을 사용하였다. Li와 Henry, Hayes와 Zhao는 다중회귀분석을 사용하였다[22,25]. Polo와 Piattini은 로지스틱회귀분석을 사용하였다[23]. Dagpinar와 Jahnke는 다중회귀분석을 사용하였고 단계적 방법을 적용하였다[24]. 유지보수 노력을 소프트웨어 메트릭을 이용하여 정량화 하여 예측하기 위한 분석방법은 다양하며 다중회귀분석이 많이 사용되고 있다[27].

• 분석 결과

대부분의 경우에 회귀분석 기법을 통하여 영향력이 있는 것으로 간주되는 메트릭은 사용된 전체 메트릭의 일부분으로 분석된다. Li와 Henry는 객체지향방법으로 개발된 소프트웨어의 클래스당 변경된 라인의 수는 소스코드로부터 얻어진 객체지향 메트릭의 조합으로 예측이 가능하였다[22]. Polo와 Piattini는 COBOL/CICS 언어를 사용하여 개발된 시스템에서 월별 변경요청 횟수가 평균값을 초과 하는지 유무를 예측할 수 있었다[23]. Dagpinar와 Jahnke는 프로그램 크기와 Import Direct 결합도 메트릭으로 유지보수에 소요한 시간을 예측하였다[24]. 그러나, 상속, 응집도, Indirect/Export 결합도 메트릭은 그렇지 못하였다. Hayes와 Zhao은 유지보수에 소요한 시간 예측모델(MainPredMo)을 제안하였다[25]. 이 모델에서 사용된 RDCRatio 메트릭은 perceived maintainability(PM)과 비교하여 변경에 소요되는 노력을 객관적 방법으로 측정하였으며 PM 순위와 80% 정도의 유사성을 보여주었다.

3. 실험 환경

본 절에서는 소프트웨어의 변경 발생과 관련성을 파악하기 위하여 사용될 객체지향 메트릭을 소개한다. 그리고 실험 대상 시스템의 규모와 역할 구조 등을 설명하고 그러한 시스템에서 데이타 수집은 어떠한 방법으로 하였으며 어떻게 메트릭을 계산하였는지 설명한다.

3.1 실험대상 메트릭

이 실험의 대상인 소프트웨어는 객체지향언어 C#으로 개발되었고, C#코드의 메트릭을 계산하기 위해서 객체지향 소프트웨어 설계도구인 Together를 이용하여 가장 적합한 메트릭을 산출하였다. 이 실험에서는 클래스의 규모, 복잡도, 결합도, 상속성, 다형성 측면을 나타내는 1개 또는 2개의 메트릭을 사용하였다.

• 규모(크기) 측정 메트릭 : LOC(Lines Of Code)

클래스의 규모(크기)를 측정하기 위한 메트릭으로서 LOC를 사용한다. LOC는 클래스의 줄 당 공백과 주석은 제외하고 프로그램의 머리부, 선언 부분, 비실행문을 포함한 라인수이다. LOC는 프로그래밍 언어의 특징이나 개발자의 코딩방법에 따라 큰 차이가 난다는 단점에도 불구하고 측정이 용이하고 직관적이기 때문에 전통적으로 가장 많이 사용되어 왔다.

• 복잡도를 측정하기 위한 메트릭 : CC(Cyclomatic

Complexity), WMPC(Weighted Methods Per Class)

복잡도가 높을수록 개발 시에 오류발생 가능성이 높고 프로그램에 대한 이해가 어려우므로 유지보수 또한 어려움이 있을 수 있다. 사이클로메틱 복잡도(Cyclomatic Complexity)는 메소드 등의 제어흐름에 대한 복잡도에 대한 척도로서, 메시지의 개수와 가중치에 의해서 정의된다. 하나의 메소드 안에서 사용되어진 메시지를 가중치에 따라 계산하는데, 과학적인 근거가 클래스 자체의 메트릭으로서 Chidamber와 Kemerer는 WMC (Weighted Methods per Class)를 구하는데 필요한 가중치를 정의하지 않고 메트릭의 적용 시에 정의하도록 남겨두었다. 여기서는 메소드의 가중치에 따라 WMPC로 정의하였다.

- 결합도 메트릭 : DAC(Data Abstraction Coupling)

결합도가 클수록 클래스간의 의존성이 높으므로 오류발생 가능성이 높아서 유지보수비용이 많이 드는 것이 일반적이다. Li와 Henry가 제안한 DAC[22]는 클래스간의 데이타 추상화에 의한 클래스간의 의존성을 의미한다. DAC는 클래스에서 사용되는 클래스 수로서 정의되는데 한 클래스와 집합의 관계가 있는 클래스의 수를 뜻한다.

- 상속성(길이)을 측정하는 메트릭 : DOIH(Depth Of Inheritance Hierarchy)

DOIH는 상속계층의 깊이를 나타낸다. 모든 인터페이스 타입은 1의 깊이를 갖고 모든 클래스들은 슈퍼클래스의 깊이보다 1크다. 일반적으로 DOIH가 증가하면 복잡도가 증가하고 행동양식을 예측하기 어려워지고 소프트웨어의 견고함이 낮아진다.

- 다형성을 측정하는 메트릭 : NOAM(Number Of Added Methods), NOOM(Number Of Overridden Methods)

NOAM은 클래스에 의해 상속과 재정의를 제외하고 추가되는 오퍼레이션의 수를 나타내며, 이 메트릭이 나타내는 값은 주어진 클래스의 기능성이 그 부모클래스의 그것과 점점 다르게 나타남을 의미한다. 같은 성격의 NOOM(Number Of Overridden Methods)은 재정의된 메소드의 수를 나타내며, 서브클래스에서 부모클래스의 메소드를 재정의함으로써 부모 클래스의 기능을 확장하여 다형성을 나타낸다.

3.2 실험대상 시스템

객체지향 메트릭과 변경 발생에 관한 연구를 위하여 사용된 대상 시스템은 한 제조업체에서 구축하고 있는 영업 및 구매 관리 시스템이며, 운영체제는 Windows 2003이고 IIS 웹서버가 설치되어 있다. 그림 1은 실험대상 시스템의 아키텍처를 보여준다. 이 시스템은 고객으로부터 견적요청이 들어오면 기업내부의 사용자가 견적을 입력하고, 입력된 견적에 의하여 계약정보가 생성된다. 이 계약정보는 웹 기반의 전자결재 흐름에 따라서 내부결재를 받은 후 구매 관리 시스템으로 정보를 보내게 된다. 그리고 구매 관리 시스템은 계약에 맞는 자재를 구매하기 위해 협력업체에게 구매 요청 정보를 보낸다. 그림 1에서 ERP 서버 중 영업 관리/구매 관리로 명기된 부분이 실험 수행 대상이다.

