C++ 프로그램의 유지보수 지원 시스템 개발

문 양 선[†] · 장 근 실[††] · 유 철 중[†††] · 장 옥 배[††††]

요 약

본 논문에서는 C++ 프로그램의 유지보수를 지원하는 도구를 개발하였다. C++-MT라고 명명된 본 도구는 기존의 연구 결과인 복잡도 측정 도구(CT)와 프로그램 시각화 도구(VT)에 본 논문의 연구 결과인 프로그램 문서화 지원 도구(DT)와 객체지향 설계 및 프로그래밍 스타일 지침 지원 도구(OOD/P-GT)를 추가함으로써 확장되었다. 확장을 위해 먼저 객체지향 프로그램의 문서화 방법을 연구하였으며, 기존의 몇몇 연구들에서 제안한 객체지향 설계 및 프로그래밍 스타일 지침들을 여러 가지 관점에서 분석하고 분류하였다. 본 논문에서 개발한 문서화 지원 도구는 프로그램의 이해, 변경, 테스트, 그리고 재사용 성분 추출을 용이하게 하며, 객체지향 설계 및 프로그래밍 스타일 지침 지원 도구는 프로그램의 품질을 향상시킬 수 있도록 안내한다.

Development of a Maintenance Support System for C++ Programs

Yang-Sun Moon[†] · Gun-Sil Jang[††] · Cheol-Jung Yoo[†††] · Ok-Bae Chang[††††]

ABSTRACT

This paper introduces a maintenance support tool for C++ programs. The tool was extended by adding a documentation supporting tool(DT) and an OOD/OOP style guidelines supporting tool(OOD/P-GT) to previous work results, the complexity measurement tool(CT) and the visualization tool(VT). The tool was named C++-MT. For the extention, we studied a documentation method of object-oriented programs, analyzed and grouped the OOD/OOP style guidelines suggested in some works. The DT developed in this paper helps programmer understand, change, and test programs, and also extract the reuse components. OOD/P-GT provides guidelines for programmer to improve program quality.

1. 서 론

객체지향 기법은 이미 개발된 프로그램 성분들의 재사용 및 변경을 용이하게 하여 소프트웨어 생산성 증가를 가져오고 있다. 그러나 문헌 [1], [2] 그리고 [3]에서 객체지향 특성 중 상속성과 다형성이 프로그램의 이해를 어렵게 하는 요인이 되고 있음을 지적하였다. 소프트웨어 개발비용 중 유지보수 비용은 전체의 60~70%를 차지하고, 이해작업은 유지보수 전체 비용의 50%이상을 차지한다는 연구보고[4]에서 알 수 있듯이 이해의 어려움은 생산성 저하를 가져오게 된다.

대형 소프트웨어 시스템의 개발자와 유지보수 요원에게 있어서 가장 중요한 문제는 그 시스템 전체의 구

※ 이 연구는 1996년도 한국과학재단 연구비 지원에 의한 결과임 (과제번호 : 961-0906-039-1).
† 정 회 원 : 서해대학 사무자동화과
†† 준 회 원 : 광양대학 전자계산학과
††† 종신회원 : 전북대학교 컴퓨터과학과
†††† 정 회 원 : 전북대학교 컴퓨터과학과
논문접수 : 1998년 1월 15일, 심사완료 : 1998년 5월 6일

조를 파악하는 것이다. 객체지향 프로그램에서는 프로그램 내에 존재하는 클래스 및 메소드에서 정의된 변수 이용의 이해와 메시지 전달 경로의 추적 등은 개발자와 유지보수자가 공통적으로 직면하는 문제이다. 객체지향 프로그램은 상속이나 다형성으로 인해 메시지 전달이 복잡하기 때문에 이러한 문제들의 해결방안이 더욱 필요하다. 본 논문의 기반이 되는 문헌 [3]에서는 이를 해결하기 위해 C++ 프로그램을 기반으로 하여 구성 성분들간의 관계를 시각화하는 역공학 도구를 개발하였다. 그러나 이 도구는 기존의 역공학 도구들처럼 언어의 구문분석을 기반으로 하기 때문에 모듈의 기능, 변수의 용도 등 프로그램 이해를 위한 충분한 정보를 표현하는데 한계를 갖고 있다.

프로그램의 분석 및 이해를 돕기 위해 행해진 연구들은 크게 두 가지 부류로 나누어 볼 수 있다. 그 중 하나는 원시코드를 입력받아 구문분석을 통해 프로그램의 구조 및 변수의 사용 범위 그리고 모듈간의 관계들을 보여주는 역공학 도구의 개발이며, 다른 하나는 모듈이나 특정 라인에 관한 주석을 프로그램 내에 삽입하는 방법이다. 전자의 경우는 프로그램 이해를 위한 충분한 정보를 표현하기 어렵고, 후자의 경우는 주석을 별도로 삽입해야 하는 업무 부담이 따른다는 점과 주석을 삽입하는 프로그래머의 표현 방법에 의존하므로 그로 인한 오해의 소지가 있다는 단점이 있다. 역공학 도구의 한계와 주석 표현의 단점을 극복하기 위해서는 프로그램 문서화 방법을 정형화시키는 것이 필요하며 그 방법을 바탕으로 자동으로 문서화된 프로그램을 생성할 수 있는 도구의 개발이 필요하다.

소프트웨어 공학을 '컴퓨터 프로그램의 설계와 구성 그리고 프로그램들을 개발하고 운영하고 유지 보수하는데 필요한 관련 문서들에 대한 과학적인 지식의 실제적인 응용'[5] 이라고 정의한다면 이 정의를 통해서 프로그램 문서의 구성이 매우 중요하다는 것을 알 수 있다. 그러나 소프트웨어 개발 및 유지보수에서 사용되는 프로그램 문서를 개발하기 위한 방법론이나 기법들에 대한 연구는 아주 미흡하고, 특히 요구분석 명세서와 설계 문서와 같은 소프트웨어 라이프 사이클의 초기단계 산출물들이나 프로그램 원시 코드들 사이의 일관성을 유지하는 기법들은 더욱 부족한 실정이다[6].

한편, 문헌 [7, 8, 9, 10, 11, 12, 13]등에서 품질 좋은 객체지향 설계와 프로그래밍을 돕기 위해 많은 지침들을 연구하여 제시하였다. 이러한 지침들은 이미 개발된 소프트웨어를 변경하거나 확장할 때도 이용될 수 있다. 그러나 이것들은 서로 다른 연구자들에 의해 논문이나 책에서 단순 나열 식으로 소개되었기 때문에 다양한 수준의 소프트웨어 개발자들이 자신이 원하는 지침들만을 참고하고자 할 때 번거로움이 있다. 따라서 효율적인 참고를 위해서는 개발자의 수준이나 지침의 중요도 등 여러 관점에서 분석 및 분류가 필요하다.

