객체 지향 CASE 도구에 대한 재구조화 실험

조 장 우† · 김 태 균†

요 약

객체 지향 기법은 일반적으로 소프트웨어의 재사용성을 향상시킨다고 알려져 있다. 그러나 실제 객체 지향 소프트웨어를 재사용하기 위해서는 재구조화 과정이 필요하다는 사실이 점차 인식되고 있다. Refactoring은 객체 지향 소프트웨어의 재사용성과 유연성을 향상시키기 위하여 소프트웨어 시스템의 구조를 정제하는 과정이다. 본 논문에서는 이러한 Refactoring 기법을 기존에 구현된 바 있는 객체 지향 CASE(Computer Aided Software Engineering) 도구인 OODesigner 버전 1.x에 적용한 연구 결과를 제시한다. 버전 1.x는 Rumbaugh의 OMT(Object Modeling Technique) 모델 중에서 객체 모델을 지원하며, 객체 도표 작성, 클래스 자원에 대한 문서화, 자동적인 C++ 코드 생성, 클래스 재사용을 위한 정보 저장소, C++ 코드의 역공학 기능들을 지원한다. 비록 버전 1.x가 요구되는 기능과 신뢰성에 문제가 없었으나 구조적인 관점에서 문제를 가지고 있었기 때문에 새로운 기능을 추가하고 유지 보수하는데 많은 어려움이 있었다. 그러므로 본 연구에서는 기존의 버전 1.x를 재구조화하여 보수 유지가 용이한 시스템 구조로 만들었다. 본 논문에서는 재구조화의 필요성, 재구조화 과정, 재구조화 내용 그리고 재구조화 효과에 대해 기술하고, 기존의 OODesigner 버전 1.x와 재구조화된 OODesigner 버전 2.x를 소프트웨어 메트릭(metric)를 통해 비교 분석한 결과와 이 과정에서 얻은 경험들을 제시한다.

An Experimant in Refactoring an Object-Oriented CASE Tool

Jang-Wu Jo† · Tae-Guun Kim†

ABSTRACT

Object-oriented programming is often touted as promoting software reuse. However it is recognized that objected-oriented software often need to be restructured before it can be reused. Refactoring is the process that changes the software structure to make it more reusable, easier to maintain and easire to be enhanced with new functonalities. This paper describes experience gained and lessons learned from restructuring OODesigner, a Computer Aided Software Engineering(CASE) tool that supports Object Modeling Technique(OMT). This tool supports a wide range of features such as constructing object modeler of OMT, managing information repository, documenting class resources, automatically generating C++ and Java code, reverse engineering of C++ and Java code, searching and reusing classes in the corresponding repository and collecting metrics data. Although the version 1.x was developed using OMT(i.e. the tool has been designed using OMT) and C++, we recognized that the potential maintenance problem originated from the ill-designed class architecture. Thus this version was totally restructured, resulting in a new version that is easier to maintain than the old version. In this paper, we briefly describe its restructuring process, emphasizing the fact that the Refactoring of the tool is conducted using the tool itself. Then we discuss lessons learned from these processes and we exhibit some comparative measurements of the developed version.

※ 본 연구는 1996년도 한국과학재단 연구비지원(과제번호 : 961-0906-038-2)에 의한 결과임.

† 정 회 원 : 부산외국어대학교 컴퓨터공학과 교수

논문접수 : 1998년 12월 7일, 심사완료 : 1999년 2월 11일

1. 서 론

객체 지향 기법이 구조적인 방법에 비해서 문제를 풀기 위한 자연스러운 방법을 제공해 준다는 사실은 과거 십여 년간의 연구결과로 알 수 있다[3,6,7,13,17]. 객체 지향 기법은 추상화, 정보은닉, 모듈화, 계층 구조화, 지속성, 동적 바인딩, 다형성과 같은 개념들을 지원하여 소프트웨어의 재사용과 유지 보수가 용이하다[1]. 그러나 객체 지향 기법을 이용하여 개발된 소프트웨어를 좀 더 재사용하기 쉽고 유연성이 있게 하기 위하여 개발 중인 소프트웨어에 대한 반복적인 설계와 구현이 필요하다는 사실이 점차 인식되고 있다[15,16]. Refactoring이란 이러한 반복적인 과정을 위하여 매우 유용한 기법으로, 기존의 소프트웨어를 이해하기 쉽고 보수 유지가 용이하게 하며, 새로운 기능을 쉽게 추가할 수 있도록 소프트웨어의 구조를 바꾸는 것을 목적으로 한다[15,16]. 본 논문에서는 이러한 Refactoring 기법을 적용한 예로서 기존에 구현된 바 있는 객체지향 CASE 도구인 OODesigner 1.x[8,9,10]를 재구조화한 연구 결과를 제시한다.

기존의 버전 1.x는 Rumbaugh의 OMT[18]를 지원하는 환경을 제공하기 위하여 1993년부터 3년에 걸쳐 개발되었다. OODesigner는 1996년에 GNU의 Public Domain에 공개되어 세계적으로 6,000여 곳에서 교육용과 개발용으로 사용되고 있다. 현재 OODesigner는 ftp://203.230.73.24/pub/OOD 사이트로부터 다운 받을 수 있으며, 미국의 재사용 소프트웨어 기관인 ASSET(Asset Source for Software Engineering Technology)에 등록되어 있다.

