객체지향 메트릭과 유지보수성과의 관계에 대한 실험적 연구

정 우 성[†] · 채 흥 석[††]

요 약

유지보수는 소프트웨어 생명 주기에서 대부분의 비용이 소요되는 중요한 활동이다. 기존에 소프트웨어 유지보수 활동에 소요되는 비용의 예측을 위하여 다양한 소프트웨어 메트릭이 제안되었다. 본 논문에서는 객체지향 소프트웨어의 유지보수성과 기존에 제안된 메트릭과의 관계를 파악하기 위해서 수행된 실험의 결과를 소개한다. 실험에서는 메트릭으로서 LCOM, RFC, DAC, LOC를 사용하였으며 클라이언트/서버 및 웹 기반 시스템을 대상으로 유지보수 활동 시 실제로 소요되는 시간을 측정하였다. 본 실험을 통해서 메트릭과 유지보수 비용과의 관계에 대한 기존의 일반적인 주장을 실제 시스템의 유지보수 활동을 통해서 확인한 결과 기존의 주장과 달리 일부 시스템의 경우에는 기존의 주장을 뒷받침하지 못할 수 있음을 확인하였다. 구체적으로 말하면 소프트웨어 아키텍처, 설계 패턴 등이 적용되는 개발된 최근의 소프트웨어에서는 메트릭과 유지보수 활동과의 관계에 대한 추가적인 많은 연구가 필요함을 확인하였다.

키워드 : 객체지향 메트릭, 유지보수성

An Empirical Study of Relationship between Object-oriented Metrics and Maintainability

Woo Seong Jung[†] · Heung Seok Chae[††]

ABSTRACT

Software maintenance is an important and very expensive activity in software life cycle. To estimate the maintainability cost of software, many software metrics have been proposed. This paper presents the result of an experimental study to explore the relationship between maintainability and some software metrics. LCOM, RFC, DAC, and LOC are employed as metrics and time really spent for maintenance activity has been collected. In the experimental study, we have found that for some systems, the existing metrics may not be an indicator to maintenance effort, which is not consistent with our general knowledge on the relationship between them. Specifically speaking, we recognized that there should be more empirical study on the relationship between metrics and maintainability of softwares which have been developed using recent technologies such as software architecture and design pattern.

Key Words : Object-Oriented Metrics, Maintainability

1. 서 론

유지보수(Maintenance)란 소프트웨어가 제품으로 개발된 후 사용자의 사용이 시작하면서부터 폐기될 때까지 소프트웨어에 대해서 수행되는 모든 수정 활동을 뜻한다[10]. 사용자에게 전달된 소프트웨어는 다양한 이유로 수정의 필요성이 발생한다. 즉 사용자가 소프트웨어를 사용하는 중에 발견한 결함을 제거하거나, 운영체제 및 주변 장치 등과 같은 소프트웨어의 수행 환경의 변화를 반영하기 위해서, 사용자에 의한 요구사항의 추가/변경 등을 수용하기 위해서, 그리고 코드 리팩토링(Refactoring)과 같이 프로그램의 품질을 향상시키기 위한 모든 활동이 유지보수에 해당된다. 소프트웨어 유지보수가 전체 생명 주기(life cycle)에서 차지하는 비중이 매우 높은 것으로 알려져 있다. 따라서 효율적인 소프트웨어 유지보수 활동의 수행을 위한 많은 연구들이 진행되고 있다.

본 논문에서는 소프트웨어 메트릭(metric)을 바탕으로 한 유지보수 비용의 예측에 초점을 둔다. 즉 유지보수 활동에 사용될 수 있는 제한된 자원을 효율적으로 활용하기 위해서는 큰 비용의 유지보수가 요구될 수 있는 모듈을 미리 예측하고 이들 모듈을 재구성(restructuring)함으로써 유지보수성을 개선시킬 필요가 있다. 소프트웨어 메트릭은 소프트웨어의 품질을 평가하기 위하여 사용되는 척도로서 정량적으로 소프트웨어를 평가하고 이를 바탕으로 개선 활동을 하는 데

※ 이 논문은 교육인적자원부 지방연구중심대학육성사업 (차세대물류IT기술연구사업단)과 2004년도 부산대학교 교내학술연구비(신임교수 연구정착금)지원으로 이루어졌음.
† 준 회 원 : 한진중공업 정보시스템
†† 정 회 원 : 부산대학교 컴퓨터공학과 조교수
논문접수 : 2005년 2월 23일, 심사완료 : 2006년 1월 16일

핵심적인 역할을 한다. 또한 객체지향 개념이 도입된 이후로 많은 객체지향 관련 메트릭이 도입되었다. 그리고 일부 메트릭의 경우에는 메트릭과 오류의 발생 가능성과 생산성을 포함하여 유지보수 비용 등을 관련시키는 연구도 수행되었다.

그러나 기존에 수행된 유지보수성과 메트릭과의 관계에 관한 실험적 연구[3, 11]는 유지보수성과 메트릭간의 관계를 정확하게 나타내기에는 부족한 점이 있다. 단적인 예로서 기존의 실험에서는 유지보수성을 유지보수 과정에서 수정이 가해진 라인의 수로 측정하였다. 그러나 유지보수성은 유지보수 활동에 필요한 총체적인 비용을 포함해야 하며 단순히 소스 코드에서 변경된 라인 수만으로 소프트웨어의 유지보수성을 표현하는 것은 적절하지 않다. 다시 말하면 유지보수 비용은 단순히 코드를 추가/변경/삭제하는 시간뿐만 아니라, 코드 수정의 근본적인 원인 이해, 수정이 필요한 코드의 파악, 수정된 코드의 확인 등과 같은 비용도 포함되어야 한다.

본 논문에서는 기업에서 실제 사용되고 있는 응용 시스템의 유지보수 활동을 대상으로 유지보수 활동에서 실제로 소요된 시간을 바탕으로 객체지향 메트릭과 유지보수성과의 관계를 살펴본 결과를 소개한다. 즉 한 중공업 업체의 전산실에서 자재 구매 시스템을 대상으로 수행된 유지보수 활동을 대상으로 실제 유지보수에 소요된 시간을 측정하였다. 그리고 대상 시스템을 구성하는 각 클래스로부터 대표적인 객체지향 메트릭을 계산하였고 이들 메트릭과 수집된 유지보수 시간과의 상관관계를 분석함으로써 둘 간의 관련성을 파악하였다. 본 실험에 의하면 예상과 달리 기존의 메트릭이 유지보수 비용을 적절히 반영하지 못할 수 있음을 알았다.

본 논문의 구성은 다음과 같다. 2장에서는 대표적인 객체지향 메트릭을 소개하고 메트릭과 품질과의 관계를 파악하기 위하여 기존에 수행된 실험적인 연구를 소개한다. 3장에서는 본 논문에서 수행된 실험의 환경으로서 실험 대상 메트릭, 실험 대상 시스템, 유지보수 데이터 수집 방법과 메트릭 값 계산 방법을 설명한다. 4장에서는 실험 결과로서 측정된 메트릭 값과 수집된 유지보수 데이터에 대한 요약과 이들 간의 상관 관계를 설명한다. 마지막으로 5장에서는 결론과 앞으로 수행될 연구 방향에 대해서 설명한다.

2. 관련 연구

이 장에서는 소프트웨어 품질을 정량적으로 측정하기 위하여 사용되는 객체지향 메트릭에 대하여 간략히 설명을 한다. 그리고, 소프트웨어 품질과 메트릭과의 관계를 파악하기 위하여 기존에 수행된 연구 결과를 소개한다.

2.1 객체지향 메트릭

데이터를 구성하는 각 메트릭들은 소프트웨어의 복잡도를 보는 각각 다른 관점을 나타낸다. 따라서 이러한 메트릭은 전통적으로 코드의 라인 수(LOC, Line of Code) 및 제어 흐름(control flow)의 복잡도(조건 문이나 분기 문 같은 제어가 나뉘어지는)를 가지고 특정 모듈의 복잡도를 계산해왔다.