따라서 본 논문에서는 객체지향 프로그램 문서화 방법을 정형화하고, 이를 토대로 C++ 원시 프로그램이 반자동으로 문서화될 수 있도록 지원하는 기능과 양질의 객체지향 설계 및 프로그래밍을 안내할 수 있도록 이미 제안된 지침들을 분류하여 사용자가 원하는 그룹의 지침을 제공하는 기능을 갖도록 문헌 [3]을 확장하였다. C++-MT라고 명명된 이 도구는 문헌 [3]의 결과인 복잡도 측정 도구(CT: Complexity measuring Tool)와 시각화 도구(VT: Visualization Tool)에 본 논문의 연구 주 내용인 프로그램 문서화 지원 도구(DT: Documentation Tool)와 객체지향 설계 및 프로그래밍 스타일 지침 지원 도구(OOD/P-GT: OOD/OOP Guidelines Tool)가 추가된 4개의 모듈로 구성된다. CT와 VT에 대해서는 문헌 [3]에서 자세히 기술하였으므로, 본 논문에서는 DT와 OOD/P-GT에 중점을 두어 기술한다. 도구의 구현은 X 윈도 시스템에서 이루어졌으며 역공학 과정에서의 파서 구현은 lex와 yacc을 이용하였고 사용자 인터페이스를 위해서는 Tcl/Tk가 사용되었다.

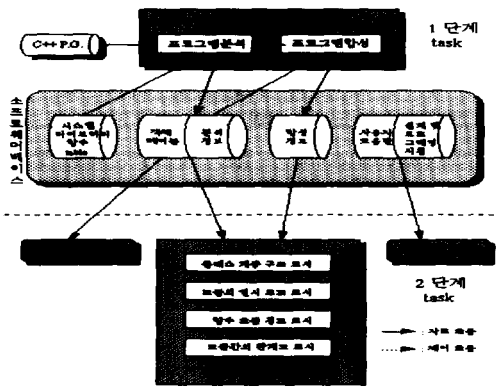
본 논문의 구성은 다음과 같다. 2장에서는 C++-MT의 이전 도구의 구성과 확장된 도구의 구성에 대해서 개략적으로 논한다. 3장에서는 이전 도구에서 확장된 DT와 OOD/P-GT 개발을 위한 사전 연구와 프로그램 문서화 관련 연구에 대해서 논하고, 4장에서는 DT와 OOD/P-GT의 구성 및 기능에 대해서 기술한 후 도구에 대한 평가를 할 것이다. 마지막으로 5장에서는 결론과 향후 연구과제에 대해서 논한다.

2. C++-MT의 구성

2.1 기존 도구의 구성

(그림 1)은 기존의 도구[3]에 대한 시스템 구성도이다. (그림 1)을 통해 알 수 있듯이 기존 도구는 프로그램 분석 모듈, 품질 평가모듈, 이해 지원 모듈, 도움말 모듈로 구성되어 있다.

프로그램 분석 모듈에서 C++ 프로그램을 입력으로 받아 클래스에 관한 정보, 멤버함수에 관한 정보, 일반 함수에 관한 정보, 'main'함수에 관한 정보들을 분석하고, 그것들 간의 관계성을 고려하여 소프트웨어 베이스의 분석 정보에 저장한다. 또한 상속과 다형성 문제를 해결하기 위해 호출된 함수가 어느 클래스에서 정의되었지를 알아내는데 필요한 정보를 추출하여 객체 테이블에 저장한다. 그리고 나서 분석정보와 객체 테이블에 저장된 자료를 이용하여 'main' 함수를 통해 호출되는 함수의 이름과 그 함수를 정의하고 있는 클래스 이름을 합성시켜 소프트웨어 베이스의 합성정보에 저장한다.



(그림 1) 기존 시스템의 구조
(Fig. 1) Architecture of existing system

프로그램을 분석할 때 토큰이 사용자 정의 함수인지 시스템 함수인지를 구별하기 위해서 시스템 이름이 기록되어있는 시스템 라이브러리 함수 테이블을 참고한다. 이렇게 하여 프로그램의 분석이 완전히 수행되면 품질 평가 모듈이나 이해지원 모듈에서 분석 정보와 합성 정보를 이용하여 사용자가 원하는 정보를 보여주게 된다.

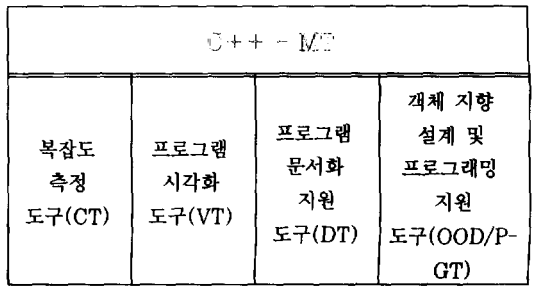
품질 평가 모듈에서는 문헌 [7]에서 제안된 복잡도 척도에 기반을 두어 C++ 프로그램의 복잡도를 측정해 준다. 이해 지원 모듈에서는 클래스 계층 구조, 함수 호출 경로 과정, 모듈간의 관계도를 시각적으로 보여주어 복잡한 메시지 전달 과정을 추적하고 호출된 다형성 함수가 어느 클래스에서 정의된 것인지를 쉽게 알 수 있도록 보여준다. 이것은 문헌 [1, 2, 14, 15]에서 해결해 주지 못하는 기능이다.

도움말 파일과 설계 및 프로그래밍 지침 파일은 미리 소프트웨어 베이스에 저장되어 있어 도움말 모듈에서 보여주는 정보를 담고 있다. 설계 및 프로그래밍 지침 파일은 문헌 [7, 8, 9, 10, 11, 12, 13]에서 발췌한 지침들로서 아무런 기준 없이 단순 나열식으로 저장되어 있다.

2.2 확장된 도구의 구성

원시 코드의 구문 분석을 기반으로 하는 역공학 도구들은 성분들에 관한 자세한 정보를 제공하는데 한계가 있다. 기존의 도구 역시 이러한 문제를 갖고 있다. 따라서 기존의 도구를 보완하여 더욱 효율적인 유지보수 지원 시스템이 되도록 다음과 같이 확장하였다.

(그림 2)는 기존의 도구를 확장하여 'C++-MT'라고 명명한 도구의 서브 툴들을 보여준다. 이 서브 툴들 중 CT와 VT는 기존 도구 [3]의 품질평가 모듈, 이해지원 모듈을 지칭하며, 본 논문에서 DT, OOD/P-GT가 확장되었다.



(그림 2) C++-MT의 구성
(Fig. 2) Structure of C++-MT

기존의 도구 CT, VT에 DT, OOD/P-GT가 확장될 때 도구의 내부 구조는 달라지는 것이 거의 없다. DT에서도 (그림 1)의 프로그램 분석 모듈에 의해서 분석된 정보들을 소프트웨어 베이스로부터 이용하며, 단지 OOD/P-GT에서 (그림 1)의 소프트웨어 베이스의 설계 및 프로그래밍 지침 파일의 내용이 메모리의 효율을 위해 새로운 구조로 변경되었다.

DT와 OOD/P-GT를 개발하기 위한 사전 연구로 본 논문에서는 C++ 프로그램 문서화 방법을 연구·제안함과 동시에 품질 좋은 객체지향 설계 및 프로그래밍을 위한 스타일 지침들을 분석하여 분류하였다.

3. 도구 확장을 위한 방안

3.1 C++ 프로그램 문서화

여기에서는 객체지향 언어 중 널리 사용되는 C++를 기반으로 프로그램 문서화 방법을 제안한다.