그림 2는 실험대상 시스템인 영업 관리 / 구매 관리 시스템의 세부 구성이다. 실험대상 시스템은 3-Tier형태로 아래의 그림과 같이 구성되어 있다.

사용자 인터페이스는 Windows기반과 Web기반의 2가지로 이루어져 있다. 비즈니스 로직 부분은 Web-Service/COM+ 형태의 C# 코드로 작성되어 있으며, 프레젠테이션 부분은 ASP.NET과 UI Object가 C# 으로 되어 있다. 이 소프트웨어는 구현된 기능에 따라 6개의 네임스페이스(intranet, mm, myclass, myDev, sd, SmartService)를 가지고 있으며, 총 94개의 클래스로 이루어져 있다. 표 2는 실험 대상 시스템의 기본정보를 나타낸다. 실험 대상 시스템을 구성하는 총 94개의 클래

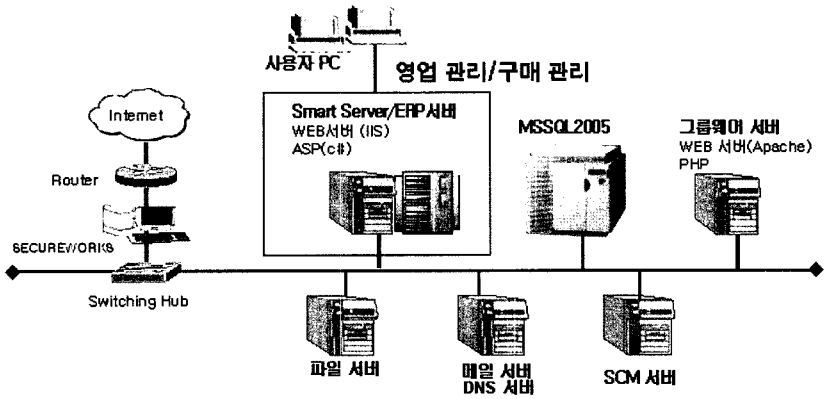


그림 1 실험대상 시스템의 구성

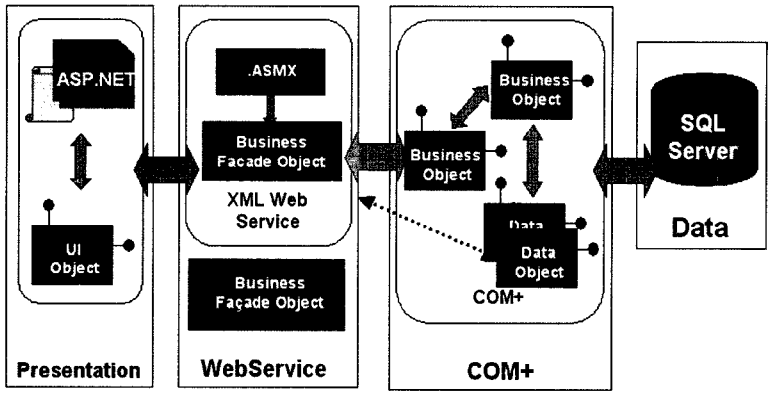


그림 2 영업 관리 / 구매 관리 시스템의 아키텍처

표 2 영업 관리 / 구매 관리 시스템의 기본 정보

	인스턴스 변수	오퍼레이션
평균	7.33	9.73
최소값	0	0
중간값	5	10
최대값	33	33

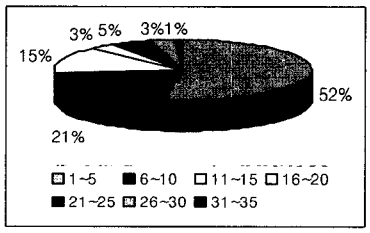
스에 대하여 클래스의 인스턴스 변수와 연산의 수를 파악하였다.

그림 3은 실험 대상 시스템의 각 기본정보에 대한 구성 분포를 보여준다. 실험 대상 소프트웨어의 클래스별 인스턴스 변수의 수를 살펴보면 5개 이하의 변수를 사용한 클래스가 50개로서 전체의 52%에 해당한다. 6개 이상 10개 이하의 인스턴스 변수를 사용한 클래스로 21%의 비율을 차지하며, 11개 이상 15개 이하를 사용한 클래스들로 15%이다. 가장 많은 변수를 가지고 있는 클래스는 33개까지 가지고 있으나 전체의 1%밖에 되지 않는다. 연산의 수의 경우에는 5개 미만이 38%를 차지하고, 6개 이상 10개 미만이 14%, 11개 이상 15개 미만이 20%를 나타낸다. 요약하면 클래스들이 전체분포에서 많은 수보다는 적은 수의 변수와 연산을 가지고 있는 비율이 높다.

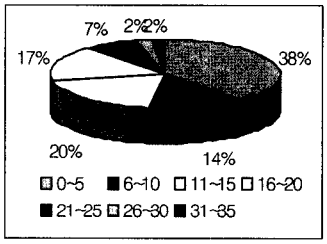
3.3 변경 발생 데이타 수집

본 실험은 인위적 환경 설정이 아니라 중소기업의 전산실에서 일상적인 실제 작업을 바탕으로 하고 있다. 개발 공정 중에서 통합 시험 단계에서 자료를 수집하였다. 수집된 자료는 사용자에 의해서 요청된 요구사항의 변경과 결함의 발견으로 인한 수정, 개발자의 모듈 최적화를 위한 변경을 포함한다. 시스템 개발자 5명이 2달의 기간 동안에 발생한 각 변경에 대하여 클래스 단위로 기록하였다. 표 3은 수집된 데이타의 일부를 보여 준다. 데이타의 모듈 명을 명시한 후 클래스 명을 작성하고, 변경이 발생한 날짜를 적고 어떤 내용으로 변경이 일어났는지 기술하였다.

표 4는 네임스페이스별 클래스 수와 변경발생 횟수를 나타낸다. 즉, 변경발생이 일어나는 클래스를 파악하기



(a) 클래스 별 인스턴스 변수 분포



(b) 클래스 별 연산의 수 분포

그림 3 클래스의 규모에 대한 분포

표 3 수집된 변경 발생 데이타의 예

모듈	클래스명	날짜	변경작업내용
수주전검토 등록	Sd202007	4월12일	영업부서조회는 영어본부조회로 쿼리시 테이블변경
견적수주이관	Sd020208	4월12일	디자인 pop창 관리자에게만 오픈 되도록 전역변수설정
견적수주이관	Sd020208	4월13일	시스템 BaseForm상속원 sd.Basic에서 mydev.Basic으로 수정
…….	…….	…….	…….