〈표 1〉 Chidamber & Kemerer 메트릭 스위트

메트릭	설 명
WMC(Weighted Methods Per Class)	WMC는 클래스 내에서 구현된 메소드의 수 또는 메소드의 복잡도(메소드 복잡도는 사이클로메틱 복잡도로 측정된다)의 합을 의미한다.
DIT(Depth of Inheritance)	DIT는 상속 트리에서 클래스 노드로부터 트리의 루트 노드까지의 경로에 존재하는 최대의 조상 클래스의 수이다.
NOC(Number of Children)	NOC는 계층 내에서 어떤 클래스의 하위에 있는 서브 클래스들의 개수이다.
CBO(Coupling Between Object class)	CBO는 어떤 한 클래스가 결합되는 다른 클래스들의 수이다. CBO는 클래스가 의존하는 비상속 관련 클래스 계층의 수로 계산함으로써 측정된다.
RFC(Response For a Class)	RFC는 클래스의 객체에 응답하는 메시지나 임의의 클래스 C가 호출하는 모든 메소드 집합의 수이다.
LCOM(Lack of Cohesion in Method)	LCOM은 응집도에 대한 척도로서 낮은 LCOM 값이 높은 응집도를 뜻한다. LCOM은 클래스의 메소드 쌍이 공유 인스턴스 변수를 가지는 지 그렇지 않은 지에 의해서 정의된다.

객체지향 개념이 소개되고, 객체지향 설계 및 구현이 일반화됨에 따라, 객체지향에 고유한 메트릭들이 제안되었고, 이러한 메트릭 스위트(suite) 중의 하나가 Chidamber & Kemerer[8]가 제안한 메트릭 스위트이다. Chidamber와 Kemerer는 대부분의 메트릭들이 이론적인 근거없이 제안되었음을 지적하고, Weyuker [15]가 제시한 6가지 기준을 통해 검증된 객체지향 메트릭을 제안하였다. Chidamber와 Kemerer가 제안한 객체지향 소프트웨어 메트릭은 각 개별 클래스 단위로 측정하게 되어 있는 6가지의 메트릭으로 구성된다. <표 1>은 Chidamber와 Kemerer가 제안한 6개의 메트릭을 요약한 것이다.

2.2 기존의 실험적 연구

그리고 이들 메트릭은 많은 실험 대상으로 사용되고 있다. <표 2>는 기존의 실험적 연구들이다. 표에서 알 수 있듯이, 기존에 수행된 실험적 연구에서는 C&K(Chidamber and Kemerer) 메트릭을 바탕으로 추가적인 몇몇 다른 메트릭을 척도로서 사용하였다. 그리고, 평가 대상 품질 요소로는 오류 발생 가능성, 생산성, 유지보수성 등이 있다.

<표 2>에서 볼 수 있듯이, 유지보수성과의 관계를 파악한 연구는 Li의 연구[11]와 Binkley의 연구[3]가 있다. Binkley의 연구에서는 CDM(Coupling Dependency Metric) [2]을 결합도 메트릭으로서 사용하였다. CDM은 동일한 연구자에 의해서 제안된 결합도 메트릭으로서 참조, 구조, 그리고 데이터 무결성(integrity) 측면의 의존성에 의해서 정의된다. Binkley의 연구에서는 유지보수 과정에서 추가, 변경, 삭제 되는 소스 코드의 행수와 수정된 클래스의 수를 유지보수 데이터로 가정하였다.

Li and Henry의 연구에서는 객체지향언어인 Ada로 구현된 두 가지의 상용 시스템을 연구에 이용하였다. 그리고 유닉스 환경의 LEX와 YACC를 이용하여 변경된 코드의 수를 분석하였고 메트릭과 유지보수성과의 관계를 알아보는데 이용하였다.

Li의 연구와 Blinkley의 연구는 모두 유지보수성과의 관계를 변화(입력, 변경, 삭제)된 코드의 수만으로 유지보수에 소요된 비용을 측정하였다. 그러나, 유지보수 비용은 단

〈표 2〉 기존의 실험적 연구

기존 연구	평가 대상 품질 요소	사용 메트릭
1993 Li and Henry[11]	유지보수성	C&K 메트릭
1996 Basili et al.[1]	오류 발생 가능성	C&K 메트릭
1998 Binkley and Schach[3]	유지보수성	CDM
1998 Chidamber et al.[7]	생산성	C&K 메트릭
1999 Briand et al.[5]	오류 발생 가능성	CBO, RFC, LCOM
1999 Tang et al.[14]	오류 발생 가능성	WMC, RFC
2000 Briand et al.[4]	오류 발생 가능성	C&K 메트릭
2000 Cartwright and Shepperd[6]	오류 밀도	DIT, NOC
2001b El Emam et al.[9]	오류 발생 가능성	C&K 메트릭

순히 프로그램 소스 코드를 입력/변경/삭제하는 것뿐만 아니라, 변경의 목적을 이해하고, 변경의 대상이 되는 부분을 파악하고, 어떻게 코드를 변경할 것인가를 결정하고, 변경된 코드가 올바르게 수행되는 지를 확인하는 모든 시간을 포함해야 한다. 이러한 측면에서 기존에 수행된 Li의 연구와 Blinkley의 연구는 정확하게 유지보수 비용을 수집한 것으로 보기 어렵다. 기존의 연구와 달리 본 논문에서는 실제 업무용 시스템을 유지보수할 때 소요된 전체적인 유지보수 시간을 수집하고 이를 바탕으로 메트릭과의 관계를 살펴본다.

3. 실험 수행 환경

이 장에서는 본 논문에서 수행된 실험의 환경을 소개한다. 먼저, 유지보수성과의 관련성을 파악하기 위하여 사용된 객체지향 메트릭을 소개한다. 그리고, 실험 대상이 된 기간계 시스템의 규모, 플랫폼(platform) 등의 특성을 설명한다. 마지막으로 유지보수 비용의 수집 방법과 메트릭 계산 방법을 소개한다.

3.1 대상 메트릭

본 논문에서는 위의 Chidamber와 Kemerer[8]의 6가지 Metrics중 RFC, LCOM을 사용하였고 Li의 연구[11]에서 제안된 DAC 메트릭, 그리고 LOC를 사용하여 실험을 하였다. 기존의 객체지향 메트릭 중에서 이와 같은 4개의 메트릭을 대상으로 선정한 이유는 다음과 같다.

- 메트릭은 크게 응집도(cohesion)[12], 결합도(coupling)[16], 그리고 규모(size)와 관련된 세가지 부류로 분류될 수 있다. 본 실험에서는 각 분류 별로 대표적인 메트릭을 선정하였다. 즉 응집도 메트릭으로서 C&K의 LCOM을 사용하였다. 그리고, 결합도 메트릭으로서는 C&K의 RFC와 Li가 제안한 DAC를 사용하였다. 프로그램의 규모에 관한 메트릭으로는 LOC를 선정하였다.
- 다음 장에서 소개되듯이 본 실험에서는 Java로 개발된 시스템뿐만 아니라 Delphi로 개발된 시스템도 포함하고 있다. Java 또는 C++ 언어의 경우에는 여러 객체지향 메트릭을 계산하는 도구들이 존재하지만 Delphi로 개발된 프로그램에 대한 메트릭 계산 도구를 확보하지 못하였다. 따라서 본 실험에서는 Delphi 코드에 대해서는 수작업으로 메트릭 값을 계산해야 했기 때문에 많은 메트릭을 선정하지 못하고 응집도, 결합도 그리고 규모의 범주 별로 대표적인 것을 선정하게 되었다.