버전 1.x를 개발하기 위한 목표는 제품에 대한 목표(product goal)와 개발 공정에 대한 목표(process goal)의 두 가지 관점으로 설정되었다. 제품에 대한 목표는 OODesigner의 기능에 관한 요구사항에 해당된다.

- 일반적인 그래픽 편집 기능
- OMT의 세 가지 모델을 지원하는 그래픽 도구
- 클래스 자원의 문서화 기능
- 설계된 객체 모델을 관리하기 위한 정보 저장소
- 설계된 문서로부터 C++ 코드의 자동 생성 기능
- C++ 코드로부터 클래스 도표를 작성하는 역공학 도구
- 클래스 재사용을 위한 클래스 저장 및 추출 기능
- C++ 코드로부터 메트릭(metric) 데이터 수집 기능

아울러 개발 공정에 대한 목표는 다음과 같다.

- 객체 지향 설계와 구현을 수행하기 위한 요원의 능력 향상
- 객체 지향 개발의 경계 없고(seamless) 반복적인 공정을 숙달
- OODesigner를 차기 버전 개발을 위한 CASE 도구로 사용
- 다른 기종으로의 이식과 시스템 기능의 추가를 용이하도록 보수 유지성을 확보

1996년에 버전 1.x가 완성되었으며, 제품에 대한 목표의 대부분을 만족하였다. 버전 1.x는 OMT 기법을 사용해서 설계되었고, 약 6만 라인의 C++ 코드로 구현되었다. 그러나 버전 1.x는 개발 공정에 대한 목표 중에서 유지 보수성과 관련된 목표를 만족시키지 못하였다. 즉, 버전 1.x는 주어진 기능을 완벽하게 수행하지만 부적절한 클래스 구조로 인해 새로운 기능을 추가하는 것이 어려운 것으로 인식되었다. 따라서 버전 1.x에 대해 Refactoring 기법을 적용하여 재구조화를 수행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 버전 1.x의 문제점과 재구조화 목표, 공정, 내용, 그리고 결과를 기술하였다. 그리고 3장에서는 버전 1.x와 재구조화된 버전 2.x를 소프트웨어 메트릭(metric)을 통해 비교 분석한 결과를 기술하였다. 마지막으로 4장에서 결론을 기술하였다.

2. OODesigner의 재구조화

이 장에서는 버전 1.x에 대한 재구조화의 필요성, 재구조화 과정, 재구조화 내용 그리고 재구조화 결과 얻어진 효과를 기술한다.

2.1 문제점 및 목표

버전 1.x의 설계 구조의 문제점은 버전 1.x의 구현 후반기에 파악되었다. 잘못된 구조로 인하여 새로운 기능의 추가가 점차 어려워졌으며, 코드의 중복으로 인한 보수 유지 문제가 심각하게 인식되기 시작하였다. 특히 보수 유지와 관련된 가장 큰 문제는 동적 모델러와 기능적 모델러의 추가 개발에 관한 것이다. 이들 동적 모델러와 기능적 모델러 기능을 추가하기 위한 방법으로는 1) 잘못된 구조를 가지고 있는 버전 1.x 상에다 추가 개발하는 것, 2) 버전 1.x를 재구조화한 후에 추가 개발하는 것의 두 가지 방법이 있었다. 두

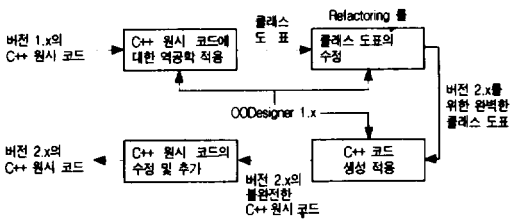
가지 방법에 따른 장단점을 면밀히 분석한 이후에 2)의 방법을 선택해서 재구조화를 시작하였다. 시스템 구조에 존재하는 문제점을 인식하며 얻은 교훈과 버전 1.x를 개발하는 과정에서 습득한 객체 지향 기법의 특성들을 기반으로 다음과 같은 재구조화 목표를 설정하였다. 결국 이러한 목표들은 시스템을 유지 보수하기 쉬운 구조로 만드는 것이다.

- 최대한 응용분야로부터 독립적인 클래스의 추출
- 상속 트리 재조직을 통한 코드 분량의 감소
- 클래스간의 적절한 결합도와 응집력 유지
- 기계 의존성을 지역화(localization)
- 코드 중복의 최소화
- 전역 변수와 전역 함수의 최소화
- 코드 신뢰성 향상

2.2 재구조화 과정

버전 1.x의 재구조화는 12개월 동안 진행되었다. 재구조화는 다음과 같은 절차로 이루어졌으며, (그림 1)은 SADT 표기법을[14] 이용하여 재구조화 과정을 도식화한 것이다. 이 과정에서 버전 1.x의 C++ 역공학 기능과 C++ 코드 생성 기능을 사용하였다.

1) 버전 1.x를 이용하여 버전 1.x의 소스 코드에 대한 역공학을 수행하였다. 이 단계에서 버전 1.x의 클래스 다이어그램을 생성하였다.
2) 버전 1.x의 클래스 다이어그램 편집 기능을 사용하여 단계 1)에서 생성된 클래스 자원을 재구조화하였다.
3) 버전 1.x의 코드생성 기능을 사용하여 단계 2)에서 생성된 클래스 자원에 대한 C++ 코드를 자동 생성하였다.
4) 생성된 C++ 코드를 대상으로 수정 및 디버깅을 수행하였다.