3.1.1 문서화 방법

1) 개발 초기 단계의 산출물들과 일관성을 유지하기 위한 문서화 정보

개발 초기단계는 소프트웨어 생명주기에서 요구분석 단계와 설계 단계를 의미한다. 본 논문은 원시코드를 연구 대상으로 하기 때문에 요구 분석이나 설계 단계에서 산출되는 모든 문서 영역을 포함하지 않는다. 소프트웨어 요구분석 명세서는 소프트웨어 계획서에 명시된 소프트웨어 영역을 기초로 하여 소프트웨어의 기능 및 성능을 설정하고 관련 자료구조를 설명하며 요구시험 기준이나 설계할 때 제약 조건 등과 같은 본질적인 요구사항 및 외부 인터페이스에 대한 사항을 기술하게 되며, 설계 명세서는 소프트웨어 구조에 관한 사항을 다루고 구성요소의 세부사항을 포함하게 된다[16]. 이러한 명세서 내용들 중 유지보수나 재사용에 도움을 주는 정보는 소프트웨어의 기능과 관계 자료구조 및 시스템 요소들간의 접속관계에 관한 내용이 된다. 이런 관점에서 본 논문에서 정의된 요구분석 명세서 및 설계 명세서에 기록될 내용 중 유지보수나 재사용에 직접 필요한 문서화 정보는 다음과 같다.

①클래스 관련 정보

Cl_Name : 클래스 명

Cl_Role : 클래스의 기능에 대한 설명

Cl_Precondition : 클래스의 전제조건. 예를 들어 스택을 구현한 클래스가 존재한다면, 그 클래스의 전제조건은 데이터가 입력되기 전에 반드시 저장공간이 할당되어 있어야 한다는 내용 등이 포함됨

Cl_Constraint : 클래스에 대한 제한사항 정보. 예를 들어, 스택을 구현한 클래스가 존재하고, 그 스택 클래스에 데이터를 저장할 수 있는 공간이 200개로 제한되어 있을 경우 그러한 상황에 관한 정보

CL_Error : 에러 발생 경우와 에러 발생시 예외처리 모듈에 대한 정보로서 예를 들면, 데이터 입력조건이나 잘못된 입력 등으로 에러가 발생했을 때 실행될 함수나 해결책 혹은 피할 수 있는 방법에 관한 정보

Cl_InstanceName : 인스턴스 명

Cl_ParentName : 슈퍼클래스 명

Cl_MfName : 멤버함수 명

Cl_DataMemberName : 데이터 멤버 명

Cl_DataMemberRole : 클래스에서 정의된 데이터 멤버들이 갖는 의미나 사용 범주 등에 대한 정보

Cl_FriendName : 프렌드 함수 명

Cl_Type : 클래스의 type에 대한 정보로서 추상 클래스이면 'Abstract', 기반 클래스이면 'Base', 가상클래스이면 'Virtual'로 기술

Cl_Scope : 클래스의 범위에 관한 정보로서 외부 모듈에서 참조하는 클래스라면 'External', 정적 클래스면 'Static', 전역 클래스면 'Global'로 기술

Cl_ParentAccess : 접근 지정자에 대한 정보로서 어떤 클래스로부터 상속받을 경우 부모 클래스에 대한 접근 지정자를 기술

②멤버함수 관련 정보

Mf_Name : 멤버함수 명

Mf_Type : 멤버함수의 형(type)에 대한 정보로 다음과 같은 정보들이 혼합 혹은 단독으로 기술된다. 일반적으로 C++에서 멤버 함수의 종류들은 생성자의 경우 'constructor', 'default constructor', 'copy constructor', 소멸자의 경우는 'destructor', 연산자 오버로딩의 경우 'overload', 가상함수의 경우 'virtual' 또는 'pure virtual', 프렌드 함수의 경우 'friend'로 기술

Mf_Role : 멤버함수의 기능에 대한 설명

Mf_Precondition : 멤버함수 실행에 앞서 만족되어야 할 조건 정보

Mf_SideEffect : 멤버함수의 실행에 의해 영향받는 모듈의 이름이나 변수들과 같은 부작용 관련 정보

Mf_Prototype : 멤버함수의 원형

Mf_Argument : 멤버함수의 인수 및 초기 값

Mf_Scope : 멤버함수가 갖는 범위 기술

Mf_Access : 접근 지정자

Mf_MsgSender : 멤버함수에 메시지를 보내는

함수(호출하는 함수)
Mf_MsgReceiver : 멤버함수로부터 메시지를 받는 함수(호출 당하는 함수)

③일반함수 관련 정보
Gf_Name : 일반함수 명
Gf_Role : 일반함수의 기능 정보
Gf_Precondition : 일반함수의 전제 조건
Gf_SideEffect : 일반함수에 의해 영향받는 모듈에 관한 정보
Gf_Prototype : 일반함수의 원형
Gf_Argument : 일반함수의 인수
Gf_Scope : 일반함수의 범위
Gf_MsgSender : 일반함수에 메시지를 보내는 함수(호출하는 함수)
Gf_MsgReceiver : 일반함수로부터 메시지를 받는 함수(피 호출 함수)

2) 프로그램의 자세한 이해를 위한 문서화 정보

프로그램의 재사용을 위해서는 소프트웨어 각 구성 성분들과 그들간의 관계에 대한 이해가 우선 되어야 한다. 앞에서 정의한 문서화 정보들도 소프트웨어를 재사용할 때 도움을 줄 수 있는 정보이지만 이 외에 구체적으로 프로그램의 이해를 도울 수 있도록 정의된 정보들을 구성 성분별(단위 파일, 클래스, 멤버함수, 일반함수)로 나누어 보면 다음과 같다.

①단위 파일 이해 정보
Fl_Name : 파일 이름
Fl_Program : 프로그램이나 프로젝트의 이름
Fl_Include : Include(헤더) 파일 목록
Fl_Date : 모듈/프로그램 단위를 완성한 날짜
Fl_Version : 프로그램 버전
Fl_Environment : 개발환경
Fl_Target : 목적 모듈의 실행환경
Fl_Author : 작성자 명
Fl_Address : 작성자 주소
Fl_Email : 작성자의 전자우편 주소
Fl_Name : 파일 이름
Fl_Characters : 이 정보는 현재의 단위 파일의 성질을 표현하며, 'Extendable', 'Executable', 'Include' 중 하나가 된다. 실행 가능한 완전한 독립 프로그램이면 'Executable', 일반함수들의 모임이나 클래스나 멤버함수에 대한 기술 프로그램이라면 'Extendable', 그리고 순수한 헤더파일(header file)일 경우 'Include'로 기술
Fl_Contents : 프로젝트를 구성하는 모든 파일들의 목록(헤더파일은 제외)
Fl_Role : 단위 파일의 기능 정보
Fl_Misc : 컴파일이나 링크 옵션, 최소 실행환경 등 기타 정보

②클래스 이해 정보
Cl_Date : 클래스 작성일
Cl_Version : 클래스 버전
Cl_Author : 개발자 명
Cl_Environment : 클래스 작업 환경
Cl_Constraint : 일반적인 제약 사항에 대한 정보로서, 클래스에 대한 기능의 한계나 주의해야 할 사항에 대한 정보
Cl_Error : 에러 및 예외 처리 - 데이터 입력 조건이나 잘못된 입력 등 에러가 발생할 때 실행될 함수 명을 기술

③멤버함수 이해 정보
Mf_BlockRole : 특정 블록의 기능 정보
Mf_LineRole : 특정 라인의 기능 정보
Mf_VarRole : 특정 변수의 기능 정보
Mf_Polymorphism : 다형성으로 정의된 멤버함수의 위치 및 원형

④일반함수 이해 정보
Gf_BlockRole : 특정 블록의 기능 정보
Gf_LineRole : 특정 라인의 기능 정보
Gf_VarRole : 특정 변수의 기능 정보
Gf_Polymorphism : 다형성으로 정의된 일반함수의 위치 및 원형
Gf_InstanceName : 모듈에서 생성되는 인스턴스와 클래스

지금까지 정의한 문서 정보들은 기존의 유지보수 관련 도구에서 얻을 수 있는 정보도 있지만 그렇지 못한 정보, 즉 프로그램 구성성분(클래스, 함수들, 특정 원시코드)들의 기능이나 이들이 변경될 때 나타날 수 있는

부작용(side effect)에 관한 설명과 작성자들의 정보를 기술함으로써 유지보수 관련 작업들에서 도움을 줄 수 있도록 하였다.