본 실험에서 사용된 LCOM, RFC, DAC와 LOC에 대하여 간략히 정리하면 다음과 같다.

- LCOM (Lack of Cohesion in Methods)

LCOM는 C&K가 제안한 대표적인 응집도 메트릭으로서 다음과 같이 정의된다.

$$LCOM = |P| - |Q|, \quad |P| > |Q| \text{ 일 때}$$
$$= 0, \quad |P| <= |Q|$$

여기서 P는 클래스를 구성하는 메소드(method)들 사이에서 공통적으로 사용된 인스턴스 변수(instance variable)들이 있는 메소드 쌍의 수이고, Q는 인스턴스 변수를 공유하지 않는 메소드 쌍의 수이다. 정의에서 볼 수 있듯이 LCOM은 응집도에 대한 반대 척도이다. 즉 클래스의 응집도가 높을수록 LCOM은 낮은 값을 가지고 응집도가 낮으면 LCOM은 높은 값을 가진다. 또한 LCOM은 정규화(normalization)가 되지 않은 척도이다. 즉 LCOM의 최소값은 0(높은 응집도)으로 정의되어 있지만 최대값은 결정되어 있지 않다.

LCOM은 이론적인 측면에서는 비판을 받고 있지만 실제 실험에서는 가장 많이 사용되고 있는 척도이다. 응집도가 높을수록 단순하고 재사용성이 높음을 의미한다. 그리고, 낮은 응집도를 가진 클래스들은 응집도가 높아지는 두 개 혹은 그 이상의 클래스들로 쉽게 나뉘어 질 수 있다. 반면에 낮은 응집도나 응집도의 결여는 복잡도를 증가시켜서 개발 과정 동안에 오류의 양을 증가시킬 뿐만 아니라 프로그램에 대한 이해와 유지보수에 어려움을 주는 것으로 알려져 있다.

- RFC(Response for a Class)

RFC는 클래스의 객체에 응답하는 메시지나 임의의 클래스 C가 호출하는 모든 메소드의 수이다. 메시지를 통해 클래스에서 호출되는 메소드가 많아지면 많아질수록 해당 클래스는 다른 클래스와 복잡한 관계를 가진다. 따라서 RFC는 클래스 간의 의존성 정도를 나타내는 결합도 메트릭에 해당된다. 결합도가 클수록 오류의 발생 가능성도 높고 유지보수의 비용이 높은 것이 일반적이다. 즉 한 클래스로부터 호출되는 메소드의 수가 클수록 수정에 많은 노력이 소요될 수 있다.

• DAC(Data Abstraction Coupling)

DAC는 Li에 의해서 제안된 결합도 메트릭으로서 클래스 간의 데이터 추상화에 의한 클래스 간의 의존성을 의미한다. DAC는 클래스에서 사용되는 클래스의 수로서 정의된다. 즉 한 클래스와 집합(aggregation) 관계가 있는 클래스의 수를 뜻한다.

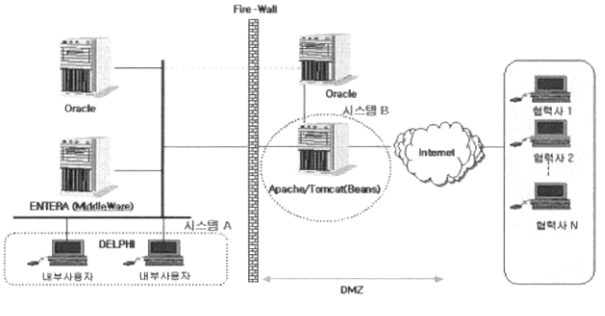
• LOC(Line of Code)

LOC는 프로그램을 구성하는 라인의 수로서 정의된다. 구체적으로 말하면 프로그램의 줄당 공백(blank)과 주석(comment)을 제외한 모든 라인의 수, 특히 프로그램 머리부, 선언부분, 비실행문을 모두 포함한다. LOC는 프로그래밍 언어의 선택, 프로그래머의 코딩 스타일 등에 따라 큰 차이를 가진다는 단점이 있지만 측정이 용이하고 직관적이기 때문에 전통적으로 프로그램 규모의 단위로서 사용되어 왔다.

3.2 대상 시스템

유지보수 비용에 대한 자료를 수집하기 위해서 사용된 시스템은 한 중공업 업체에서 구축되어 사용되고 있는 자재 구매 시스템이다. 자재 구매 시스템은 기업 내에서 각종 자재를 구매하기 위한 견적에서 입고까지의 일련의 공정을 지원하는 시스템이다. (그림 1)은 실험의 대상이 되는 자재 구매 시스템의 아키텍처를 간략히 보여준다.

그림에서 볼 수 있듯이 이 자재 구매 시스템은 클라이언트/서버 시스템으로 구성되어 있다. 그리고 기업 내부의 사용자를 위해서는 윈도 기반의 클라이언트를 제공하며 인터넷을 통한 사용자를 위해서는 웹 기반의 클라이언트를 제공한다. 편의상 전자를 시스템 A 그리고 후자를 시스템 B라고 부르겠다.



(그림 1) 실험 대상 시스템

3.2.1 시스템 A

시스템 A는 기업 내부의 사용자를 위하여 제공되는 시스템으로서 TP(Transaction Processing) 형태의 미들웨어와 데이터베이스로 구성된 전형적인 클라이언트/서버(client/server)로 구성되었다. 사용자 인터페이스는 Delphi(Object Pascal)로 구현되었고 미들웨어는 C언어 기반으로 ENTERA가 사용되었다. ENTERA에서는 C 프로그램이 존재하지만 이는 Oracle 데이터베이스에 대한 접근만을 전담하는 형태를 취하며 실제적인 비즈니스 로직은 Delphi 클라이언트에서 제공되고 있다.

〈표 3〉 시스템 A의 특성

	최소값	중간 값	최대값	평균값
메소드의 수	3	64.5	126	48.72
인스턴스 변수의 수	78	150	500	153.5

본 실험에서는 객체지향 메트릭과 유지보수성과의 관련성에 초점을 두므로 Delphi로 개발된 클라이언트 시스템만을 대상으로 한다. 클라이언트는 총 18개의 클래스로 구성되었다. <표 3>은 시스템 A를 구성하는 클래스들의 메소드 및 인스턴스 변수에 대한 요약이다.

자재 구매 시스템에 대한 클래스로서 18개는 그 수가 작다고 판단될 수가 있다. 그러나 시스템 A는 전형적인 4GL 프로그램으로서 사용자 인터페이스를 위한 컨트롤(control)들과 이벤트 프로시저(event procedure)들을 포함하여 많은 메소드와 인스턴스 변수들이 비즈니스 로직(business logic)을 구현하기 위해서 사용되고 있기 때문에 각 클래스의 규모(인스턴스 변수와 메소드의 크기)는 크다고 볼 수 있다. 참고로, (그림 2)는 시스템 A의 메소드와 인스턴스 변수의 분포를 나타낸다. 그림에서 볼 수 있듯이 90% 정도의 클래스가 100개 이상의 인스턴스 변수를 가지며 60% 정도의 클래스가 50개 이상의 메소드를 가지고 있다.