표 4 네임스페이스별 클래스 수와 변경 발생 횟수

네임스페이스명 (기능별분류)	포함 클래스 수	변경발생 횟수
intranet	5	18
mm	21	129
myClass	34	39
myDev	12	16
sd	17	99
SmartService	5	11
합계	94	312

쉽게 기능별 묶음인 네임스페이스별 요약을 보여준다. 이 표에서는 총 6개의 네임스페이스에 대하여 각 네임스페이스에 포함된 클래스의 수와 해당 네임스페이스의 클래스에서 발생한 변경의 수를 보여 준다.

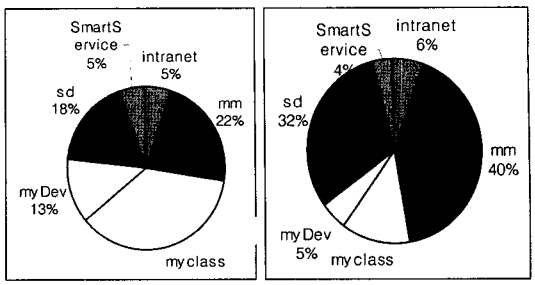
• myClass, myDev, SmartService는 공통 모듈에 해당한다. 전적으로 비즈니스 로직만 담당하고 있다. SmartService는 서버와 클라이언트와의 통신을 위해 사용되므로 구성하는 클래스 수도 5개 이며, 변경 발생 횟수는 11번 일어났다. 이에 비해 대부분의 프레젠테이션 부분과 연관되어 비즈니스 부분만 처리 하고 있는 것이 myClass와 myDev이다. myClass 네임스페이스내의 클래스들은 시스템 설계 시에 전체 시스템의 크기와 성능을 고려하여 어떠한 클래스에서도 접근이 가능하도록 만들어진 공통 클래스들로서 가장 많은 클래스를 포함하고 있다.

• Intranet 네임스페이스내의 클래스 수는 SmartService와 같지만 변경발생이 일어난 횟수는 더 많다. 여기에 포함되는 클래스들은 프레젠테이션 부분을 같이 포함하고 있어서 웹페이지와 연동이 되기 때문에 소프트웨어 전체 클래스에서 차지하는 비중은 작지만 변경 발생 횟수는 SmartService 보다 더 많다.

• mm 네임스페이스가 포함하는 클래스의 개수는 21개로서 네임스페이스 중에서는 두 번째로 많은 클래스를 차지하고 있다. 그러나 변경 발생 수는 무려 40%로 가장 높은 비율을 차지한다. sd와 더불어 프레젠테이션 부분을 처리하고 있으며 mm은 구매 부분을 처리하는 모듈들이다.

• sd 네임스페이스 또한 영업 관리 부분을 처리하기 위한 프레젠테이션 부분을 같이 처리하고 있으며 클래스 수는 17개이고 변경 발생 수는 38%로서 두 번째다.

그림 4는 네임스페이스별로 클래스의 수와 변경 발생의 수의 분포를 보여준다. 요약 하면 구매/영업 관리의 실제 로직을 담당하는 mm과 sd 네임스페이스 변경발생의 78%가 나타나고 있다. 그리고 myClass 13%, intranet 6%, myDev 5%, SmartService 4% 순으로 변경 발생은 분포하였다.



(a) 클래스 분포 (b) 변경 발생 분포

그림 4 네임스페이스 별 클래스와 변경 발생 분포

3.4 객체지향 메트릭 계산

본 실험에서는 C# 코드로부터 메트릭 결과를 계산하기 위하여 Together를 이용하였다. Together는 Java 플랫폼뿐만 아니라 .NET 플랫폼 기반의 C#코드로 구현된 실험대상 시스템을 모델링하고 이해하는데 필요한 기능을 지원하는 CASE 도구이다. Together는 UML 모델링 기능뿐만 아니라, 다양한 메트릭 계산을 지원하고 있다. 그림 5는 Together를 이용하여 측정된 메트릭 값을 보여준다. 좌측 Explorer은 클래스들을 나타내고 우측 Designer는 클래스들의 상속관계를 다이어그램으로 나타내고, Editor는 선택된 클래스의 소스코드를 나타낸다. 그리고 하단의 Message Pane에는 측정된 메트릭의 값을 보여준다.

4. 실험결과 및 분석

이 절에서는 수행된 실험 결과와 분석 결과를 소개한다. 먼저 계산된 각 메트릭 값과 수집된 변경 발생 횟수를 요약한 후에 둘 간의 관계를 살펴본다. 각 메트릭들이 소프트웨어의 변경에 영향을 미치는가를 확인한 후, 상관관계를 바탕으로 다중회귀 모형을 구성한다.

4.1 객체지향 메트릭과 변경 발생 횟수 측정 결과

표 5는 각 클래스별 메트릭 값과 변경 발생 횟수를 보여준다. 각 클래스는 CC, DAC, DOIH, LOC, NOAM, NOOM, WMPC 메트릭으로 구분하여 값을 보여주고 있으며, 그 클래스에 해당되는 변경 발생 횟수도 함께 표시하고 있다.

표 6은 메트릭 측정 결과를 합계, 평균, 최소값, 중간값, 최대값으로 요약한 것을 보여 준다.

그림 6-12는 사용된 7개의 메트릭에 대하여 메트릭 값 별로 클래스 수의 분포를 보여준다.