(그림 1) 재구조화 공정
(Fig. 1) The Refactoring Process

버전 1.x를 재구조화하는 과정에서 OODesigner를 CASE 도구로 사용함으로써 재구조화 과정은 매우 생산성 있게 진행되었다. 본 실험에서 CASE 도구를 적용함으로써 얻은 경험은 또 하나의 중요한 소득이다. OODesigner와 같은 객체 지향 CASE 도구는 시스템 엔지니어가 수행해야 하는 사소하고 지루한 작업을 감소시켜 주기 때문에 객체 지향 과제의 성공적인 수행을 위해서는 CASE 도구의 이용이 필수적이라고 믿어진다.

2.3 재구조화 내용

버전 1.x에 대한 재구조화 내용을 항목별로 나열하면 다음과 같다.

- 중복된 데이터 멤버와 코드의 삭제 : 중복된 데이터 멤버들을 상위 클래스에 정의하고 중복된 코드를 가지는 메소드들은 상위클래스에 새로운 메소드로 정의하여 중복을 삭제하였다. 재구조화 과정에서 코드 중복의 삭제는 가장 중요한 작업이었다. 그 이유는 재구조화 이후에 원시 코드의 수정이 쉽게 이루어질 수 있었기 때문이다. 수정의 용이성은 보수 유지의 용이성을 의미한다.
- 그래픽 사용자 인터페이스 요소의 캡슐화 : 버전 1.x에서는 그래픽 사용자 인터페이스의 구현을 위하여 OSF/Motif에서 제공되는 Widget들과 콜백 함수를 사용하였다. 그러나 이들 자료 구조와 콜백 함수가 객체화되지 않음으로 재사용할 수가 없었다. 그러므로 재구조화 과정에서 모든 GUI 요소들을 캡슐화하여 클래스로 만듦으로서 재사용성을 증가시켰다.
- 동적 바인딩 사용의 극대화 : C++ 언어는 동적 바인딩과 정적 바인딩 방식을 지원하는데, 일반적으로 초보자들은 정적 바인딩을 선호하는 경향이 있다. 버전 1.x에서도 정적 바인딩과 동적 바인딩을 혼용하였는데, 정적 바인딩을 동적 바인딩으로 변환하였다. 이러한 방법은 소프트웨어의 유연성을 향상시키고 제어 구조를 단순화한다.
- 전역 변수와 전역 함수의 캡슐화 : 시스템에 존재하는 전역 변수와 전역 함수는 프로그램의 이해를 어렵게 하고 재사용성을 감소시킨다. 따라서 이들 전역 변수와 전역 함수들을 관련 있는 클래스의 멤버로 포함시키거나 새로운 클래스를 만들어 캡슐화하였다. 그리고 플랫폼에 종속적인 라이브러리들도 캡슐화 하여 시스템의 기기 독립성을 향상시켰다.

■ 클래스 소멸자의 완벽한 구현 : Dangling Reference를 제거하기 위하여 클래스 소멸자(destructor)를 완벽하게 구현하였다. 이러한 방법은 시스템의 신뢰성을 향상시키고 디버깅을 쉽게 하였다.

■ 코딩 관습의 채택 : 시스템의 구현을 위한 나름대로의 코딩 관습을 정하였다. C++ 언어는 하나의 기능을 구현하는데 다양한 문법적 구조들을 제공하지만, 이러한 다양한 문법 구조는 혼동의 여지를 제공한다. 그러므로 C++ 언어의 다양한 선택들 중에서 다음과 같은 관습을 사용하였다.

- 매개 변수 전달에는 포인터 형의 변수만을 사용하고 레퍼런스 형은 사용하지 않는다.
- 상속 기능만을 사용하며 템플리트는 사용하지 않는다.
- 데이터 멤버는 항상 Private나 Protected 가시성을 가지도록 함으로서 데이터 멤버에 대한 접근은 항상 멤버 함수를 통해서만 이루어지도록 한다.
- 함수 포인터 기능은 사용하지 않는다.

재구조화의 결과로 155개의 클래스로 구성된 5개의 모듈을 작성하였다. 그 중의 한 모듈로서 (그림 2)와 (그림 3)은 재구조화 과정에서 이루어진 시스템의 구조 변화를 보여준다. 이 객체 모델들은 OMT 표기법에 관련된 객체들을 모델링한 것인데, (그림 2)와 (그림 3)은 각각 버전 1.x와 버전 2.x의 객체 모델이다.