3.2.2 시스템 B

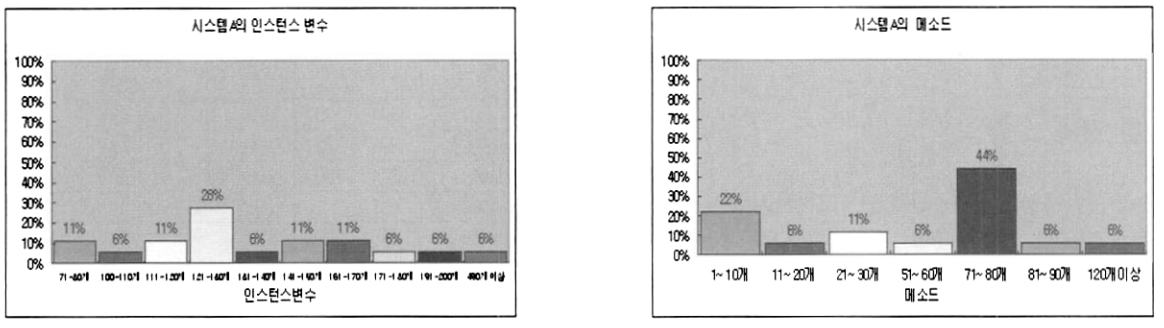
시스템 B는 기업 외부의 사용자를 위해서 웹 환경 기반으로 제공되는 시스템으로서 사용자 인터페이스의 구현은 JSP로, 비즈니스 로직과 유틸리티 형태의 기능을 제공하는 클래스들은 자바 빈으로 구현 되었다. 본 실험에서는 JSP를 제외한 총 33개의 Java 클래스를 대상으로 하였다.

(그림 3)은 시스템 B를 구성하는 33개의 클래스에 대한 메소드와 인스턴스 변수의 분포를 나타낸다. 시스템 B에서는 시스템 A와는 달리 비즈니스 프로세스(process)와 유틸리티(utility) 기능이 세분화되어 개별적인 클래스로 구현되었기 때문에 클래스의 수는 많다. 반면에 메소드와 인스턴스 변수의 수는 시스템A처럼 많지는 않다. 이는 시스템 A에서는 Delphi로 개발된 클래스가 비즈니스 로직 뿐만 아니라 복잡한 사용자 인터페이스를 클래스의 기능으로서 함께 제공하는 반면에 시스템 B에서는 사용자 인터페이스는 JSP로 개발되었고 실험 대상인 Java 클래스는 비즈니스 로직만을 담당하고 있기 때문이다.

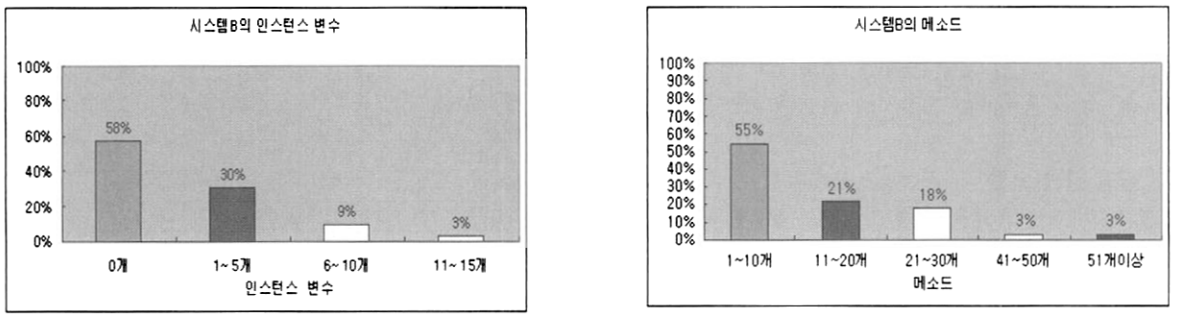
3.3 유지보수 데이터 수집 및 메트릭 계산

3.3.1 유지보수 데이터 수집

본 실험에서는 자재 구매 시스템을 대상으로 4명의 개발자가 3개월간 수행한 유지보수 활동에서 클래스 별 유지보수 시간을 수집하였다. 즉 본 실험을 위해서 인위적인 환경을 구성한 것이 아니라 실제 자재 구매 시스템을 운영하는 한 중공업 업체의 전산실에 근무하는 4명의 개발자가 3개월 동안 수행한 유지보수 활동을 그대로 데이터로서 수집하였다. 3개월 동안 수행된 유지보수 활동은 사용자에 의해서



(그림 2) 시스템 A의 클래스의 규모



(그림 3) 시스템 B의 인스턴스 변수와 메소드의 분포

요청된 요구사항의 변경(예를 들면, 기능의 추가 또는 사용자 인터페이스의 변경 요구)과 발견된 결함의 수정 등을 포함하며 운영체제와 주변장치 등의 변화에 따른 유지보수와 재구성 활동은 수행되지 않았다. 그리고 4명의 개발자는 실제로 자개 구매 시스템을 개발할 때 참여한 개발자로서 5년 내외 정도의 소프트웨어 개발 경력을 가지고 있다.

유지보수 데이터의 수집은 <표 4>의 형태로 수집되었다. 각 유지보수에 대한 요구를 수용하기 위하여 소요된 시간을 실제 수정이 가해진 클래스 별로 기술하였다. 즉 유지보수 기간 동안 유지보수 사항이 발생시 해당 클래스 명, 조치사항, 프로그램 수정에서 테스트 그리고 반영까지의 시간을 10분 단위로 기록하였다. 이 때 소요 시간은 단순히 소스 코드를 수정하는 데 소요된 시간만을 나타내는 것이 아니라 유지보수 활동에 소요된 총체적인 시간을 포함한다. 즉 유지보수의 발생 원인을 분석하고 이를 수용/반영하기 위한 방법을 결정하고 실제로 소스 코드를 수정한 후에 테스팅을 통하여 수정을 확인하는 모든 시간을 포함하고 있다.

4명의 개발자는 3개월 동안 총 112회 유지보수 활동을 수행하였다. 그리고 유지보수에 소요된 전체 시간은 4,540분이며 1회의 유지보수 시 평균 41분이 소요 되었다.

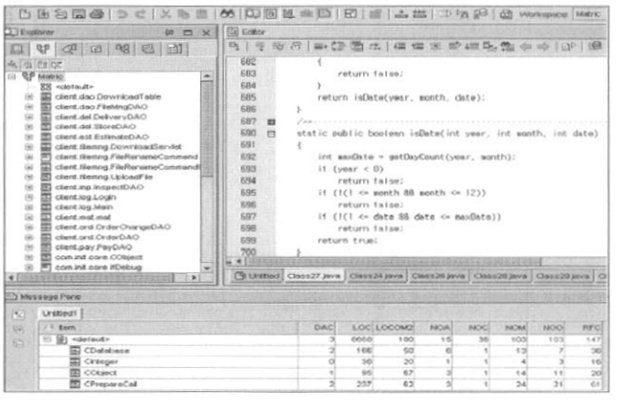
〈표 4〉 수집된 유지보수 데이터의 예

모듈	클래스 명	소요 시간 (분)	조치사항
견적정보	Maia0010	20	디테일 화면에 스크롤 불가
주문정보	Maia0050	30	원화 합계금액 표시되지 않음
입고정보	Maia0100	50	개별처리에 대한 수량, 금액 확인 필요
...	...	...	.....

3.2.2 메트릭 계산

Delphi로 구현된 시스템 A의 18개의 클래스로부터 메트릭을 계산할 때는 각 메트릭의 정의에 따라서 수작업으로 수행하였다. 기존에 Delphi 코드에 대해서는 메트릭의 자동적인 계산을 지원하는 도구를 확보하지 못했기 때문이다. 실제로 클래스의 수가 18개에 불과하지만 <표 3>과 (그림 2)에서 볼 수 있듯이 각 클래스의 규모가 크기 때문에 LCOM, DAC, RFC를 계산할 때 상당한 시간이 소요되었다.

반면에 시스템 B는 Java 언어로 개발되었기 때문에 메트릭 측정 도구를 이용하여 4개의 메트릭 값을 자동으로 추출하였다. 본 실험에서는 자바 언어에 대한 UML 모델링 도구인 Together를 이용하여 4개의 메트릭을 모두 계산하였다. (그림 4)는 Together를 이용하여 메트릭을 계산한 결과를 보여 준다. 그림에서 볼 수 있듯이 좌측 상단은 메트릭 측



(그림 4) Together를 이용한 메트릭 값 계산

정 클래스들, 우측 상단은 클래스의 코드가 보여지며 하단에는 클래스들에 대한 메트릭 값을 계산하여 보여준다.