• CC 메트릭

CC 메트릭 값은 분포를 살펴보면 전체적으로 0~120 사이에 분포한다. 가장 높은 비율을 차지하는 값은 0~10 사이의 값으로서 10 이하의 작은 수치에 약 36%가 분포하고 있다. 그 다음이 21~30 사이의 값으로 15%를

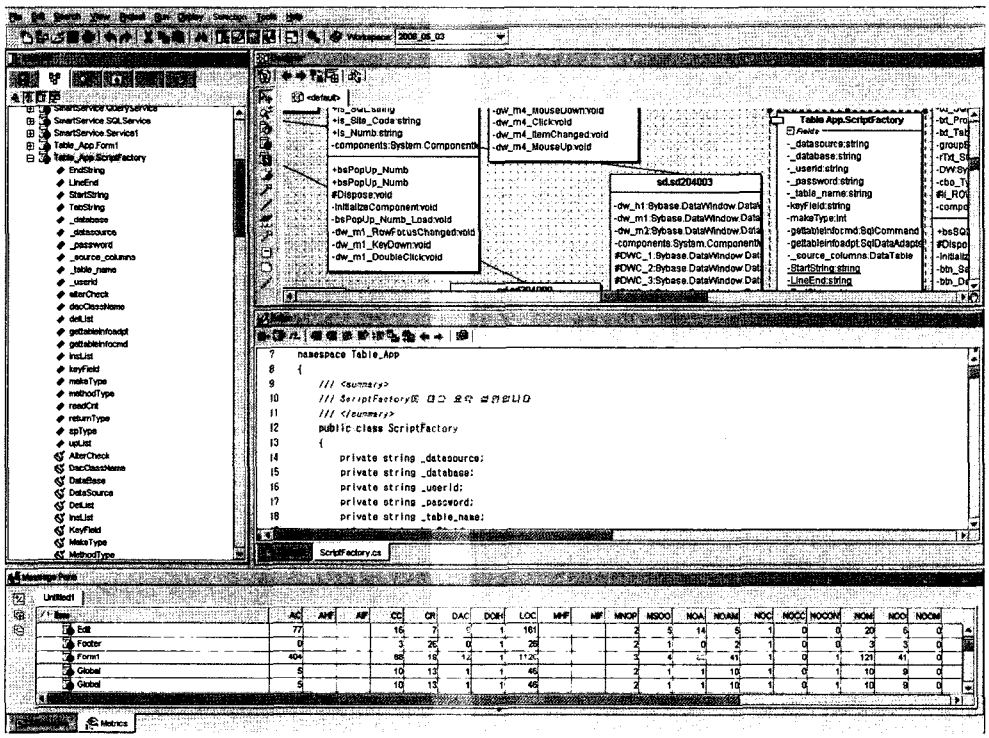


그림 5 Together를 이용한 메트릭 측정

표 5 메트릭과 변경 발생 횟수

	CC	DAC	DOIH	LOC	NOAM	NOOM	WMPC	변경발생
intranet.conn	6	0	1	74	4	0	12	1
intranet.Content	50	0	1	426	14	0	36	10
intranet.download	3	0	1	37	2	0	6	1
intranet.list	8	0	1	75	5	0	11	5
intranet.photo	1	0	1	33	1	0	3	1
mm.mm301001	107	6	2	854	32	1	72	14
mm.mm301002	57	5	2	441	23	1	59	11
mm.mm301003	50	2	2	408	19	1	49	9
mm.mm301004	26	4	2	266	13	1	33	7
mm.mm301005	16	3	2	167	8	1	20	4
mm.mm302001	56	3	2	409	17	1	43	4
mm.mm302002	26	3	2	212	12	1	33	4
mm.mm302003	76	3	2	471	19	1	50	4
...	...	...	...	...	...	...	...	...

표 6 각 메트릭의 기본 정보

	CC	DAC	DOIH	LOC	NOAM	NOOM	WMPC
평균	25.6	2.25	1.38	240.5	9.947	0.37	25.91
최소값	0	0	1	1	0	0	0
중간값	20	3	1	201	10	0	26
최대값	117	9	2	854	33	1	90

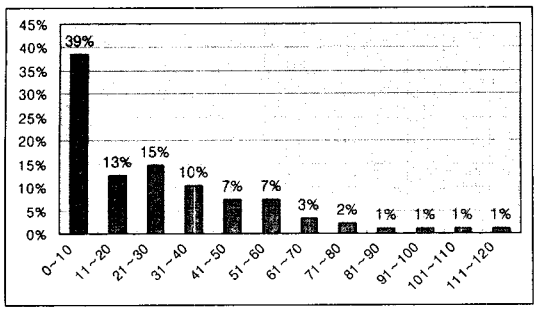


그림 6 CC 메트릭의 분포

차지하며, 11~20 사이 값이 13%로 세 번째를 차지한다. 그리고 81이상의 값은 단지 5% 만을 차지하고 있다. 81이상의 높은 값이 나온 클래스는 구매 관리 모듈인 mm 네임스페이스에서 3개, 영업 관리 모듈인 sd 네임스페이스에서 2개가 있다.

• DAC 메트릭

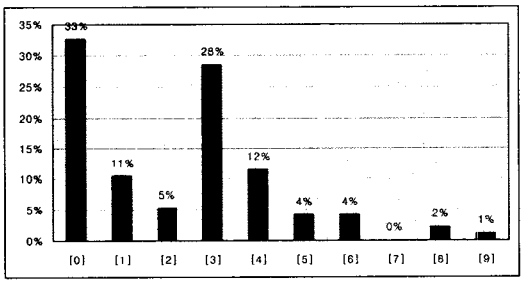


그림 7 DAC 메트릭의 분포

DAC 메트릭의 분포를 보면 메트릭 값이 0인 클래스가 31개로서 33%이다. DAC가 클래스간의 결합도를 나타내는 것인데 이 31개 클래스는 무관하다고 판단된다. 그리고 두 번째가 3의 값으로 28%를 차지하고 그 다음으로는 메트릭 값이 1, 4인경우가 11%, 12%로 분포한다.

• DOIH 메트릭

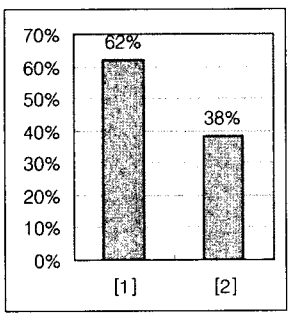


그림 8 DOIH 메트릭의 분포

상속의 깊이를 나타내는 DOIH 메트릭의 값인 1인경우가 59개의 클래스로 62%이고 나머지가 2인 경우로 나타났다. 즉 실험 대상 시스템에서는 상속의 깊이는 모두 2 이하이다.

• LOC 메트릭

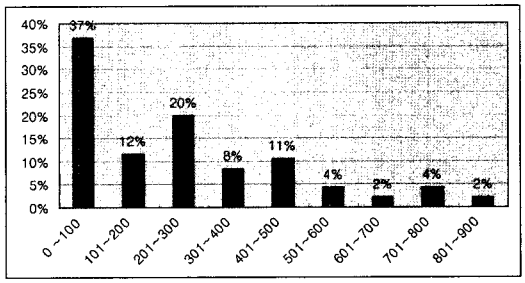


그림 9 LOC 메트릭의 분포

LOC의 분포를 보면 100라인 이하가 37%를 차지하며 제일 높은 비율을 차지하고 있으며 201~300라인 사이가 20%로 그 다음이고, 나머지가 201~300 사이가 12%, 401~500사이가 11% 로서 81%가 500 라인 이하이다.