3.1.2 프로그램 문서화의 효율성

본 논문에서 정형화한 문서화 정보들은 직접 프로그래머가 코딩할 때 삽입해야 하기 때문에 프로그래머의 업무 부담이 증가되지만, 소프트웨어의 유지보수 단계가 개발 주기의 전체 비용에서 60% 이상을 차지하고, 뿐만 아니라 프로그램의 재사용을 위해서는 원시코드의 분석 및 이해가 선행되어야 한다는 것과 아울러 역공학 도구의 한계를 감안할 때 이상과 같은 객체지향 프로그램 문서화는 반드시 필요하다.

따라서 위에서 정의된 문서화 정보로 C++ 원시 프로그램을 문서화함으로써 얻을 수 있는 장점들은 다음과 같다.

1) 요구 변경에 다른 프로그램 변경 용이

요구분석 명세서 및 설계 명세서와 같은 초기 개발 단계의 기록 내용이 프로그램 문서 안에 들어 있으므로 요구변경이 있을 경우 변경시킬 프로그램 요소를 쉽게 찾을 수 있어 요구분석 명세서나 설계 명세서가 존재하지 않을 경우라도 수정이 용이하다.

2) 여러 수준의 사용자들이 요구하는 정보 제공

원시 프로그램을 이해할 때 사용자들은 자신의 환경이나 능력에 따라 다른 종류의 정보를 필요로 한다[17]. 즉 시스템 관리자의 경우 각 라인이나 식별자 등의 자세한 코딩 수준의 정보가 아닌 모듈들 사이의 인터페이스나 알고리즘 혹은 프로그램 전반에 걸친 개념적이고 구조적인 정보를 원하고, 프로그래머일 경우 개념적인 정보보다는 각 모듈이 갖는 상세한 수준의 정보들(예를 들면 모듈을 구성하는 알고리즘의 실제 코딩 정보나 각 라인 또는 변수 등의 역할)을 원한다. 이와 같은 사용자의 수준에 맞는 정보를 효과적으로 얻을 수 있어 원시 프로그램을 이해하는데 도움을 준다.

3) 동적인 정보 획득으로 실행 경로 분석 용이

기존의 브라우저들이나 다른 분석기들의 분석 영역은 원시 코드의 구문 분석에 그치기 때문에 원시 코드를 이해하는데 필요한 동적인 정보의 획득이 어렵다. 예를 들어 동적 바인딩의 경우 제어의 흐름이 컴파일 시간이 아닌 실행 시간에 결정된다. 이로 인해 단순한 원시 코드의 구문 분석으로는 그 표현이 불가능하다. 그러나 프로그램 내부에 삽입된 문서화 정보를 통해서 실행의 전반적인 동적 측면의 정보를 얻을 수 있다. 이와 같은 정보들은 원시 코드 개발자가 가장 효과적으로 제공할 수 있는 정보이기 때문이다.

4) 식별자의 사용 영역(Scope)에 대한 정보 획득으로 테스팅 용이

특정 식별자에 대하여 프로그램에서의 전체적인 용도 및 사용된 영역 정보를 제공하고, 제어문이나 분기문에 사용되는 식별자의 경우 조건이나 값에 대한 정보를 제공하기 때문에 프로그램 테스팅에도 도움이 된다.

5) 프로그램 각 성분들의 기능 정보 획득으로 재사용 성분 추출 용이

기존의 프로그램을 재사용하기 위해서 재사용자는 자신이 원하는 성분이 존재하는가를 알아야 하고, 이를 위해 먼저 프로그램 구성 성분들의 기능을 파악하는 과정이 필요하다. 이러한 점에서 볼 때, 정의된 문서화 정보 중 구성 성분들의 기능 정보는 재사용 가능한 성분 추출에 많은 도움을 줄 수 있다.

3.2 프로그램 문서화 관련 연구

프로그램 문서화에 대한 많은 관련 연구 중에서 주석을 이용했던 몇몇 연구들이 있다. Marovac[6]은 절차지향 언어를 대상으로 'KEYWORDS', 'FACETS', 'TRACEBACK', 'REFERENCE', 그리고 'DESCRIPTION' 등 다섯 종류의 flagged sentences를 사용하여 문서화 정보들을 기술하고 도구에 의해서 추출할 수 있도록 하였다. 이 연구는 절차지향 프로그램에 기반을 두었기 때문에 객체지향 패러다임에 적용하기 어렵고, 주석 정보의 복잡한 구조와 많은 종류의 플래그들을 사용하여 문서화 정보를 표현하기 때문에 문서화 정보를 생성하고 이해하는데 많은 어려움이 따른다.

송후봉[18]은 대상언어를 C++로 하여 클래스, 함수, 시스템 내의 중요 변수와 알고리즘을 기술할 때 각 라인에 해당되는 복합문을 문서화의 대상으로 하였다. 이 연구는 요구분석 명세서 및 설계 명세서와 같은 초기 단계의 기록 내용이 프로그램 문서 안에 들어 있지 않으므로, 요구변경이 있을 경우 변경시킬 프로그램 성분을 찾는데 도움을 주지 못하며 표현할 수 있는 정보

가 빈약하다.

ObjectManual[19]은 C++를 대상언어로 하며, 문서화 내용을 HTML을 이용하여 작성하도록 하였기 때문에 작성된 문서화 정보를 보기 위해서 웹브라우저를 이용해야 하며, HTML 편집기로 문서화 정보 편집 및 갱신이 가능하다. 문서화의 주된 대상은 C++의 기본 데이터 구조인 클래스이다. 이 연구의 장점은 문서화에 필요한 정보를 원시 프로그램의 컴파일 시에 얻기 때문에, 구문에러가 존재하지 않는 정확한 문서화 정보만을 얻는다는 것이다. 생성된 HTML 형식의 문서는 클래스 구조 정보, 클래스에 정의된 메소드들의 번호, 파생 클래스의 메소드 정보, 클래스의 상속 메소드 정보, 슈퍼 클래스와 원시 파일과의 하이퍼링크(hyper-link), 그리고 개발자의 이용을 위한 특수 정보 등을 포함하고 있다. 그러나 클래스만을 문서화 대상으로 한다는 점과 클래스에 대해 제공되는 정보 또한 클래스의 구조나 상속 계층, 메소드들의 인터페이스와 같은 간단한 정도의 정보만을 제공하기 때문에 효율성이 부족하다. 또한 제공되는 문서가 클래스의 상속 구조를 사용자 정의형이 아닌, 언어 자체가 제공하는 추상 클래스나 기반 클래스까지 확대하여 각 클래스마다 하나의 개별적인 파일로 보관하기 때문에 문서를 유지보수하는데 어려움이 따른다.