2.4 재구조화 효과

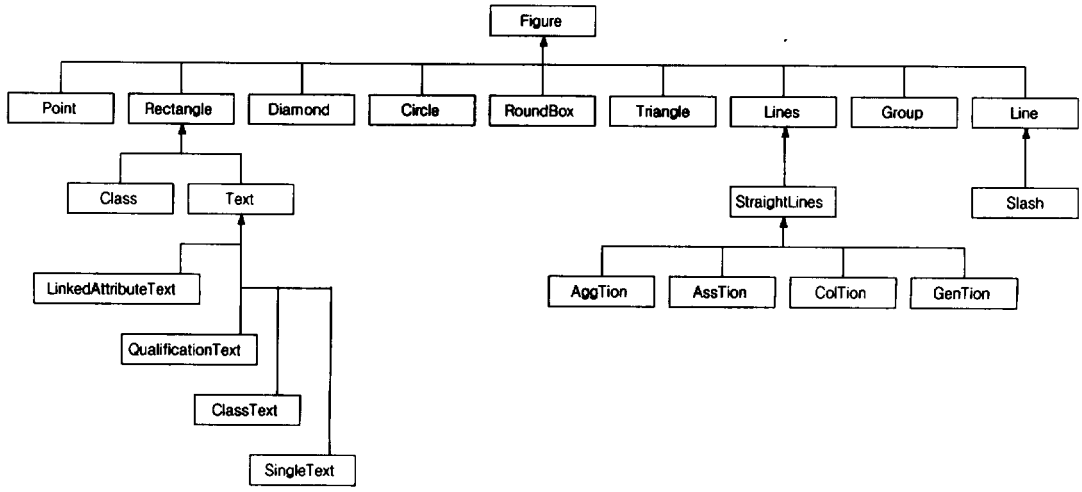
OODesigner 버전 1.x를 재구조화한 결과인 OODesigner 버전 2.x는 다음과 같은 특성을 갖게 되었다.

■ 기능 추가가 용이해졌다. 버전 1.x에 동적 모델러와 기능적 모델러를 추가 개발하기 위하여 약 12 MM (Man/Month)의 노력이 소요될 것으로 예상했다. 그러나 버전 2.x에서는 대부분의 GUI 객체들을 재사용할 수 있게 됨으로서 두 가지 모델러를 약 1 MM의 노력으로 개발할 수 있었다. 또한 Java 언어를 위한 역공학 기능과 코드 생성 기능의 추가도 약 2 MM의 노력으로 이루어졌으므로 새로운 기능의 추가가 매우 용이해졌음을 알 수 있다.

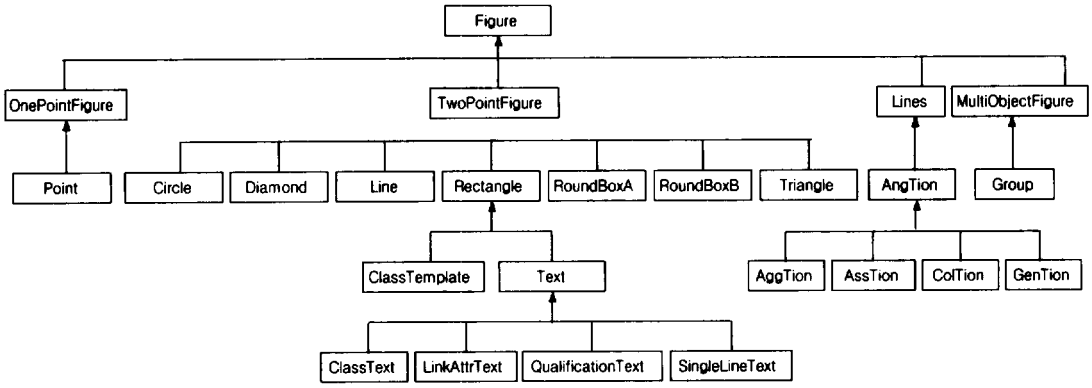
■ 수정이 용이해졌다. 버전 2.x가 대해 기기 독립성과 효율성을 증대시키기 위하여 데이터 저장 양식을 여러 번 변경하였다. 이와 같은 작업이 대개의 경우 짧은 시간에 이루어질 수 있었다.

■ 이해가 쉬워졌다. 소프트웨어의 개발은 때때로 장기간 정지될 때가 있다. 예를 들어 OODesigner의 개발을 6개월 동안 멈춘 경우도 있었는데, 다시 개발을 재개하는 경우에도 곧 바로 추가 구현을 시작할 수 있었다.

■ 다른 플랫폼으로의 이식이 용이해졌다. OODesigner의 재구조화 이후에 PC 버전을 Visual C++을 이용하여 개발하고 있다. 1 MM의 노력으로 20,000라인 분량의 53개의 클래스를 구현할 수 있었다. 또한 Java 버전도 구현 중인데, 1 MM의 노력으로 20,000



(그림 2) 버전 1.x에서 OMT 표기법에 관련된 객체 모델
(Fig. 2) (Partial) Object Model for OMT Notation in Version 1.x



(그림 3) 버전 2.x에서 OMT 표기법에 관련된 객체 모델
(Fig. 3) (Partial) Object Model for OMT Notation in Version 2.x

라인 분량의 57개의 클래스를 구현할 수 있었다. 물론 이와 같은 작업을 처음부터 수작업으로 하는 게 아니라 OODesigner의 역공학 기능과 코드 생성 기능을 이용한 변환 작업을 통하여 수행되었다.

- 유연성이 증가하였다. 때때로 특정 클래스의 구현부뿐만 아니라 명세부도 변경할 필요가 있었다. 그런데 이러한 변경의 영향이 매우 지역적, 즉 변경이 일어난 곳 주위에 한정될 수 있었다. 이러한 점은 버전 2.x에서는 동적 바인딩을 많이 사용되었기 때문에 소프트웨어 객체들이 느슨하게 결합된(loosely coupled) 점에 기인한다.
- 신뢰성과 안정성이 증대되었다. 버전 1.x에서는 에러를 수정하는데 대개의 경우 몇 일의 시간이 소비되었다. 그런데 버전 2.x를 개발하면서 발견된 에러들은 대개의 경우 몇 시간 내에 수정될 수 있었다.