• NOAM 메트릭

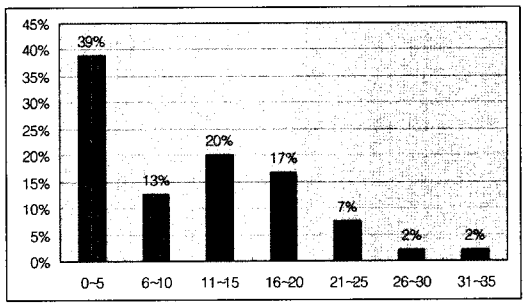


그림 10 NOAM 메트릭의 분포

다형성을 나타내는 NOAM의 값의 분포는 5이하가 39%, 11~15사이가 20%, 16~20사이가 17%로를 나타내며, 20이하에서 89%를 차지한다.

• NOOM 메트릭

소프트웨어의 다형성을 나타내는 또 다른 메트릭 NOOM의 분포를 보게 되면 0, 1 두 가지만 사용되고 있다. 0인 경우가 63%, 1인경우가 37%이다.

• WMPC 메트릭

WMPC 값의 분포는 0~90사이이다. 10이하의 값이 34개로 36%를 차지하며 가장 많고, 21~30사이가 17개로 18%, 그 다음이 41~50사이 값으로 13,14%를 차지하고 있다.

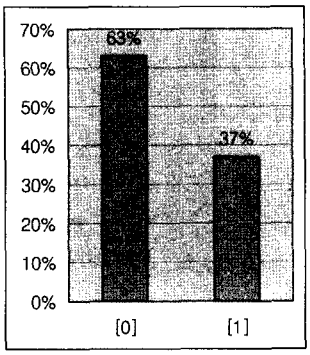


그림 11 NOOM 메트릭의 분포

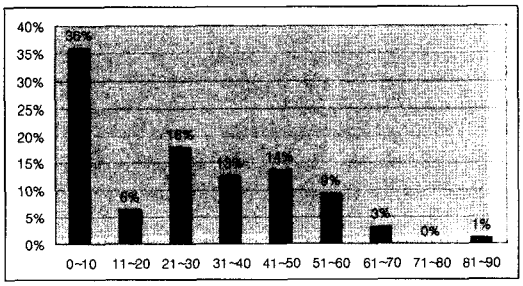


그림 12 WMPC 메트릭의 분포

4.2 객체지향 메트릭과 변경 발생의 상관관계

본 실험에서는 각 객체지향 메트릭과 변경 발생의 상호연관성을 파악하기 위해서 상관계수를 계산하였다[28]. 표 7은 클래스별 변경 발생 횟수와 각 메트릭과의 상관계수이다.

변경 발생 횟수와 각 메트릭은 모두 양의 상관관계를 나타내고 있다. 즉 각 메트릭은 변경 발생의 횟수와 순방향의 관계가 있음을 뜻한다. 특히 CC 메트릭은 상관계수가 0.740으로 가장 높게 나타난다. 이는 클래스가 복잡할수록 변경 발생 횟수도 많이 일어난다고 해석될 수 있다. 다형성을 나타내는 NOOM은 0.727로 다형성을 많이 가질수록 변경 발생도 많이 일어난다고 볼 수 있다. 클래스 상속길이를 나타내는 DOIH 값은 0.701로 나타나서 상속의 깊이가 깊을수록 연관된 클래스가 많으므로 변경 발생 횟수도 많이 일어나게 된다. 다형성을 나타내는 NOAM 메트릭도 변경발생횟수와 0.675의상관계수를 가진다. NOOM 메트릭은 재정의된 메소드의 수로 다형성의 값을 나타내며 NOAM 메트릭이 클래스에 의해 더해지는 메소드의 수로서 계산한 값이므로 재정의된 메소드의 수가 더 상관관계가 높다고 볼 수 있다. WMPC 메트릭은 복잡성을 나타내는 메트릭으로서 0.637이며 메트릭 값이 커질수록 변경 발생도 더 일어난다고 볼 수 있다. 소프트웨어 크기를 나타내는 LOC메트릭의 경우도 0.635이고 상관관계가 있다고 보이며 소스코드의 크기가 클수록 변경이 더 일어난다고 할 수 있다. 마지막으로 DAC 메트릭과의 상관 계수는 0.36으로서 다른 메트릭 값에 비해서 비교적 작은 값을 가진다. DAC 메트릭은 클래스간의 의존성을 나타내는 결합도 값이며 일반적으로 결합도가 높을수록 변경발생횟수도 높아진다고 볼 수 있으나 다른 메트릭에 비해서는 상관관계가 낮은 편이라는 결과를 보여준다.

4.3 회귀분석을 통한 예측

객체지향 메트릭과 수집된 변경 발생 횟수가 어느 정도 상관관계가 있음이 위에서 밝혀졌다. 본 실험에서는 추가적으로 다중회귀 분석기법을 이용해서 각 메트릭이 변경발생 횟수에 미치는 영향을 분석하고 객체지향 메트릭을 이용한 변경 발생 횟수 예측모형을 제시한다.

4.3.1 단계적 회귀분석

다중회귀분석에서는 고려해야 할 점이 있는데 회귀식에 포함된 독립변수들끼리 높은 상관관계를 가질 경우가 많다. 이처럼 독립 변수들 간의 상관관계가 높을 때, 이것을 변수들 간의 다중공선성(multicollinearity)이 있다고 말한다. 독립변수들 사이에 다중공선성이 존재한다면 추정된 회귀계수가 통계적으로 유의하지 않게 나타날 가능성이 있기 때문에 다중공선성의 발생을 방지해야 한다. 불필요한 변수들이 들어있는 완전모형(full model)보다는 필요한 변수들만 들어있는 축소모형(reduced model)을 통해 모형을 보다 간명하게 하는 것이 바람직하다[28]. 이를 방지하기 위하여 미리 변수들 간의 상관계수를 파악하여 상관관계가 높은 두 변수들 중 하나를 회귀분석모형에서 제거하거나 단계적 회귀방법을 이용하여 상관관계가 높은 변수들 중 가장 설명력이 있는 독립변수만을 모형에 포함시켜는 것이다.