4. 실험 결과

이 장에서는 수행된 실험 결과를 시스템 A와 시스템 B로 나누어서 설명한다. 우선 실험 결과 데이터를 소개하고 이를 바탕으로 유지보수성과 각 메트릭과의 관계에 대한 분석 결과를 설명한다.

4.1 시스템 A의 실험 결과

이 절에서는 시스템 A에 대한 실험 결과를 설명 한다. 먼저 계산된 각 메트릭 값과 수집된 유지보수 시간을 요약한 후에 메트릭과 유지보수 시간과의 관계를 살펴본다.

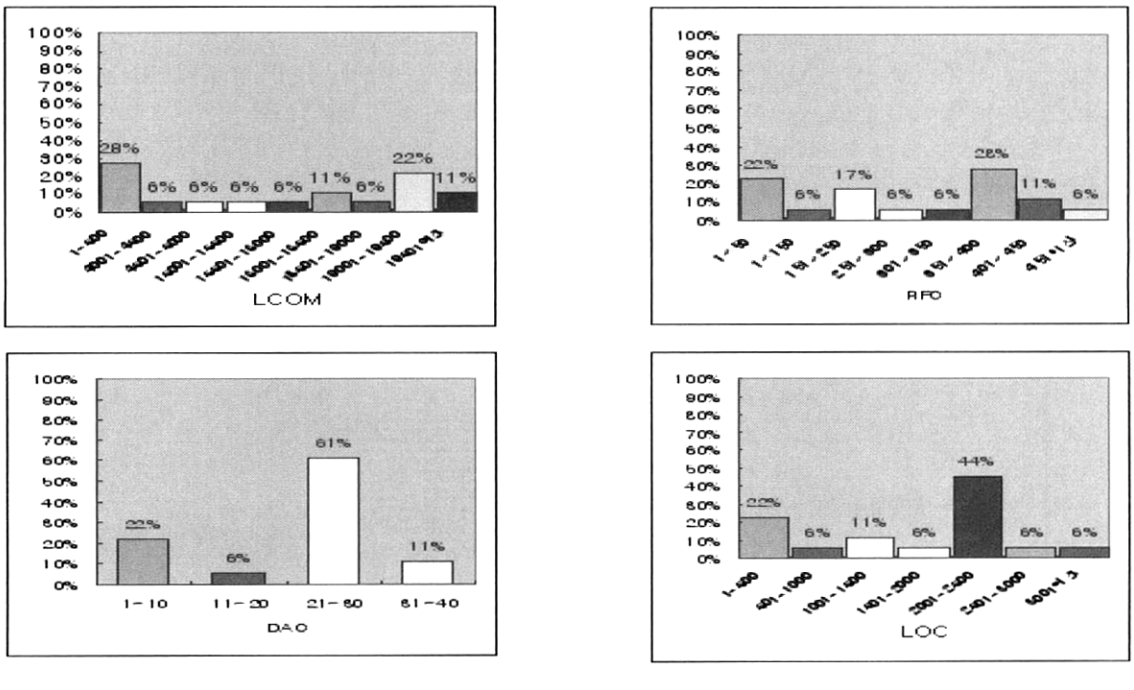
4.1.1 실험 결과 요약

<표 5>는 시스템 A의 각 클래스 별로 측정된 메트릭 값과 수집된 유지보수 시간을 보여 준다. 가장 좌측은 시스템 A를 구성하는 클래스 이름을 보여준다. 그리고 각 클래스별로 LCOM, RFC, LOC, DAC 값을 보여 준다. 총 유지보수 시간은 해당 클래스에 대해서 수행된 유지보수 시간의 총합이고 평균 유지보수 시간은 해당 클래스에 대해서 수행된 1회의 유지보수 시 소요된 평균 시간(즉 총 유지보수 시간/유지보수 횟수)을 뜻한다. 시스템 A의 경우 5개(maia009a, maia0061, maia0072, maib0162, maib0172)를 제외한 나머지 클래스에서 유지보수가 발생 하였다. 그리고 메트릭 값을 살펴보면 4개(maia009a, maia0092, maib0162, maib0172)는 200 라인 내외의 크기이지만 나머지 클래스는 대부분의 경우 500 라인 이상의 크기를 가지며 심지어 maia0051는 6500 라인 이상의 크기를 가지고 있다.

<표 5> 시스템 A에 대한 메트릭과 유지보수 결과

클래스 이름	LCOM	RFC	LOC	DAC	총 유지보수 시간(분)	평균 유지보수시간(분)
maia0011	4394	289	1368	31	750	53.57
maia0021	4892	232	1295	25	240	26.67
maia0051	15304	1602	6536	33	310	25.83
maia009a	1	3	228	6	0	0
maia0061	18762	402	2282	29	0	0
maia0071	16112	357	2125	23	240	43.33
maia0072	213	131	657	15	0	0
maia0081	16190	358	2100	25	70	35
maia0091	20508	225	2203	27	200	50
maia0092	6	4	179	9	60	60
maia0101	33960	446	2508	26	300	100
maia0111	19206	307	1799	26	60	60
maib0111	15670	243	2189	28	40	20
maib0161	19044	375	2110	26	70	70
maib0162	1	10	158	8	0	0
maib0171	19060	362	2110	26	110	36.67
maib0172	4	12	170	7	0	0
maib0211	19387	365	2078	25	60	60

(그림 5)는 시스템 A에서 계산된 4개 메트릭 값의 분포를 보여준다. LCOM의 경우에 500이하의 LCOM 값을 가지는 클래스가 전체 클래스 중 28%에 해당되고 나머지 72 %에 해당되는 클래스들은 4000이상의 높은 LCOM 값을 가진다. LCOM의 값이 높다는 것은 낮은 응집도를 의미한다. 즉



(그림 5) 시스템 A의 메트릭 값 분포

클래스의 메소드가 여러 가지 목적을 가지고 있다는 것을 의미한다. 코드를 분석한 결과 이는 유틸리티 성격의 기능을 구현하는 많은 수의 메소드를 가지고 있고 이들 메소드가 해당 클래스의 다른 메소드와 관련이 없기 때문에 LCOM의 값이 매우 큰 것으로 파악된다.

RFC의 경우에 300이상의 RFC 값을 가지는 클래스가 전체 클래스 중 56 %에 해당 된다. RFC 또한 매우 높은 값임을 알 수 있다. 즉, 대부분의 클래스가 다른 객체들과 매우 많은 수의 의존성(호출관계)를 가지고 있는 것으로 파악되었다. 이것은 결국 다른 클래스의 수정이 해당 클래스의 수정을 유발할 가능성이 매우 높은 상황이기 때문에 높은 유지보수의 비용을 초래할 것으로 예측된다..

DAC의 경우에 전체 클래스 중 72 %에 해당하는 클래스들이 21이상의 높은 값을 가진다. DAC가 높은 것은 각 클래스가 다른 많은 수의 클래스를 내부 인스턴스 변수로서 가지고 있음을 뜻한다. 코드를 분석할 결과 시스템 A의 경우에는 Delphi를 이용해서 비즈니스 로직뿐만 아니라 사용자 인터페이스를 제공하고 있으므로 사용자 인터페이스를 위한 많은 수의 클래스를 포함하고 있기 때문인 것으로 분석되었다.

LOC의 경우에 전체 클래스 중 56%에 해당하는 클래스들이 2000이상의 값을 가진다. 높은 LCOM 값과 연관을 시켜서 생각하면 이는 각 클래스가 모듈화가 충분히 되어 있지 않음을 의미한다. 즉 한 클래스가 너무나 많은 기능을, 게다가 서로 관련이 없는 여러 기능을 제공하고 있는 것으로 분석된다.