앞에서 열거한 재구조화의 효과로 볼 때 재구조화된 버전 2.x는 보수 유지가 매우 용이해졌음을 알 수 있다.

3 재구조화 메트릭과 교훈들

3.1 재구조화 메트릭

본 장에서는 OODesigner 버전 1.x와 버전 2.x 간의 메트릭 데이터를 비교 분석한다. 소프트웨어를 평가하기 위해서 메트릭 데이터를 수집하는 일은 매우 중요하다. 본 연구에서는 OODesigner의 메트릭 수집 기능을 이용하여 이러한 데이터를 수집하였다. 본 연구에서 사용한 메트릭은 Chidamber와 Kemerer(C&K로 표현함)[2]의 객체 지향 메트릭 집합 중에서 WMC(Weighted Methods per Class), DIT(Depth of Inheritance Tree), NOC(Number of Children), CBO(Coupling between Object Classes) 항목들과 상식적인 선에서 수집된 항목들로 구성된다.

<표 1>의 메트릭 데이터는 잘못 설계된 객체 지향 시스템(버전 1.x)과 잘 설계된 객체 지향 시스템(버전 2.x)의 차이점을 파악하는데 유용할 것으로 보인다. 이 표에서 절대 값보다는 상대적인 값의 차이가 의미가 있으며, 이는 바람직한 객체 지향 설계 및 구현을 위한 좋은 지침이 되리라고 판단된다. 예를 들어, <표 1>의 07 항목으로부터 "클래스의 크기가 작게 설계될수록 바람직하다"는 추론을 할 수 있다.

〈표 1〉 버전 1.x와 2.x의 메트릭 비교
〈Table 1〉 Metric Comparison between version 1.x and 2.x

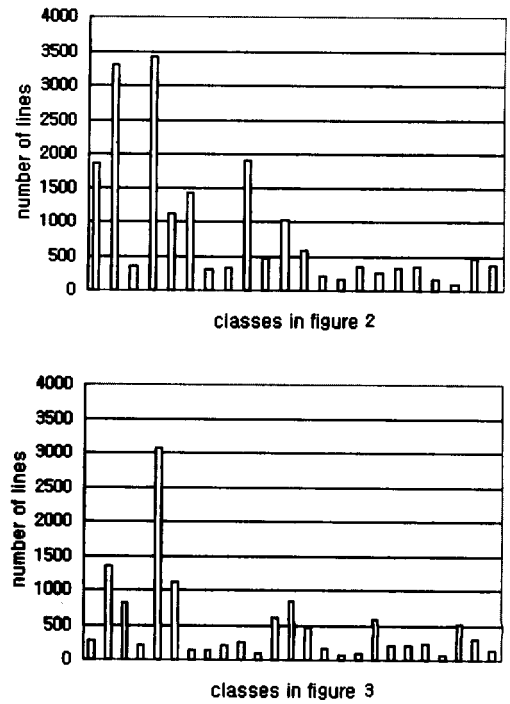
메트릭스	버전 1.x	버전 2.x	증감비율
01. 시스템의 라인 수	58925	59662	+1%
02. 시스템의 실행문 수	33245	30535	-8%
03. 시스템의 조건문 수	6382	5622	-12%
04. 시스템의 루프 수	1261	1091	-13%
05. 시스템의 클래스 수	97	155	+60%
06. 클래스의 전체 실행문 수	25692	25916	+1%
07. 클래스의 평균 실행문 수	265	167	-37%
08. 클래스의 전체 유일한 단어 수	2635	3561	+35%
09. 클래스의 평균 유일한 단어 수	27	23	-15%
10. 클래스의 전체 데이터 멤버 수	795	1030	+30%
11. 클래스의 평균 데이터 멤버 수	8.2	6.6	-20%
12. 클래스의 전체 메소드 수	918	1430	+56%
13. 클래스의 평균 메소드 수	18	15	-17%
14. 메소드의 평균 실행문 수	14	11	-21%
15. 최대 상속 깊이	3	4	+33%
16. 상속 깊이 전체 누적 수	134	233	+74%
17. 평균 상속 깊이	1.38	1.50	+9%
18. 최대 하위 클래스 수	19	29	+53%
19. 전체 하위 클래스 수	76	126	+66%
20. 평균 하위 클래스 수	0.78	0.81	+4%
21. 객체간의 전체 커플링 수	526	1002	+90%
22. 객체간의 평균 커플링 수	5.4	6.5	+20%
23. Law of Demeter 위반 횟수	485	891	+83%

<표 1>의 05 항목을 보면 버전 2.x 의 클래스 수가 많이 증가했다. 이는 버전 1.x의 GUI 관련 함수와 자료 구조가 클래스로 캡슐화 되었기 때문이므로, <표 1>의 06, 08, 10, 12 항목들은 의미 있는 데이터로 해석될 수 없다.