표 8은 각 메트릭간의 상관계수를 행렬로 나타낸다. 상관계수행렬(correlation matrix)은 변수들의 관계를 한 눈에 알기 쉽게 하기 위하여 각 메트릭 간의 상관계수를 행렬형태로 정리한 것이다. 유의성 검정의 결과는 상관계수행렬에서 각 모집단상관계수가 '0'이라는 가설검정에 대한 결과를 기본적으로 p값이 5% 유의수준으로 귀무가설이 기각되면 '*'로, 1% 유의수준에서 기각되면 '**'이다. 표 8에서 각 메트릭간의 상관계수를 살펴보았을 때 CC 메트릭의 경우 LOC, NOAM과는 매우 높

표 7 메트릭과 변경 발생과의 상관관계

메트릭	CC	DAC	DOIH	LOC	NOAM	NOOM	WMPC
상관계수	0.740	0.356	0.701	0.635	0.679	0.727	0.637

표 8 상관분석-상관계수행렬

	CC	DAC	DOIH	LOC	NOAM	NOOM	WMPC
CC	1.0000						
DAC	0.5287 **	1.0000					
DOIH	0.6208 **	0.3293 **	1.0000				
LOC	**0.9345** **	0.5902 **	0.5498 **	1.0000			
NOAM	**0.9061** **	0.6830 **	0.5922 **	**0.8887** **	1.0000		
NOOM	0.6448 **	0.3545 **	**0.9778** **	0.5743 **	0.6189 **	1.0000	
WMPC	**0.8872** **	0.6761 **	0.5356 **	**0.8750** **	**0.9895** **	0.5639 **	1.0000

(*: 유의수준 0.05에서 유의, **: 0.01에서 유의)

은 상관으로 0.9이상을 보이며, DOIH의 경우도 NOOM과 0.9이상의 매우 높은 상관관계를 나타내고 있으며, NOAM과 WMPC간에도 0.9이상으로 매우 높은 상관관계가 파악되었다.

본 실험에서는 다중공선성의 문제를 해결하기 위하여 가장 많이 사용되고 있는 단계적 방법을 사용하였다[29]. 즉 각 객체지향 메트릭을 후보 변수로 두고 이 중에서 회귀모형절차에 따라 독립변수를 선택한다. 다중회귀분석은 변수가 많고 복잡하기 때문에 국내의 통계 소프트웨어인 S-Link라는 통계 툴을 이용하여 계산하였다[30]. 회귀 추정치는 95% 신뢰구간에서 의미를 갖는다.

시작 단계에서는 종속변수에 대한 상수항을 먼저 구한다. 표 9는 그 결과를 요약하여 보여 주고 있다. 상수항의 회귀계수가 3.2로 나왔고 t-검정에서 유의수준이 0.000으로 나왔으므로 매우 유의하다. 그리고 아직 독립변수를 적용하지 않은 상태이며 회귀모형에 포함되지 않은 변수를 보면 독립변수들의 F값 유의수준의 p값도

표 9 단계적 회귀방법 : 시작단계

(a) 회귀모형의 결정계수

변수	추정치	표준오차	t값	p값
상수항	3.2000	0.3294	9.713	0.0000

(b) 회귀모형에 포함되지 않은 변수

변수이름	부분상관계수	F값	p값
CC	0.7249	102.989	0.000
DAC	0.3655	14.343	0.000
DOIH	0.7051	91.972	0.000
LOC	0.6090	54.834	0.000
NOAM	0.6890	75.315	0.000
NOOM	**0.7310**	**106.738**	**0.000**
WMPC	0.6222	58.749	0.000

구하였다.

독립변수들 중에서 상관계수가 높으면서 p값이 유의한 변수 순으로 하나씩 선택하여 회귀분석 식의 독립변수로서 고려된다. 표 9(b)를 보면 NOOM 메트릭이 상관계수가 가장 높고 p값이 0.000으로 매우 유의하므로 먼저 선택하게 된다. 이 과정을 반복적으로 적용하면 NOOM, CC, LOC 메트릭이 선택된다. 표 10은 최종적으로 이 세 개의 메트릭이 선택된 결과를 보여준다.

4.3.2 실험적 예측모형

단계적 회귀분석 과정을 거쳐 다음과 같은 예측모형을 도출하였다.

변경발생횟수 =

0.9608 + 2.8457 (NOOM) + 0.1048 (CC)~0.0062 (LOC)

(단, 결정계수는 0.6691)

NOOM, CC는 양의 영향을 주고 LOC 메트릭은 음의 영향을 준다. 추정치가 2.8457인 NOOM 메트릭은 클래스 내에서 재정의된 메소드 수이다. 즉 슈퍼클래스에서 만든 메소드를 서브 클래스에서 재정의한다는 것은 슈퍼클래스와 서브클래스가 연결되어 있어서 슈퍼 클래스의 변화는 서브클래스까지 변경하게 하는 파급효과가 있다. 즉 NOOM 메트릭 값의 크기에 변경 발생 횟수가 영향을 받게 되는 것이다. 추정치 0.1048을 가지는 CC 메트릭은 복잡도를 나타낸다. 복잡도가 높을수록 개발시에 오류발생 가능성이 높아서 변경 발생 역시 가능성이 높아지게 되는 것이다. 0.2262회귀 계수를 가지는 LOC는 소스코드의 라인수를 나타내는데, 소스코드의 라인이 길어서 규모가 크면 변경 발생이 좀 더 일어난다.

4.4 실험결과 분석

본 절에서는 본 실험의 결과를 간략히 요약하고 기존 연구와의 비교를 통하여 유사성과 차별성을 설명한다.

- 국내 중소기업의 전산실에서 실제적으로 수행된 소프트웨어 유지보수 작업을 소프트웨어의 규모, 복잡도,

표 10 단계적 회귀방법 : 최종 단계

(a) 회귀모형의 결정계수

중상관계수(R)	결정계수(R-square)	결정계수의 변화량	추정의 표준오차(s)	오차제곱합의 변화량
0.8180	**0.6691**	0.0247	1.87729	23.9218

(b) 분산분석표

요인	제곱합	자유도	평균제곱	F값	P값
회귀	648.4962	3	216.1654	61.3371	**0.0000**
오차	320.7038	91	3.5242		
전체	969.2000	94			

(c) 회귀모형에 포함된 변수

변수이름	추정치	표준오차	t값	p값
상수항	0.9608	0.2830	3.394	0.0010
NOOM	2.8457	0.5252	5.418	**0.0000**
CC	0.1048	0.0222	4.729	**0.0000**
LOC	-0.0062	0.0024	-2.605	**0.0107**

(d) 회귀모형에 포함되지 않은 변수

변수이름	부분상관계수	F값	p값
DAC	0.0414	0.154	0.695
DOIH	-0.0542	0.265	0.608
NOAM	0.0684	0.423	0.517
WMPC	0.0118	0.012	0.911