3.3 객체지향 설계 및 프로그래밍 스타일

3.3.1 지침 분석

문헌 [7, 8, 9, 10, 11, 12, 13]등에서 품질 좋은 객체지향 설계와 프로그래밍을 돕기 위해 많은 지침들을 연구하여 제시하였다. 여기에서는 다양한 수준의 소프트웨어 개발자들이 쉽게 참고할 수 있도록 중복 지침들을 배제하고, 같은 내용에 대해 서로 다른 의견을 제시해 놓은 지침(예: 문헌 [13]에서는 클래스를 정의할 때 'private' 부분을 'public' 보다 먼저 기술하는 것이 좋다고 하는데 반해 문헌 [10]에서는 그 반대로 제시하고 있음)들을 제외하여 138가지의 지침을 모아 다음과 같이 여러 가지 관점과 특성별로 분류하였다.

1) 중요도에 따른 분류 (G1)

기존의 연구에서는 지침들의 중요성을 강조하기 위해서 'Never'나 'Must'라는 용어를 사용하였으나 새로운 관점으로 지침들을 다음과 같이 소그룹으로 분류하였다.

① 필수적인 지침(G1.1)
메모리 폴트(memory fault)나 실행 오류가 일어나는 상황을 방지하기 위해 제시된 지침들로 만일 이러한 지침들을 지키지 않으면 원하는 결과를 얻을 수 없게 된다.

② 권장 지침(G1.2)
직접 실행 결과에는 영향을 미치지는 않지만 메모리 관리, 실행 시간의 효율 및 프로그램의 빠른 이해를 유도하기 위해 제시된 지침들이 해당된다.

③ 참고 지침(G1.3)
각 참고 문헌에서 권고 사항으로 제시한 지침들이 이에 해당된다.

이러한 분류 목적은 프로그램의 오류를 줄이고 효율적인 프로그램을 개발하도록 유도하는데 있다.

2) 프로그램 성분에 따른 분류 (G2)

프로그램 성분에 관한 관점에서 지침들을 다시 다음과 같이 소그룹으로 분류한다.

① 파일 관련 지침(G2.1)
② 클래스 관련 지침(G2.2)
③ 멤버함수 관련 지침(G2.3)
④ 일반함수 관련 지침(G2.4)
⑤ 코드 관련 지침(G2.5)

이러한 분류 목적은 기존의 절차 지향 프로그래밍 스타일과 프로그램 성분을 비교해 볼 수 있도록 하는데 있다.

3) 객체지향 특성에 따른 분류 (G3)

객체지향 프로그래밍 특성에 관한 관점에서 지침들을 다시 다음과 같이 소그룹으로 분류한다.

① 상속 관련 지침(G3.1)
② 다형성 관련 지침(G3.2)
③ 캡슐화 관련 지침(G3.3)

이러한 분류 목적은 객체지향성이 높은 소프트웨어의 설계와 프로그래밍을 위해서 좋은 참고 자료가 되도록 하자는 데 있다.

4) 프로그램 개발 단계에 따른 분류 (G4)

프로그램 개발 단계에 관한 관점에서 지침들을 다시 다음과 같이 소그룹으로 분류한다.

① 프로그램 관리 관련 지침(G4.1)
다른 프로그램들과의 구분을 쉽게 할 수 있고 프

로그램의 버전(version) 관리를 효율적으로 할 수 있도록 제시된 지침들이 이에 속한다.

② 실행 관련 지침(G4.2)

효율적인 메모리 관리나 실행 시간을 단축 할 수 있도록 유도하는 지침들이 이에 속한다.

③ 이해 관련 지침(G4.3)

프로그램 논리를 이해하기 쉽게 유도하는 지침들이 이에 속한다.

이러한 분류 목적은 유지보수의 비용을 절감할 수 있는 프로그램 생성을 돕는데 있다.

참고 문헌들에서 제시된 지침들 각각은 적어도 G1의 한 소그룹에 속하며 동시에 G2, G3, G4의 한 소그룹들에 속할 수 있다.

3.3.2 지침 분류 예

여기에서는 앞에서 분류한 항목들에 따라 참고문헌에서 제시된 지침을 대상으로 분류한 예를 보인다.

지침: 사용자 정의 자료형 변환은 한 객체에 대하여 오직 한번만 시스템에 의해 자동적으로 수행되게 하라[9].

분류결과: G1.2, G2.5, G4.2

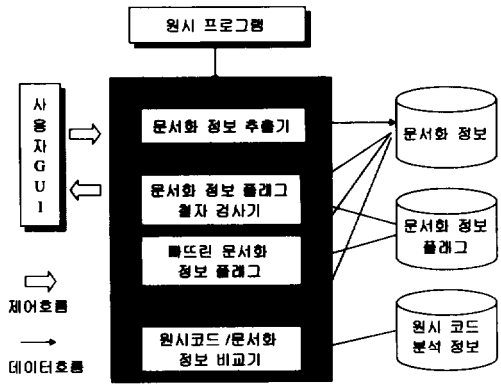
자료형 변환은 실행 효율을 떨어뜨리므로 이 지침은 G1.2 그룹에 속하며, 코드 관련 지침이고 실행 관련 지침이므로 G2.5와 G4.2 그룹에도 속한다.

4. 확장 도구의 기능

4.1 문서화 도구(DT)

4.1.1 구성

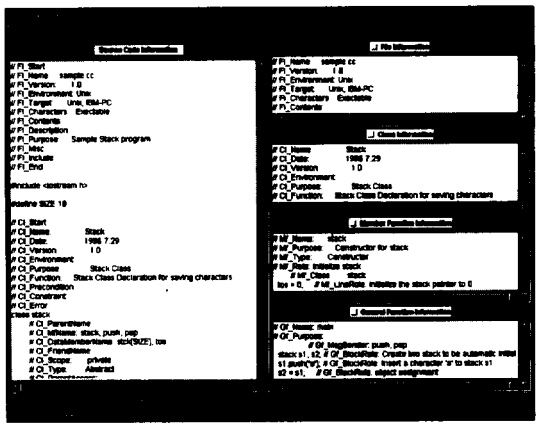
문서화 도구는 (그림 3)에서 보는 것처럼 문서화 정보 추출기, 문서화 정보 검사기, 그리고 원시코드/문서화 정보 비교기로 구성되어 있다. 문서화 정보 추출기는 사용자 GUI를 통해서 선택된 C++ 프로그램 내에 삽입된 문서화 정보들을 추출하여 문서화 정보 데이터베이스에 저장한다. 문서화 정보 검사기는 추출된 소프트웨어 문서화 정보들이 본 논문에서 정의한 방법에 따라 문서화가 잘 이루어 졌는지를 조사하는데, 조사 기준은 문서화 정보 플래그의 코딩 오류와 빠뜨린 문서화 정보가 있는지에 관한 것이다. 원시 코드/문서화 정보 비교기는 (그림 1)에서 분석된 원시코드 분석 정보 데이터베이스와 문서화 정보 데이터베이스의 내용(예: 클래스 이름, 상위 클래스 이름, 호출한 멤버함수 이름 등)이 일치하는지를 비교하여 그 결과를 사용자 GUI를 통해서 보여준다.