4.1.2 상관 관계 분석

본 실험에서는 각 메트릭 값과 유지보수 비용과의 관계를 파악하기 위해서 각 메트릭 값과 유지보수 평균시간과의 상관 계수를 계산하였다[17]. <표 6>에서는 클래스 별 평균 유지보수 시간과 각 메트릭과의 상관 계수 값을 보여준다.

〈표 6〉 시스템 A에 대한 상관 관계 분석

메트릭	LCOM	RFC	DAC	LOC
상관계수	0.64	0.16	0.43	0.24

각 메트릭과의 상관 계수는 양수의 값을 가진다. 이는 메트릭이 평균 유지보수 비용에 비례함을 뜻한다. 각 메트릭의 특성 별로 자세히 해석을 해 보면 LCOM의 값이 클수록 높은 유지보수 비용이 소요됨을 의미한다. LCOM은 응집도 메트릭이지만 높은 LCOM 값은 낮은 응집도를 뜻한다. 따라서 이 분석 결과는 낮은 응집도의 클래스가 많은 유지보수 시간이 소요되었음을 뜻한다. 이는 기존에 응집도가 높을수록 유지보수 비용이 낮을 것으로 기대되는 예상과 일치하고 있다. 그리고, RFC와 DAC의 상관 계수가 양수 인 것도 일반적인 예상과 일치한다. 즉 결합도 메트릭인 RFC와 DAC가 클수록 유지보수 비용이 많이 소요됨을 실험을 통해서 확인하였다. 마찬가지로 LOC는 클래스의 규모이므로 크기가 큰 클래스일수록 많은 유지보수 시간이 소요될 것이라는 예상과 일치하고 있다. 요약하면 시스템 A의 경우에는 응집도, 결합도, 규모 메트릭과 유지보수 비용과의 관계에 대한 일반적인 예측이 실제 시스템의 유지보수 활동에서 적용되고 있음을 확인하였다.

4.2 시스템 B의 실험 결과

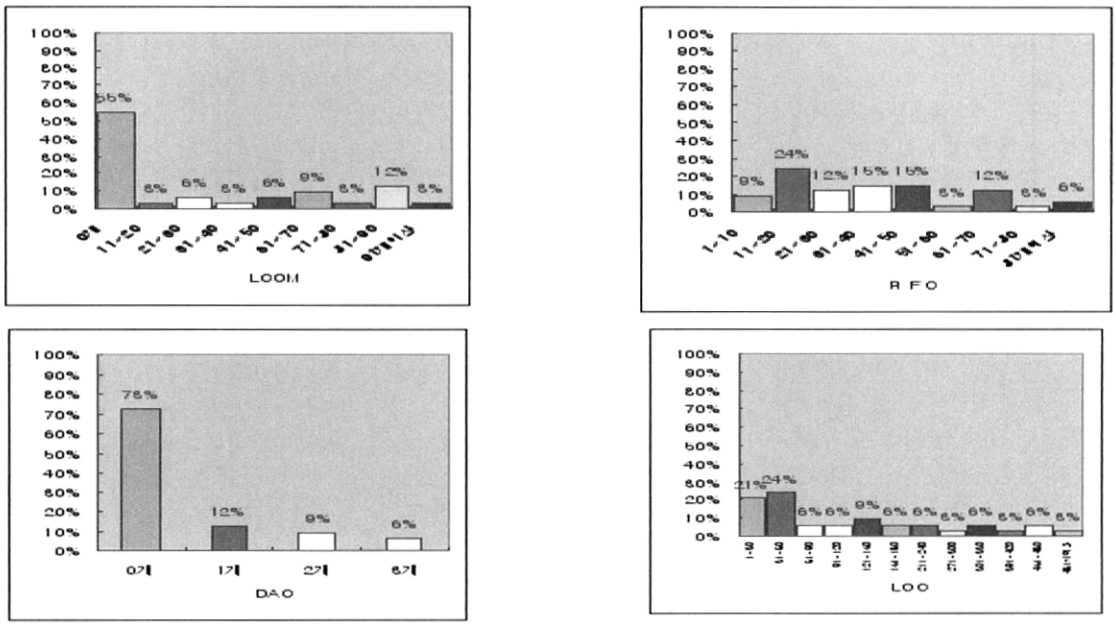
이 절에서는 시스템 B에 대한 실험 결과를 설명 한다. 먼저 측정된 메트릭 값과 수집된 유지보수 시간을 요약한 후에 메트릭과 유지보수 시간과의 관계를 살펴본다.

4.2.1 실험 결과 요약

<표 7>에서는 시스템 B를 구성하는 33개의 Java 클래스에 대하여 측정된 메트릭 값과 수집된 유지보수 시간을 보여 준다. 시스템 B의 경우에는 33개의 클래스 중에서 유지보수 활동 동안에 13개의 클래스에서만 실제 유지 보수가 발생하였다. 즉 시스템을 구성하는 클래스 전체에서 균일하게 수정이 발생하지 않고 일부 클래스에서만 유지보수 활동이 수행되었다. 코드에 대한 자세한 분석을 한 결과 이는 시스템 A와 달리 시스템 B에서는 전체 클래스 중에서 일부

〈표 7〉 시스템 B에 대한 메트릭과 유지보수 결과

클래스	LCOM	RFC	LOC	DAC	총 유지보수시간 (분)	평균 유지보수시간 (분)
CDatabase	50	36	166	2	20	20
CInteger	20	16	30	0	0	0
CObject	67	20	95	1	0	0
CPrepareCall	62	61	237	2	0	0
CQuery	72	62	319	3	0	0
CResultSet	84	94	614	2	0	0
CommonUtil	100	16	13	0	0	0
DataSet	0	41	63	0	0	0
DeliveryDAO	0	49	647	0	320	64
EstimateDAO	0	51	461	0	560	46.67
InspectDAO	0	48	512	0	80	26.67
Lib100	69	22	77	1	0	0
Lib200	0	147	2289	0	0	0
Lib300	29	65	277	1	0	0
Lib400	21	32	227	0	0	0
Lib410	0	30	214	0	0	0
Lib500	88	41	467	0	0	0
Login	0	14	30	0	60	30
DownloadServlet	0	20	59	0	190	63.33
DownloadTable	86	14	52	0	10	10
FileMngDAO	0	27	129	0	110	36.67
UploadFile	50	20	54	0	180	90
Main	0	17	92	0	110	18.33
MultipartRequest	81	64	411	3	0	0
OrderChangeDAO	0	34	105	0	60	30
OrderDAO	0	34	85	0	80	20
ParameterHelper	37	73	284	1	0	0
PayDAO	0	50	320	0	0	0
StoreDAO	0	37	164	0	120	30
ifDebug	0	1	5	0	0	0
ifError	0	6	10	0	0	0
ifLog	0	1	5	0	0	0
Mat	0	22	25	0	0	0



(그림 6) 시스템 B의 메트릭 값 분포

클래스만이 실제 비즈니스 로직을 가지고 있음에 기인하였다. 즉 대부분의 유지보수는 비즈니스 로직의 변경과 오류의 수정으로 발생을 한다. 시스템 A의 경우에는 대부분의 클래스가 비즈니스 로직을 제공하고 있으므로 전체 클래스에서 유지보수가 발생하였다. 그러나 시스템 B의 경우에는 일부 클래스(DeliveryDAO, EstimateDAO, InspectDAO 등)가 비즈니스 로직을 전담하며 다른 클래스들은 단순한 데이터 컨테이너(data container) 역할만을 수행하고 있기 때문에 비즈니스 로직을 가지고 있는 클래스에서 집중적으로 유지보수 활동이 수행된 것으로 분석된다.