결합도, 상속성, 다형성을 설명하는 객체지향 메트릭을 사용하여 정량적인 수치를 산출한 후 다중회귀분석을 수행하였다. 그 결과 소프트웨어의 규모를 측정한 LOC, 복잡도를 측정한 CC, 다형성을 측정한 NOOM 메트릭이 변경 발생 횟수와 상관관계를 가졌다. 다중선형회귀모형에서 변수로 선택되지 않은 DAC 메트릭은 표 7에서와 같이 비교적 낮은 상관관계를 보이므로 다중회귀식의 변수로는 적합하지 않았다. DOIH, NOAM, WMPC 메트릭은 변경 발생과 비교적 높은 상관관계를 보였으나, 독립변수 간에도 높은 상관을 나타냈다. 즉 다중공선성의 문제를 야기하므로 두 개 이상의 변수가 높은 상관관계를 가질 때는 수치가 높은 하나를 선택하고 하나는 제외시켜야 한다. 표 8에서 NOAM, WMPC 메트릭은 LOC 메트릭과 DOIH는 NOOM과 높은 상관관계를 나타낸다. 따라서, 독립변수가 여러 개인 다중선형회귀식을 이용한 예측 모델에서는 그 영향이 중복 발생되므로 유의 수준을 만족하면서 상관계수가 좀 더 높은 NOOM, CC, LOC 메트릭만이 영향을 끼치는 변수로 선택되었다.

- 본 실험에서는 소프트웨어의 크기와 복잡도 메트릭을 사용하여 소프트웨어의 변경 발생 횟수를 예측할 수 있었다. 이 결과는 이전 연구에서도 공통적인 결과를 나타내었다[23,31]. Polo와 Piattini는 소프트웨어의 크기와 변경요청 횟수 및 변경에 걸린 시간은 상관관계가 있다고 결론을 내리고 예측모델을 제안하였다[23]. Welker와 Oman은 LOC, CC, Lines of Comments를 이용하여 Maintainability Index(MI)를 제안하였다[31]. 다형성과 관련 있는 NOOM 메트릭 역시 소프트웨어의 변경 발생 횟수를 예측할 수 있었다. 다형성과 관련 있는 또다른 메트릭인 NOAM의 경우는 메소드의 추가가 소프트웨어의 크기를 증가시키므로 LOC와 비례관계이다. 따라서 독립변수인 NOAM과 LOC는 높은 상관관계를 가지므로 종속변수와 더 높은 상관관계를 가지는 LOC에 그 영향이 반영되었다.
- 본 실험은 우리나라 중소기업의 전산실에서 실제적으로 수행된 소프트웨어 유지보수 작업에서 데이타를 수집하였다. 또, 현재 현업에서 많이 사용하고 있는 .NET 기반의 3-tier 웹 환경에서 운용되는 시스템을 대상으로 실험을 진행하였다. 따라서 실험의 환경과 동일한 소프트웨어 개발 환경인 중소기업의 소프트웨어 유지보수 작업에 실험결과를 직접적으로 이용할 수 있을 것이다. 소프트웨어 유지보수 작업에서 변경발생 횟수를 예측한 Dagpinar와 Jahnke는 perfective/adaptive 유지보수 노력에 관하여 예측 하였다[24]. 본 논문에서는 요구사항의 변경, 결함의 발견 및 수정, 개발자의 모듈 최적화에 의한 변경발생 횟수와 같이 좀 더 광범위한 범위에서의 변경발생 횟수를 예측하였다. 이전의 연구에서는 소프트웨어 유지보수 작업에 소요되는 유지보수 시간에 관하여 연구하였다[32]. 하지만 연구의 규모가 작고 단순히 상관관계만을 분석하였다.

5. 결론 및 향후 연구 방향

본 논문에서는 회귀분석기법을 적용한 변경 발생 예측 모형을 이용하여 객체지향 메트릭과 변경 발생 횟수 간의 관계를 실제 시스템의 개발 과정에서 수집된 데이타를 대상으로 조사하였다. 이번 실험적 연구에서 발견된 연구결과를 요약하면 다음과 같다.

첫째, 객체지향 메트릭과 소프트웨어의 변경 발생 횟수는 상관관계가 있었다. 객체지향 메트릭은 소프트웨어의 특성을 나타내는 값으로 규모, 결합도, 복잡도, 상속성, 다형성의 5가지 범주에서 계산하였을 때 선택된 7개의 메트릭은 모두 변경 발생 횟수와 순방향의 상관관계를 보여 주었다.

둘째, 변경발생횟수를 추정하기 위한 모형에서는 5가지 범주의 7개 메트릭 중 NOOM 메트릭, CC 메트릭, LOC 메트릭이 변경 발생 횟수의 예측에 큰 영향을 미치는 것으로 드러났다. DOIH 메트릭, NOAM 메트릭, WMPC 메트릭은 변경 발생 횟수의 예측에 영향을 미쳤다. 그러나 단계적 회귀분석에서 해당 메트릭과 높은 상관관계를 가지는 메트릭 중 가장 높은 상관관계를 가지는 NOOM, LOC, CC 메트릭에 그 영향이 반영되었다. DAC 메트릭은 변경 발생 횟수와 낮은 상관관계를 가지므로 회귀분석모델에서 제외 하였다. 실험대상에 적용 가능한 변경 발생 예측 모형을 설정함으로써 소프트웨어의 유지보수 비용 산정 예측에 관한 토대를 마련할 것으로 기대된다.

이 실험적 연구과 관련된 향후 연구 방향은 다음과 같다.

객체지향 메트릭만을 사용하여 변경 발생을 예측하는 것은 영향을 줄 수 있는 다른 요인들이 존재하므로, 예측 결과가 기대수준에 미치지 못할 가능성도 존재한다고 추측된다. 이러한 한계점을 보완하기 위해서는 표본 소프트웨어의 크기와 유형을 다양하게 하여 추가적인 실험을 수행할 필요가 있다. 또한, 변경 발생은 요구사항의 변경/추가/삭제, 플랫폼의 변화, 품질의 개선 등과 같은 다양한 이유로 시작된다. 향후에는 변경 발생의 원인을 구분하여 메트릭과의 관계를 조사할 계획이다.

참 고 문 헌

[1] N. F. Schneidewind, "The State of Software Maintenance," IEEE Trans. Software Eng., Vol. 13, No 3, pp. 303-310, Mar., 1987.