(그림 3) DT의 구성
(Fig. 3) Structure of DT

1) 문서화 정보 추출기

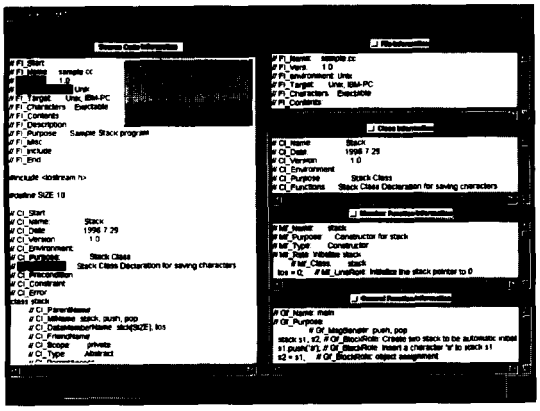
일반적으로 원시 프로그램 내부에 삽입된 소프트웨어 문서 정보들은 각 모듈에 따라 나누어져 실행 코드와 혼합된 형식으로 전체 영역에 분포되어 있기 때문에 필요한 문서 정보만을 효과적으로 이용하기가 어렵다. 이와 같은 문제를 해결하기 위해 문서화된 원시 프로그램으로부터 문서화 정보만을 추출하여 C++ 프로그램의 각 모듈(파일, 클래스, 멤버함수, 일반함수)별로 나누어 (그림 4)와 같이 보여준다.



(그림 4) 문서화 정보 추출
(Fig. 4) Extraction of documentation information

2) 문서화 정보 플래그 철자 검사기

이 모듈은 문서화 작업을 할 때 발생될 수 있는 구문 에러를 최소화하기 위한 것으로 좀 더 안전하고 명확한 소프트웨어 문서화를 보장하기 위한 것이다. 원시 프로그램에 있는 문서화 정보 플래그의 철자를 검사하여 잘못 작성된 플래그를 검출한 후 검출된 플래그의 개수를 계산하여 (그림 5)와 같이 보여준다.



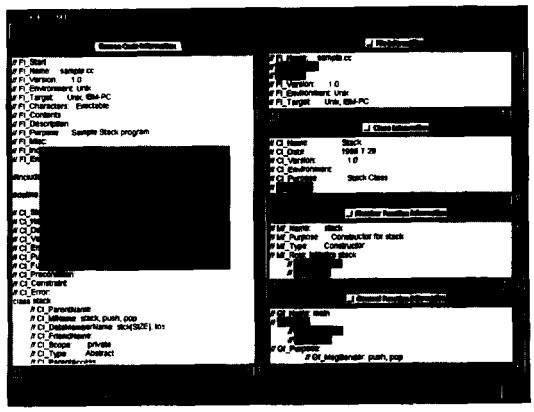
(그림 5) 문서화 플래그 조사
(Fig. 5) Check of documentation flags

3) 빠뜨린 문서화 정보 검사기

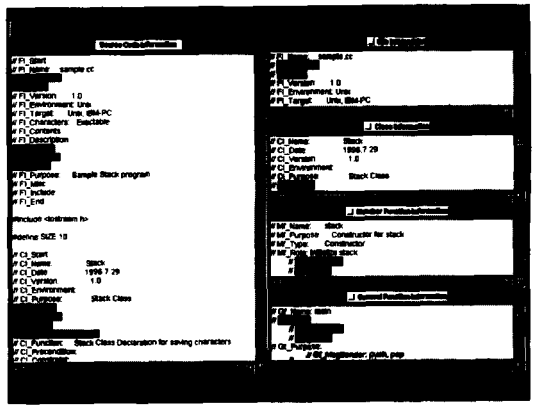
프로그램 문서화의 비효율성을 줄이기 위해 문서화가 자동으로 이루어 질 수 있도록 도구의 개발이 필요하다. 이 모듈은 앞에서 정의한 문서화 정보들 중 일부분을 자동 삽입하는 기능을 갖고 있다. 삽입되는 문서화 정보는 문헌 [3]에서 프로그램 구문 분석을 통해 분석해 놓은 정보들이다. 문서화 정보들에 대해 재사용이나 용이한 이해를 위해 반드시 제공되어야 할 정보들에 대해 생략되거나 실수로 빠뜨린 정보를 검사해서 자동 삽입이 불가능한 정보들은 삽입할 수 있도록 지원하기도 한다(그림 6-1, 6-2 참조).

4) 원시 코드/문서화 정보 비교기

이 모듈에서는 문헌 [3]에서 추출한 원시프로그램의 정보와 문서화 정보 추출기에서 추출한 정보를 모듈별로 비교하여 그 결과를 (그림 7)과 같이 보여준다. (그림 7)은 사용자가 선택한 C++ 프로그램 내의 'push' 멤버함수에 대한 일치성 비교를 실행한 화면이다. 사용자들은 이 기능을 이용하여 각 모듈에 대한 정확성을 효과적으로 유지할 수 있고, 또한 원시 프로그램의 잘



(그림 6-1) 빠뜨린 문서화 정보 조사
(Fig. 6-1) Check of missed documentation information



(그림 6-2) 빠뜨린 문서화 정보 반자동 삽입
(Fig. 6-2) Semi-automated insertion of missed documentation information

못된 내용을 효과적으로 식별할 수 있기 때문에 올바른 코드의 작성에도 도움을 얻을 수 있다.

(그림 7) 원시코드와 문서화 정보의 비교
(Fig. 7) Comparison of source code to documentation information

5) 원시코드 분석 정보 데이터베이스

(그림 1)의 프로그램 분석 모듈로부터 추출된 원시 코드에 대한 정보들을 포함하고 있다.

6) 문서화 정보 데이터베이스

문서화 정보 추출기에서 추출된 정보들을 보관하고 있으며, 텍스트 파일의 형태로 구성된다. DT는 4개의 데이터베이스 파일을 생성하는데, 첫 번째는 모든 주석 라인을 포함하는 파일이고, 나머지 세 개는 정보 비교기에서 이용 가능한 정보들을 포함하는 파일이다. 이들은 클래스와 멤버함수 및 일반함수에 대한 정보를 갖게 되며, 이 파일들은 원시 코드의 내용과 문서화 정보를 비교할 때 이용된다.

4.1.2 구현

1) 문서화 정보 추출기

이 모듈에 대한 알고리즘은 (그림 8)과 같고, 여기에서 이탤릭체로 표시한 부분은 반복될 수 있는 정보임을 나타낸다.

문서화 정보와 원시 프로그램의 정보를 비교할 때 사용될 수 있는 정보들을 추출하여 각 모듈의 정보에 대한 데이터베이스를 생성하는 알고리즘은 (그림 9)와 같다. 여기에 제시된 알고리즘은 클래스에 대한 정보를 생성하는 부분이다. 그리고 〈표 1〉은 이 과정에서 생성되는 각 모듈별 데이터베이스 파일의 내용을 보여주며, 〈표 2〉는 문서화 정보를 추출하면서 생성된 데이터베이스 중 멤버함수 관련 정보를 보여주고, 볼드체로 표시된 부분이 추출된 정보들을 의미한다.

```
#-------------------------------------------------#
#문서화된 원시 프로그램으로부터 문서화 정보 추출 루틴
#-------------------------------------------------#
proc DisplayTexts {} {
    전역변수 선언;
    if { 저장된 정보 } {
      open 입력파일;
      open 출력파일;
      while { end of file } {
            get 라인;
            if { 주석정보 포함 } { puts 출력파일; }
      }     :# end of while
      close 입력파일;
      close 출력파일;
    }  :# end of if
    open 출력파일;
    while { end of file } {
        get 라인;
        if { 클래스 정보 } {
            set 주석정보;
            주석정보를 윈도에 출력;
            while { 클래스 정보 } {
              set 주석정보;
              if { 마지막 클래스 주석정보 } {
              클래스 정보 구분자를 윈도에 출력;
              } else {
                 주석 정보를 윈도에 출력;
              }         :# end of if
            }  :# end of while
        }      :# end of if
        if { 멤버함수 정보 }        (:#---- 멤버함수 관련)
        if { 일반함수 정보 }        (:#---- 일반함수 관련)
        if { 단위파일 정보 }        (:#---- 단위파일 관련)
    }  :# end of while
    close 출력파일;
} :# end of procedure
```