(그림 6)에서는 시스템 B로부터 계산된 각 메트릭의 분포를 보여준다. LCOM의 값이 0을 가지는 클래스가 전체 클래스 중 55%에 해당 된다. 즉 시스템 B의 경우에는 대부분의 클래스가 높은 응집도를 가지는 것으로 해석되었다. 그러나 코드를 자세히 분석한 결과 이는 클래스의 메소드들이 공유하는 인스턴스 변수가 있기 때문이 아니라 메소드들이 인스턴스 변수를 사용하고 있지 않는 것에 기인하고 있는 것으로 분석되었다. 이는 응집도의 정의에 따르면 클래스의 각 메소드가 인스턴스 변수를 사용하지 않으면 낮은 응집도 즉 높은 LCOM 값을 갖도록 정의가 되어야 하지만 LCOM은 최고의 응집도 값 즉 0의 값을 갖도록 정의되고 있는 문제점을 가지고 있으며 이것이 실제 실험에서도 확인된 것이다.

RFC의 값은 고른 분포를 보여 주며 시스템 A에 비하여 비교적 그 값이 작다. 즉 각 클래스에서 메시지를 통해 클래스에서 호출되는 메소드의 수가 비교적 작다. 이는 사용자 인터페이스를 위한 복잡한 로직을 제공하는 시스템 A와 달리 시스템 B는 비즈니스 로직만을 제공하고 있으므로 비교적 적은 수의 메소드 호출만이 존재하는 것으로 분석된다.

DAC 또한 그 값들이 작은 것은 비교적 적은 수의 클래스와 집합 관계로 구현 되어 있음을 알 수 있다. 특히 80 % 정도의 클래스가 0 값을 가지는 것은 인스턴스 변수가 없음을 뜻한다. 즉 많은 클래스가 인스턴스 변수를 사용하지 않고 메소드만을 갖고 있으며 이는 LCOM의 값이 0인 클래스가 많은 것과 관련된다.

LOC의 경우에는 35%이하의 클래스만이 60 행 이하의 코드로 구성되었다. 그리고 Lib200 클래스를 제외하면 나머지 클래스들은 대부분 수백 라인 정도의 규모이다. 이 규모는 시스템 A에 비하여 매우 작은 규모의 클래스이다. 이는 시스템 A가 Delphi 클래스로서 복잡한 사용자 인터페이스까지 구현하고 있지만 시스템 B는 사용자 인터페이스는 JSP에서 담당하고 분석 대상인 Java 클래스는 비즈니스 로직만을 담당하고 있기 때문인 것으로 해석된다.

4.2.2 상관 관계 분석

시스템 A에서와 마찬가지로 시스템 B를 대상으로 각 메트릭과 유지보수 비용과의 관계를 파악하기 위해서 둘 간의 상관 계수를 계산하였다. <표 8>는 시스템 B의 전체 클래스에 대한 메트릭과 유지보수 시간과의 상관 관계를 나타낸다.

<표 8>의 결과를 보면 메트릭 별 상관계수의 값들이 예상과는 달리 음수임을 알 수 있다. 이는 유지보수 비용과 메트릭과의 관계가 상반되고 있음을 뜻한다. 즉, 응집도가 좋을수록, 결합도가 낮을수록, 클래스의 크기가 작을수록 유지보수 비용이 낮을 수 있음을 뜻한다. 이는 기존에 높은 응집도, 낮은 결합도, 작은 크기가 낮은 유지보수 비용을 의미한다는 기대와 상반된 결과를 보여 준다. 시스템 A에서는

〈표 8〉 시스템 B에 대한 상관관계 분석

메트릭	LCOM	RFC	LOC	DAC
평균시간상관계수	-0.27	-0.15	-0.06	-0.29

4개의 메트릭과 유지보수 비용과의 관계가 예상과 일치한 반면 시스템 B의 경우에 이와 같이 일반적인 예상과 다른 결과가 나온 것은 다음과 같이 분석될 수 있다.

<표 7>에서와 같이 시스템 B의 경우는 특정 클래스에 대해서만 집중적으로 유지보수가 발생했다. 예를 들어, Estimate DAO, DeliveryDAO 등의 클래스에서는 각각 560분, 320분의 유지보수가 발생했지만, CInteger, CObject, CPrepare Call, CQuery 등과 같은 클래스에서는 유지보수 기간 동안 전혀 유지보수가 발생하지 않았다. 즉, 유지보수가 실험 대상의 클래스에서 전체적으로 발생하지 않고 일부 클래스에서만 발생하였기 때문에 상관계수가 예상과 다른 값이 산출된 것으로 분석된다.

4.3 실험 결과 분석

본 절에서는 시스템 A와 시스템 B의 실험 결과를 분석한 내용을 소개한다. 우선 시스템 A는 메트릭과 유지보수 비용과의 관계가 예상과 일치하는 결과를 보였지만 시스템 B에서는 그 반대의 결과가 나온 것이 가장 주목할 만한 사항이다. 시스템 A와 시스템 B가 비교적 작은 규모이지만 실제로 한 중공업 업체의 전산실에서 개발되었으며 5년 정도 경력의 개발자 4명이 3개월 동안 실제 유지보수 활동을 수행하면서 수집한 데이터이므로 그 결과를 그대로 받아들일 가치가 있다고 생각한다.

본 논문에서는 두 가지 시스템에서 서로 상반된 결과가 산출한 원인은 근본적으로 시스템의 유형이 다르기 때문인 것에 기인한 것으로 추측한다. 즉 시스템 A의 경우에는 전형적인 클라이언트/서버 시스템으로서 사용자 인터페이스와 비즈니스 로직을 함께 포함하고 있는 클라이언트에 해당된다. 반면에 시스템 B의 경우에는 웹 기반 시스템으로서 사용자 인터페이스는 JSP를 이용한 웹 페이지에서 전담을 하며 대상이 되는 Java 클래스는 오직 비즈니스 로직만을 제공하고 있는 모습을 취하고 있다.

구체적으로 말하면 시스템 A에서는 시스템을 구성하는 대부분의 클래스가 유지보수 활동 동안에 수정이 발생하였다. 유지보수 활동 동안에 사용자 인터페이스의 변경 또는 비즈니스 로직의 변경이 발생하였다. 시스템 A를 구성하는 클래스는 매우 큰 규모로서 각 클래스가 두 가지 유형의 로직을 모두 가지고 있기 때문에 많은 클래스에서 변경이 발생하였다.

반면에 시스템 B는 실험 대상 클래스는 사용자 인터페이스 부분이 제외된 비즈니스 로직만을 포함하고 있다. 게다가 시스템 B에서는 실제 비즈니스 로직을 수행하는 부분과 필요한 데이터를 관리하는 부분을 별도의 클래스로 분리하여 개발되었다. 예를 들어 시스템 B에서 DAO로 이름이 주어진 클래스는 데이터베이스를 접근하여 비즈니스 로직을 구현하는 클래스이고 CQuery, CResultSet 등의 클래스는 DAO 유형의 클래스에서 데이터를 전달하거나 전달받을 때 사용하는 데이터 컨테이너 역할을 하는 클래스이다.

두 가지 유형의 시스템을 대상으로 수행된 실험에서 얻은 메트릭과 유지보수 비용과의 관계에 대한 결과를 요약하면 다음과 같다.