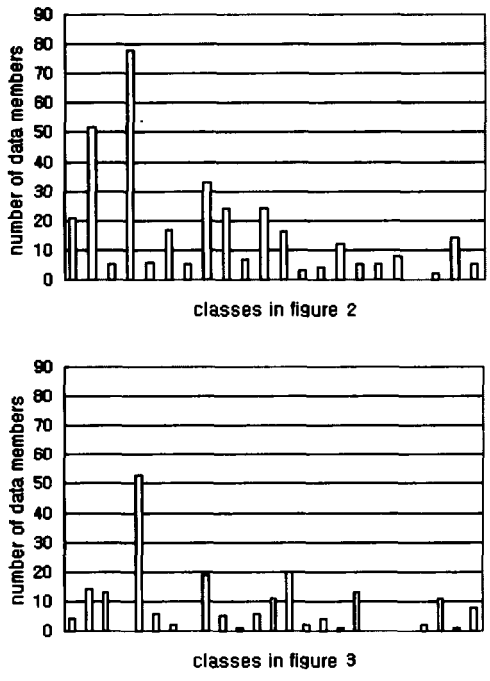
<표 1>의 개략적인 해석은 다음과 같다.

- [01, 02 항목] 전체 시스템의 크기가 변화하지 않았다. 이는 전역 변수와 전역 함수의 삭제가 시스템의 크기를 줄이도록 한 반면 새로운 클래스의 도입이 시스템의 크기를 늘이도록 한데 기인한다.
- [03, 04 항목] 시스템의 복잡도가 줄었음을 보여 준다. 이는 코드 중복을 최소화하고 동적 바인딩의 사용을 최대화한데 기인한다.
- [07, 09 항목] 개별 클래스의 크기가 작아졌다. 이 결과는 객체 지향의 초보자에게 좋은 지침이 될 수 있다. 비록 클래스의 최적 크기는 알 수 없지만 클래스의 크기를 작게 하면 클래스의 재사용성은 증가한다고 볼 수 있다.
- [11, 13, 14 항목] 클래스의 메소드와 데이터 멤버의 수가 작아지고 메소드의 크기가 줄었다. 이 항목들은 C&K의 WMC 항목과 관련이 있는데, 클래스의 멤버의 수가 많아질수록 응용에 종속적인 것으로 알려져 있다. 이들 값으로부터 버전 2.x는 응용에 독립적인 구조를 가지고 재사용성이 향상되었음을 알 수 있다.
- [15, 16, 17 항목] 상속의 깊이가 증가하였다. 이 항목들은 C&K의 DIT 항목과 관련이 있는데, 상속의 깊이가 증가할수록 시스템이 복잡해지는 반면 재사용성이 향상된다. 이들 값의 증가는 본 연구에서 복잡성을 희생하고 재사용성 향상에 노력하였음을 보여 준다.
- [18, 19, 20 항목] 서브클래스의 수가 증가하였다. 이 항목들은 C&K의 NOC 항목과 관련이 있는데, 서브클래스의 수가 증가할수록 재사용성이 향상된다.
- [21, 22 항목] 클래스간의 커플링이 증가하였다. 이 항목들은 C&K의 CBO 항목과 관련이 있는데 불행하게도 커플링이 증가했음을 알 수 있다. 그 이유는 버전 2.x에서 클래스의 수가 많아 졌기 때문이거나 커플링 측정 방법의 불완정성 때문이라고 판단된다.[5]
- [23 항목] Law of Demeter[11,12] 위반 횟수가 증가하였다. 이 법칙이 객체 지향 구현을 위한 지침으로 이용되기는 하지만 이 법칙을 적용하면 클래스의 멤버 함수 수가 증가하는 문제를 초래한다. OODesigner에 속하는 클래스들은 이미 많은 수의 멤버 함수들을 포함하기 때문에 멤버 함수의 수가 증가하는 것을 방지하려고 노력하였다. 이러한 이유로 이법칙의 위반횟수가 증가하게 되었다.

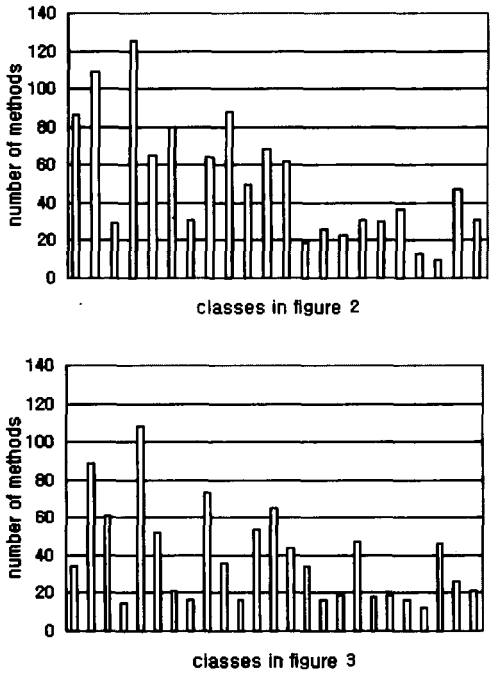
<표 1>의 내용이 나름대로 중요한 정보를 제시하지만 약간의 문제점을 내포하고 있다. 그 문제점은 두 버전간의 클래스 수의 차이가 너무 크다는 것이다. 이러한 문제점을 고려한 또 다른 메트릭 데이터가 <표 2>이다. 버전 1.x와 버전 2.x에서 똑같은 기능을 수행하는 클래스 클러스터가 있는데 그것은 OMT의 표기법 관련 객체들을 모델링한 클래스들이다. 버전 1.x에서는 22개의 클래스로 구성되어 있고, 버전 2.x에서는 25개의 클래스로 구성되어 있다. 그러므로 <표 2>는 두 버전간의 보다 정확한 비교를 위한 메트릭 데이터이다. 그리고 <표 2>의 개별 클래스의 크기, 데이터 멤버 수, 그리고 메소드 수의 차이를 (그림 4), (그림 5), (그림 6)으로 나타냈다.