(그림 8) 문서화 정보 추출 알고리즘

(Fig. 8) Algorithm for extraction of documentation information

```
#-------------------------------------------------#
#문서화 정보에 대한 데이터베이스(클래스) 생성 루틴
#-------------------------------------------------#
proc MakeCommentInfo {} {
전역변수 선언 루틴;
#------------------ 클래스 -------------------#
지역변수 선언 및 초기화;
클래스관련 소프트웨어 문서화 정보 지정;
클래스 윈도우의 전체라인 구함;
while { 전체라인보다 클 때까지 } {
  set 클래스 관련 정보 저장영역의 초기화;
  get 소프트웨어 정보 플래그 추출;
  관련정보의 첫 번째 항목으로 클래스명을 지장;
  while { 클래스 구분자 } {
    incr 1;
    get 소프트웨어 정보 추출;
    switch 소프트웨어 정보 {      : # 관련 정보를 추출
          Cl_ParentName:      { MakeCl $id 1 $ju }
          Cl_MfName:          { MakeCl $id 2 $ju }
          Cl_DataMemberName:  { MakeCl $id 3 $ju }
          Cl_FriendName:      { MakeCl $id 4 $ju }
          Cl_InstanceName:    { MakeCl $id 5 $ju }
          Cl_ParentAccess:    { MakeCl $id 6 $ju }
          ===                 { break } :# 모듈의 마지막
    }     :# end of switch
  }   :# end of while
  #---------------- 멤버함수 ----------------#
  #---------------- 일반함수 ----------------#
  }   :# end of while
}     :# end of procedure

proc MakeCl {행, 열, 정보} {
  전역변수 선언;
  set "정보"의 길이;
  if { "정보"의 길이 == 2 } {
```

```
        #       "행"과 "열"에 클래스 관련정보 저장
        set cl($ro) [lreplace $cl($ro) $co $co {}]
    } else {
        #       "행"과 "열"에 클래스 관련정보 저장
        set cl($ro) [lreplace $cl($ro) $co $co:
                [lrange $ju 2 end:]
    }       :# end of if
}       :# end of procedure
```

(그림 9) 클래스 관련 데이터베이스 생성 알고리즘
(Fig. 9) Algorithm for database creation of a class

〈표 1〉 각 모듈별 데이터베이스 파일의 내용
〈Table 1〉 Content of database file for each module

클래스 관련 정보	멤버함수 관련 정보	일반함수 관련 정보
. 클래스명 . 부모 클래스명 . 멤버함수명 . 데이터 멤버명 . 프랜드 함수명 . 인스턴스명 . 액세스 모드	. 멤버함수명 . 멤버함수 원형 . 멤버함수 인수 . 반환 정보 . 클래스명 . 다형성 . 액세스 모드 . 메시지 송신자 . 메시지 수신자 . 인스턴스명	. 일반함수명 . 일반함수 원형 . 일반함수 인수 . 반환 정보 . 다형성 . 일반함수 범위 . 메시지 송신자 . 메시지 수신자 . 인스턴스명

〈표 2〉 gun2__mf.data의 내용
〈Table 2〉 Content of gun2__mf.data

...	이전 구성성분의 내용
-----	각 구성성분 사이의 구분자
push	멤버함수명
push(char ch)	멤버함수 원형
char ch	멤버함수 인수
	반환 정보
stack	클래스명
	다형성
public	액세스 모드
	메시지 송신자
	메시지 수신자
	인스턴스 명
-----	각 구성성분 사이의 구분자
...	이후 구성성분의 내용
...	

2) 문서화 정보 플래그 철자 검사기

(그림 10)의 알고리즘에서 알 수 있듯이 이 모듈은 단순히 주석에서 추출한 문서화 정보 플래그를 키워드 목록에 저장되어 있는 내용과 비교하여 그 진위 여부를 결정하는데 한한다. 잘못 작성된 부분만을 제시하고 사용자에게 수정을 요구하게 되는데, 이 기능의 자동화는 현재 연구 중에 있다.

```
#-------------------------------------------------#
#               문서화 정보의 철자 검사 루틴
#-------------------------------------------------#
proc SpellCheck {} {
 전역변수 선언 루틴;
 지역변수 선언 및 초기화;
 키워드목록 선언;

 윈도우의 전체라인 및 현재 라인 번호 구함;

 for { 초기값 = 1; 전체라인보다 클 때까지; incr 1 } {
      get 현재 라인의 소프트웨어 정보 플래그 추출;
      if { 플래그 ∉ 키워드목록 } {
            에러카운트 ++;
            플래그 컬러화;
      }       :# end of if
 }    :# end of for

 다이얼로그 박스 생성;
 if { 에러카운트 != 0 } {
      에러메시지 출력;
 } else {
      메시지 출력;
 }    :# end of if
}     :# end of procedure
```

(그림 10) 문서화 정보 플래그 철자 검사 알고리즘
(Fig. 10) Algorithm for spelling check of documentation information flags