[2] 이병정, 객체지향 설계 행위를 보존하는 메트릭 기반 재구조화 기법, 정보과학회논문지: 소프트웨어 및 응용 제 30권 제10호, 2003.

[3] T. J. McCabe, A Complexity Measure, IEEE Trans, Software Eng. 2, 308-320, 1976.

[4] S. Henry and D. Kafura, Software Structure Metrics Based on Information Flow, IEEE Trans, Software Eng. 7, 510-518, 1981.

[5] M. H. Halstead, Elements of Software Projects: Management, Measurement and Estimation, Yourdon Press, New Jersey, 1982.

[6] W. G. Bail and M. V. Zelkowitz, Program Complexity Using Hierarchical Abstraction Computers, Comp. Lang. 13, 109-123, 1988.

[7] P. Robillard and G. Boloix, The Interconnectivity Metrics: A New Metric Showing How a Program is Organized, J. Syst. Software 10, 29-39, 1989.

[8] R. Adamov and L. Richter, A Proposal for Measuring the Structural Complexity of Programs, J. Syst, Software 55-77, 1990.

[9] V. Basili, L. C. Briand, and W. Melo, "A Validation of Object Oriented Design Metrics as Quality Indicators," IEEE Trans. Software Eng., Vol. 22, pp. 751-761, 1996.

[10] A. Binkley and S. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," In Proc. 20th Int'l Conf. Software Eng., pp. 452-455, 1998.

[11] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis," IEEE Trans. Software Eng., Vol. 24, pp. 629-639, 1998.

[12] L. C. Briand, J. Wuest, S.Ikonomovski, and H. Louis, "Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study," In Proc. Int'l Conf. Software Eng, pp. 345-354, 1999.

[13] M.H. Tang, M.H. Kao, and M.H. Chen, "An Empirical Study on Object Oriented Metrics," In Proc. Sixth Int'l Software Metrics Symp., pp. 242-249, 1999.

[14] L.C. Briand, J. Wuest, J.W. Daly, and D.V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object Oriented System," J. systems and Software, Vol. 51, No. 3, pp. 245-273, 2000.

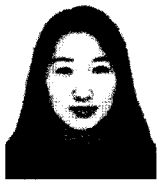
[15] M. Cartwright and M. Shepperd, "An Empirical Investigation of an Object-Oriented Software System," IEEE Trans, Software Eng., Vol. 26, No. 7, pp. 786-796, 2000.

[16] K. El Eman, W. Melo, and J.C. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," J. Systems and Software, Vol. 56, pp. 63-75, 2001.

[17] R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK Metrics for Software Defects," IEEE Trans. Software Eng., Vol. 29, pp. 297~310, April, 2003.

[18] Nikolaos Tsantalis, Alexander Chatzigeorgiou, "Predicting the Probability of Change in Object-Oriented Systems," IEEE Trans. Software Eng.,

Vol. 31, pp. 601-614, July, 2005.
[19] H. D. Rombach, A Controlled Experiment on the Impact of Software Structure on Maintainability, IEEE Trans. Software Eng. SE-13, 89-94, 1987.
[20] S. Wake and S. Henry, A model based on software quality factors which predicts maintainability, in Proceedings: Conference on Software Maintenance, pp. 382-387, 1988.
[21] H. D. Rombach, Design Measurement: Some Lessons Learned, IEEE Software 17-25, 1990.
[22] W. Li and S. Henry, "Object Oriented Metrics that Predict Maintainability," J of Systems and Software, Vol. 23, pp. 111-122, 1993.
[23] M. Polo, M. Piattini and F. Ruiz, Using code metrics to predict maintenance of legacy programs : a case study, Proceedings of the IEEE International Conference on Software Maintenance, 2001.
[24] M. Dagpinar and J. H. Jahnke, Predicting Maintainability with Object-Oriented Metrics~An Empirical Comparison, Proceedings of the 10th Working Conference on Reverse Engineering, 2003.
[25] J.H. Hayes and L. Zhao, Maintainability Prediction : A Regression Analysis of Measures of Evolving Systems, Software Maintenance, ICSM'05, Proceedings of the 21st IEEE International Conference, 601-604, Sept. 2005.
[26] S. R. Chidamber and C. F. Kemerer, "A Metric Suite for Object Oriented Software System," IEEE Trans. Software Eng., Vol. 20, No. 6, pp. 476-493, 1994.
[27] Ghosheh, E., Qaddour, J., Kuofie, M., Black, S, A Comparative Analysis of Maintainability Approaches for Web Application, Computer Systems and Applications, IEEE International Conference, 2006.
[28] 강현철, 한상태, 이은수, SPSS 데이타 분석과 활용, 자유아카데미, 2002.
[29] 姜金植, EXCEL활용 현대 통계학, 博英社, 2005.
[30] 신봉섭, S-Link와 함께 배우는 통계자료의 분석, 도서출판그린, 2004.
[31] Welker, K.D. and Oman, P.W. Software Maintainability Metrics Models in Practice, Journal of Defense Software Engineering, Volume 8, Number 11, November/December 19-23, 1995.
[32] 정우성, 채홍석, "객체지향 메트릭과 유지보수성과의 관계에 대한 실험적 연구", 정보처리학회논문지 D, 제 13권-D권 제 2호, 2006.



이 미 정

1995년 한국해양대학교 응용수학 학사 2006년 부산대학교 산업대학원 전산학 석사. 2000년~2003년 다성정보기술㈜ 개발팀. 2004년~현재 대양전기공업㈜ 전산팀. 관심분야는 소프트웨어 개발 방법론, 소프트웨어 메트릭, 소프트웨어 유지보수, 웹2.0 아키텍쳐

채 홍 석

1994년 서울대 원자핵공학 학사. 1996년 한국과학기술원 전산학 석사. 2000년 한국과학기술원 전산학 박사. 2000년~2003년 ㈜ 동양시스템즈 기술연구소 선임연구원. 2003년~2004년 한국과학기술원 전산학과 초빙교수. 2004년~현재 부산대학교 컴퓨터 공학과 조교수. 관심분야는 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭, 소프트웨어 유지보수, 미들웨어 설계, 프로덕트라인 공학

김 태 연

1998년~2001년 ㈜세안IT 병원사업부 사원. 2003년~2005년 ㈜에이치원 병원사업부 대리. 2007년 부산대 정보컴퓨터공학 학사. 2007년~현재 부산대학교 컴퓨터 공학과 석사. 관심분야는 데이타 모델 설계, 대용량 데이타 솔루션, 객체지향 설계, 소프트웨어 테스팅, 소프트웨어 메트릭