(그림 4) 개별 클래스의 크기
(Fig. 4) Class Size Distribution



(그림 5) 개별 클래스의 데이터 멤버 수
(Fig. 5) Data Members Distribution



(그림 6) 개별 클래스의 메소드 수
(Fig. 6) Method Distribution

〈표 2〉 두 버전간의 표기법과 관련된 클래스들의 메트릭 비교
〈Table 2〉 Metric Comparison between Notation-realted Classes

메트릭스	버전 1.x	버전 2.x	증감비율
01. 서브시스템의 라인 수	31219	21894	-30%
02. 서브시스템의 실행문 수	18805	12226	-35%
03. 서브시스템의 조건문 수	3688	2499	-32%
04. 서브시스템의 루프 수	771	517	-33%
05. 서브시스템의 클래스 수	22	25	+14%
06. 클래스의 전체 실행문 수	18805	12226	-35%
07. 클래스의 평균 실행문 수	855	489	-43%
08. 클래스의 전체 유일한 단어 수	1707	1549	-9%
09. 클래스의 평균 유일한 단어 수	78	62	-21%
10. 클래스의 전체 데이터 멤버 수	346	196	-43%
11. 클래스의 평균 데이터 멤버 수	15.7	7.8	-50%
12. 클래스의 전체 메소드 수	1124	958	-15%
13. 클래스의 평균 메소드 수	51	38	-25%
14. 메소드의 평균 실행문 수	17	13	-24%
15. 최대 상속 깊이	3	4	+33%
16. 상속 깊이 전체 누적 수	41	58	+41%
17. 평균 상속 깊이	1.86	2.32	+25%
18. 최대 하위 클래스 수	9	7	-22%
19. 전체 하위 클래스 수	21	24	+14%
20. 평균 하위 클래스 수	0.95	0.96	+1%
21. 객체간의 전체 커플링 수	233	249	+7%
22. 객체간의 평균 커플링 수	10.6	9.96	-6%
23. Law of Demeter 위반 횟수	300	438	+46%

3.2 재구조화를 통해 얻은 교훈들

재구조화를 통해 얻은 교훈들은 다음 두 가지로 요약된다. 이들은 이미 객체 지향 분야에서 일반적인 상식으로 거론되기도 하는 것이지만 본 논문에서는 이러한 교훈들을 실험적으로 경험하였다는 점에서 다시 강조하고자 한다. 우선 객체 지향 설계 및 구현 시 유의할 기술적인 관점에서 얻은 교훈은 다음과 같다.

- 클래스의 크기, 즉 클래스 멤버의 수와 메소드의 크기를 가능한 작게 작성하는 것이 바람직하다.
- 상속을 많이 사용하는 것이 바람직하다. 상속의 깊이와 하위 클래스의 수가 증가할수록 재사용성이 증가한다. 그러나 최대 상속 깊이는 제한되어야 하며, 경험적으로 최대 상속 깊이가 6, 또는 7 이상이 되면 이해가 어렵고 복잡도가 증가함으로서 효율성이 저하됨을 알 수 있었다.
- 동적 바인딩을 최대한 사용하는 것이 바람직하다. 객체 지향 기법의 초보자들은 동적 바인딩의 사용을 기피하는 경향이 있다. 그러나 동적 바인딩은 시스템의 유연성을 증가시키기 때문에 적극 사용해야 한다.

아울러 객체 지향 프로젝트의 관리적 측면에서 얻은 교훈은 다음과 같다.

- 비록 구조적 기법의 활용에 능숙한 시스템 공학자라도 첫 번째 객체 지향 과제에서는 실패의 가능성이 매우 높다. 이러한 실패는 구현된 시스템이 요구사항을 만족시키지 못한다는 것을 의미하는 것이 아니라 작동하는 시스템을 만들 수는 있어도 유지보수가 쉬운 시스템을 만드는 것이 어렵다는 것을 의미한다.
- 객체 지향 과제는 객체 지향 방법론과 객체 지향 언어 그리고 객체 지향 CASE 도구를 종합적으로 적용할 경우에만 성공적으로 수행될 수 있다.
- 기존의 객체 지향 소프트웨어에 대해 재구조화의 필요성을 느끼면 단호히 재구조화를 수행해야 한다. 만약 재구조화를 미루게 되면 언젠가는 심각한 보수 유지 문제를 겪게 된다.
- 클래스, 데이터 멤버 그리고 메소드에 적절한 이름을 사용하면 소프트웨어의 이해도가 향상된다. 즉 좋은 명명 규약은 사고의 복잡도를 감소시킨다.
- 잘못 설계된 객체 지향 소프트웨어는 유지 보수 작업을 매우 힘들게 하지만 잘 설계된 객체 지향 소프트웨어는 시스템 엔지니어의 작업을 보람있게 한다.