- 우선 본 실험에서는 기존의 실험과 달리 변경이 발생한 코드의 라인 수만을 측정한 것이 아니라 실제 유지보수 활동에서 수행되는 모든 작업에 소요되는 시간을 측정하였다. 따라서 비록 실험의 규모가 작지만 기존에 수행된 실험과 비교할 때 유지보수 활동에 소요되는 실제 비용을 바탕으로 메트릭과의 관련성을 파악하려고 한 것이 본 실험의 가장 큰 특징이라고 할 수 있다.
- 또한 본 실험에서 사용한 두 가지 유형의 시스템은 기존의 실험과 달리 비교적 최근의 시스템 아키텍처를 반영한다. 즉 시스템 A는 전형적인 클라이언트/서버 시스템의 클라이언트 부분에 해당되며 시스템 B는 웹 기반 시스템의 한 부분에 해당된다. 따라서 클라이언트/서버 시스템을 포함하여 최근의 웹 기반의 시스템 아키텍처를 반영하는 실험이라는 측면에서 의미가 있다고 생각한다.
- 시스템 A와 같이 클라이언트/서버 시스템에서는 대부분의 경우 클라이언트가 사용자 인터페이스뿐만 아니라 비즈니스 로직까지 가지고 있는 것이 일반적이다. 즉 클라이언트가 fat한 것이 특징이다. 또한 Delphi 또는 Visual Basic과 같은 4GL을 사용하는 경우 사용자 인터페이스 로직과 비즈니스 로직을 명백하게 별도의 클래스로 분리하지 않고 사용자 인터페이스 로직을 구현하고 있는 클래스에 비즈니스 로직이 혼재된 경우가 일반적이다. 본 실험은 이와 같은 경우에는 유지보수 비용과 메트릭과의 관계가 예상과 일치하고 있음을 확인하였다.
- 반면에 시스템 B에서는 비즈니스 로직만을 구현하고 있다. 게다가 실제 비즈니스 로직과 이 비즈니스 로직을 수행할 때 필요한 복잡한 데이터 구조를 별도의 클래스로 분리하여 개발되었다. 이와 같은 구조는 최근에 디자인 패턴 또는 리팩토링 등에 의해서 일반적으로 권장되는 설계 방식이다. 이로 인해서 시스템 B에서는 전체 클래스에서 균일하게 유지보수 활동이 수행되지 않고 실제 비즈니스 로직을 제공하는 일부 클래스에서만 유지보수 활동이 수행되었고 이것이 결국은 메트릭과 유지보수 비용과의 관계에 대해서 기존의 예상을 뒷받침하고 있지 않은 것으로 분석된 것이다. 그러나 시스템 B의 구성은 중소 규모의 전형적인 웹 기반 시스템이기도 하며 최근에 디자인 패턴과 리팩토링 등의 기법을 적용하는 것은 일반화되는 추세인 것에 비추어 볼 때 시스템 B로부터 얻은 결과 또한 나름대로 의미가 있다고 판단 된다. 즉 소프트웨어 아키텍처, 디자인 패턴, 리팩토링 기술과 같이 좋은 소프트웨어 설계를 얻기 위한 기법을 적용할수록 각 클래스의 역할이 명백하게 구분되기 때문에 클래스의 성격에 따라서 유지보수 활동이 일부 클래스에 집중적으로 수행될 수 있을 것이다. 따라서 유지보수 비용과 기존의 메트릭과의 관계에 대한 일반적인 예측이 일치하지 않을 가능성이 있게 된다.

요약하면 본 실험은 비록 한/두 시스템에 대해서 수행되었지만 최근의 웹 기반 시스템을 포함하여 다양한 설계 기법을 적용하여 개발된 시스템일 경우에는 메트릭과 유지보수 비용과의 관계에 대한 기존의 예측이 적용되지 않을 수

도 있음을 도출하였다.

5. 결론 및 향후 연구 방향

본 논문에서는 유지보수성과 메트릭과의 관계를 조사 해보기 위해 실제 운용되는 시스템을 대상으로 수행된 실험 결과를 소개하였다. 본 실험은 기존의 실험과 달리 두 가지 측면에서 기존에 수행된 실험과 다르다. 우선, 유지보수 비용을 단순히 입력/변경/삭제된 소스 코드의 행 수가 아니라, 실제 유지 보수 활동에서 소요된 시간으로 구하였다. 그리고, 최근에 일반적으로 구축되고 있는 웹 기반 시스템을 대상으로 수행되었다.

시스템 A의 경우에는 비교적 예상되는 결과가 산출되었지만, 시스템 B(웹 기반 시스템)의 경우에는 예상과 다른 결과를 얻었다. 이와 같이 시스템 B에서 예상과 다른 결과가 나온 것은 시스템 B의 경우에 클래스가 수행하는 역할 별로 근본적으로 복잡도의 차이가 있기 때문에, 유지보수가 균일하지 않게 일부 클래스에서만 발생하였기 때문으로 분석되었다. 또한 시스템 B와 같이 클래스에 특정 역할을 주는 설계 방식(예, 디자인 패턴)은 최근에 매우 일반화되고 있다. 따라서, 응집도, 결합도, 규모 등의 메트릭과 유지보수성과의 관계에 대한 기존의 일반적인 관점을 디자인 패턴 등을 적용하여 설계된 시스템에서는 적용되지 않을 수 있다고 생각한다. 물론, 이는 기존의 일반적인 주장과 반대되는 입장이므로 본 실험 하나만으로는 확인되기 어려울 것이다. 그러나, 본 실험은 메트릭과 유지보수성과의 기존의 관계가 소프트웨어 아키텍처, 설계 패턴 등과 같은 최근의 설계 기법을 적용하는 경우에는 유효하지 않을 수 있음을 시사한다. 따라서 이들 설계 기법이 적용된 시스템에서 메트릭을 통한 유지보수성을 평가/예측하기 위해서는 기존의 메트릭과의 관계로 새롭게 정립하거나 새로운 유용한 메트릭을 연구할 필요가 있다.

앞으로 본 논문에서 새롭게 제시된 객체지향 메트릭과 유지보수성과의 관계에 대해서 확인하기 위해서 보다 큰 규모가 있는 시스템에 유사한 실험을 지속적으로 수행할 필요가 있다. 또한 본 논문에서 사용된 4가지 유형의 메트릭을 포함하여 보다 다양한 메트릭을 이용하여 유지보수 비용과의 관련성을 파악해 보고자 한다.

참 고 문 헌

[1] 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.

[2] A. B. Binkley and S. R. Schach, "Toward a Unified Approach to Object-Oriented Coupling," Proc. 35th Annual ACM Southeast Conference, pp.91-97, 1997.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Trans. Software Eng., Vol.20, No.6, pp.476-493, 1994.

[9] 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.

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

[11] W. Li and S. Henry, "Object Oriented Metrics that Predict Maintainability," J. Systems and Software, Vol.23, pp.111-222, 1993.

[12] L. C. Briand, et. al, "A Unifed Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Engineering Journal, Vol.1, No.1, pp.65-117, 1998.

[13] R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Trans. Software Eng., Vol.29, pp.297-310, April, 2003

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

[15] E. Weyuker, "Evaluating software complexity measures," IEEE Trans. Software Eng., Vol.14, pp.1357-1365, 1988.

[16] L. C. Briand, et. al, "A Unified Framework for Coupling Measurement in Object Oriented Systems," IEEE Trans. Software Eng., Vol.25, No.1, pp.91-120, Jan./Feb., 1999.

[17] 김우철 외, 현대통계학, 영지문화사, 1988.

정 우 성

email : qvq@naver.com
2000년 동의대학교 컴퓨터공학과(학사)
2005년 부산대학교 산업대학원 전산학(공학석사)
2000년~현재 한진중공업 정보시스템팀 근무
관심분야 : 소프트웨어 유지보수, 소프트웨어 메트릭

채 흥 석

e-mail : hschae@pusan.ac.kr
1994년 서울대학교 원자핵공학과(학사)
1996년 KAIST 전산학과(공학석사)
2000년 KAIST 전산학과(공학박사)
2000년~2003년 동양시스템즈(주) 기술연구소 선임연구원
2003년~2004년 KAIST 전산학과 초빙교수
2004년~현재 부산대학교 컴퓨터공학과 조교수
관심분야 : 객체지향 모델링, 소프트웨어 테스팅, 분산 미들웨어, 소프트웨어 아키텍처