4. 결 론

본 논문에서는 객체 지향 CASE 도구인 OODesigner를 재구조화하고 추가 기능을 개발하는 과정에서 얻은 경험들을 기술하였다. 우선 초기 버전의 단점에 대해 기술하였고 이를 극복하기 위한 재구조화의 목표, 내용, 과정과 재구조화 결과 얻어진 효과에 대해 기술하였다. 또한 두 버전간의 메트릭 데이터를 비교 분석하였고, 마지막으로 재구조화를 통해 얻은 교훈들을 제시하였다. 결론적으로 재구조화를 통해서 시스템은 수정, 기능 추가, 이식이 용이해지고, 유연성이 증가하여 시스템의 보수 유지가 용이해졌음을 알 수 있었다.

현재 본 연구자는 OODesigner 2.x에 새로운 기능들을 추가하고 있으며, 아울러 다른 플랫폼으로의 이식을 시도하고 있다. 이러한 작업들은 OODesigner 2.x의 기기 독립적이고 유연한 특성으로 인해 매우 생산성 있게 진행되고 있다.

본 연구의 최종 목표는 그래픽 편집기에 기반을 둔 객체 지향 개발 환경의 완벽한 구현이다. 특히 UML [4]을 지원하도록 본 도구를 확장하는 연구와 분산 환경에서 사용될 수 있도록 본 도구를 확장하는 연구도 진행 중이다.

참 고 문 헌

[1] G. Booch, *Object Oriented Design with Application*, Benjamin Cummings, 1991.

[2] S.R. Chidamber and C.F. Kemerer, "A Metric Suite for Objected Oriented Design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp.476-493, 1994.

[3] R.G. Fishman and C.F. Kemerer, "Object Oriented and Conventional Analysis and Design Methodologies," *IEEE Computer*, Vol.25, No.10, pp. 22-40, 1992.

[4] M. Fowler and K. Scott, *UML Distilled : Applying the Standard Object Modeling Language*, Addison Wesley, 1997.

[5] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metric Suite : A Measurement Theory Perspective," *IEEE Transactions on Software Engineering*, Vol.22, No.4, pp.267-270, 1996.

[6] S. Honiden et. al., "An Application of Artificial Intelligence to Object-Oriented Performance Design for Real-Time Systems," *IEEE Transactions on Software Engineering*, Vol.20, No.11, pp.849-867, 1994.

[7] D. Inga, T.Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the Future : The Story of Squeak, A Practical Smalltalk Written Itself," *In Proceedings of OOPSLA'97*, pp.318-326, GA, USA, October 1997. ACM.

[8] T.G. Kim and C.S. Wu, "Design and Implementation of an Object-Oriented Design Tool," *Journal of KISS*, Vol.21, No.5, pp.909-921, 1994.

[9] T.G. Kim and C.S. Wu, "Design and Implementation of a Reverse Engineering Tool for C++," *Journal of KISS(C)*, Vol.1, No.2, pp.135-146, 1995.

[10] T.G. Kim and G.S. Shin, "Restructuring OODesigner, A CASE Tool for OMT," *In Proceedings of the 20th International Conference on Software Engineering, ICSE'98*, pp. 449-451, Kyoto, Japan, April 1998.

[11] K. Lieberherr, I. Holland, and A. Riel, "Object-Oriented Programming : An Objective Sense of Style," *In Proceedings of OOPSLA'88*, pp.323-334, 1988.

[12] K. Lieberherr and I. Holland, "Assuring Good Style for Object-Oriented Programs," *IEEE Software*, pp.38-48, September 1989.

[13] P.H. Loy, "A Comparison of Object-Oriented and Structured Development Methods," *ACM SIGSOFT SE Notes*, Vol.15, No.1, pp.44-48, 1990.

[14] D. Marca and C.M. Gowan, *SADT, Structured Analysis and Design Technics*, McGraw-Hill, 1987.

[15] I. Moore, "Automatic Inheritance Hierarchy Restructuring and Method Refactoring," *In Proceedings of OOPSLA'96*, pp.235-250, CA, USA, 1996.

[16] W.F. Opdyke, *Refactoring Object-Oriented Frameworks*, PhD thesis, University of Illinois at Urbana-Champain, 1992. (ftp://st.cs.uiuc.edu/pub/papers/refactoring/opdyke-thesis.ps.Z)

[17] M.W. Price and Sr S.A. Demurjian, "Analyzing and Measuring Reusability in Object-Oriented Designs," *In Proceedings of OOPSLA'97*, pp. 22-23, GA, USA, October 1997. ACM.

[18] J. Rumbaugh, et al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

조 장 우

e-mail : jjw@taejo.pufs.ac.kr

1992년 서울대학교 계산통계학과 졸업(학사)

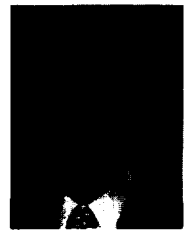
1994년 서울대학교 대학원 전산과학과(이학석사)

1996년 한국과학기술원 전산학과 박사수료

1994년~1996년 한국과학기술원 전산학과 연구조교

1997년~현재 부산외국어대학교 컴퓨터공학과 조교수

관심분야 : 프로그램 분석, 최적화 컴파일러, 객체지향 프로그래밍, CASE 도구



김 태 균

e-mail : ktg@taejo.pufs.ac.kr

1985년 서울대학교 자연과학대학 졸업(학사)

1987년 서울대학교 계산통계학과 (석사)

1995년 서울대학교 계산통계학과 (박사)

1988년~현재 부산외국어대학교 컴퓨터공학과 부교수

관심분야 : 소프트웨어공학, 객체지향 방법론, CASE 도구, 개발 환경