3) 빠뜨린 문서화 정보 검사기

(그림 11)의 알고리즘에서 알 수 있듯이 이 모듈은 3개의 프로시저로 구성되는데, 'MissingFlagCheck' 프로시저는 소프트웨어 문서화 정보 중 개발자가 빠뜨린 정보를 생성하여 각 관련 모듈 윈도에 출력해주는 루틴인 'MissingFlagProc' 프로시저와 윈도의 다이얼로그 박스에서 'Update' 버튼을 선택했을 때 각각의 관련 윈도에 나타난 생략된 정보를 원시 프로그램에 삽입해 주는 루틴인 'UpdateMissingFlag' 프로시저로 구성된다.

```
#-------------------------------------------------#
#       빠뜨린 문서화 정보의 탐색 및 반자동 삽입 모듈
#-------------------------------------------------#
proc MissingFlagCheck {} {
      각 노들에 대한 생략 불가능한 정보 변수 생성;
```

```
        각 모듈에 대한 생략 가능한 정보 변수 생성;
        # 각 모듈별로 실행
        set insertline { MissingFlagProc 모듈첨두사 관련 윈도명
                         생략불가정보, 생략가능정보 }
      각 모듈에서 발견된 생략정보의 수를 구함;
        다이얼로그 박스 생성;
        if { 생략된 정보의 수 > 0 } {
           switch {
               "Update"        :# 각 모듈별로 실행
                 {UpdateMissingFlag 관련 윈도명 식별자}
               "Exit"  { exit }
           }  ;# end of switch
        }       :# end of if
}       :# end of procedure
proc MissingFlagProc {} {
        전역변수 선언 루틴;
        지역변수 선언 및 초기화;
        윈도우의 전체라인 구함;
        for { 초기값 = 1; 전체라인보다 클 때까지;  incr 1 } {
                get 현재 라인의 소프트웨어 정보 플래그 추출;
                if { 생략가능 정보 || 비관련 정보 || NULL }
                        { continue; }
                for { 생략 가능한 정보보다 클 때까지 } {
                        if { string compare 추출정보 생략불가 정보 }
                        {  생략된 플래그 삽입위치 지정;
                           break;
                        }       :# end of if
                }       :# end of for
                for { 다른 모듈을 만날 때까지 } {
                    생략된 정보에 대한 정보생성;
                    각 관련 윈도에 삽입;
                    삽입된 라인수 ++;
                }       :# end of for
                다음 모듈의 검사를 위해 관련 변수 초기화;
        }       :# end of for
        return 삽입된 라인수;
}       :# end of procedure
proc UpdateMissingFlag { 관련윈도 구분자 }
        전역변수 선언;
        switch 구분자 {
                각 모듈의 시작 위치를 구함;
                모듈별로 반복실행;
        }       :# end of switch
        전체라인수 구함;
        while { 원시코드출력창의 라인보다 클 때까지 } {
                각 모듈 정보창의 현재라인 구함;
                원시코드 정보창의 현재라인 구함;
                if { 원시코드 정보창의 라인이 비주석 라인이면 } {
                        모듈 정보창의 라인포인터 --;
                        continue;
                }       :# end of if
                if { 각 모듈 정보창의 라인이 비주석 라인이면 } {
                        원시코드 정보창의 라인포인터 --;
                        continue;
                }       :# end of if
                while { 모듈 라인이 원시코드 라인보다 클 때까지 } {
                    ① 각 모듈 정보창의 현재라인 구함;
                    ② 원시코드 정보창의 현재라인 구함;
                    if { ① == ② } {
                       break;
                    } else {
                            모듈 정보창의 라인을 원시코드의
                                    삽입포인터에 삽입;
                            모듈 정보창의 라인포인터 ++;
                            원시코드 정보창의 라인포인터 ++;
                        }       :# end of if
                }       :# end of while
        }       :# end if while
}       :# end if procedure
```

(그림 11) 문서화 정보 플래그 조사 알고리즘

(Fig. 11) Algorithm for check of documentation information flags

4) 원시코드/문서화 정보 비교기

원시코드의 분석 정보와 문서화 정보의 일치성을 검사하는 기능을 갖는다. (그림 12)는 원시코드와 문서화 정보간의 일치성 비교를 위한 알고리즘을 보여주고 있다.

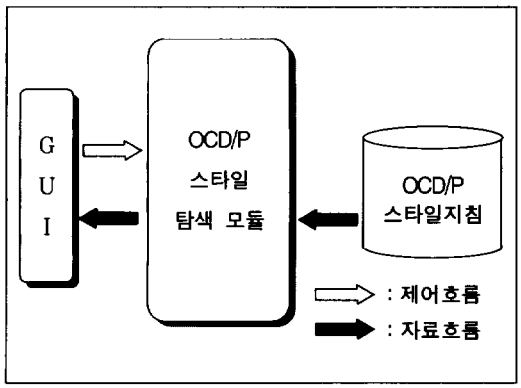
```
#-------------------------------------------------------#
# 원시 코드 분석 정보와 문서화 정보의 일치성을 검사하는 루틴.
#-------------------------------------------------------#
proc CompareInfo {구분번호 종류} {
    전역변수 선언;
    if { 종류 == 1 } {      :# 클래스에 대한 정보
        지역변수 선언 및 초기화;
        open 클래스 데이터베이스;
        파일 포인터 이동;
        while { 레코드 } {
            get line;
            비교 데이터 생성;
        }       :# end of while
        close 클래스데이터베이스;
    } elseif { 종류 == 2 } { :# 멤버함수에 대한 정보
    } else { 종류 == 3 } {  :# 일반함수에 대한 부분
    }
    while { 레코드의 수만큼 } {
        ① 소프트웨어 정보 데이터 지정;
        ② 원시 코드 정보 데이터 지정;
        if { ① == NULL && ② == NULL } {
            put "Unsubscribe";
        } elseif { ① == NULL || ② == NULL } {
            put "**";
        } elseif { ① == ② } {
            put "correct";
        } else {
            put "Mismatch";
        }       :# end of if
        display 관련 정보;
    }       :# end of while
}       :# end of procedure
```

(그림 12) 원시 코드/문서화 정보 비교 알고리즘

(Fig. 12) Algorithm for comparison of source code to documentation information

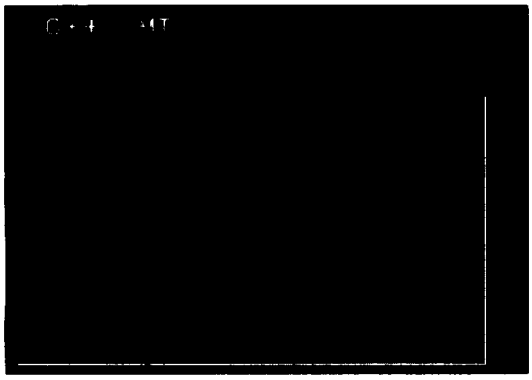
4.2 설계 및 프로그래밍 스타일 지침 지원 도구(OOD/P-GT)

설계 및 프로그래밍 스타일 지침 지원 도구는 (그림 13)과 같이 OOD/P 스타일 탐색 모듈과 지침들을 보관하고 있는 OOD/P 스타일 지침 파일로 구성되어 있다.



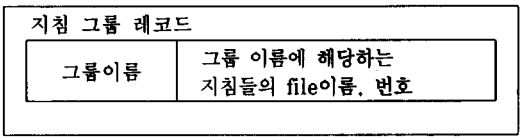
(그림 13) OOD/P-GT의 구조
(Fig. 13) Structure of OOD/P-GT

OOD/P 스타일 탐색 모듈은 OOD/P 스타일 지침 파일을 탐색하여 사용자가 원하는 그룹의 지침들을 (그림 14)와 같이 보여준다.



(그림 14) G1.1 그룹의 지침
(Fig. 14) Guidelines of G1.1 group

OOD/P 스타일 지침 파일에는 여러 관점 별로 분류된 지침들이 서로의 관계성을 고려하여 (그림 15)와 같이 저장되어 있다. 파일의 레코드는 크게 그룹 이름을 저장하기 위한 필드와 그룹 이름에 해당하는 지침들의 파일이름(번호)을 저장하기 위한 필드로 구성된다. 예를 들어 G1.1 그룹의 경우 그룹 이름 필드에는 'Essential' 이라는 문자열이 저장되며, 다른 하나의 필드에는 {2, 5, 6, 10, ...}이 저장된다. 따라서 파일은 각 그룹별로 지침 레코드가 생성되므로 14개의 레코드로 구성된다.



(그림 15) OOD/P 스타일 지침 자료구조
(Fig. 15) Data structure for OOD/P style guidelines

4.3 도구의 평가

기존의 프로그램 문서화 관련 연구와 본 논문에서 연구된 DT의 기능을 비교한 결과는 〈표 3〉과 같다.

〈표 3〉 관련 연구와 DT의 기능 비교
〈Table 3〉 Comparison of related studies to DT

관점 \ 연구	Maro-vac[6]	송후봉 [18]	Object-Manual[19]	DT
정보표현 범위	프로그램의 모든 성분		클래스	프로그램의 모든 성분
대상언어	절차 지향	C++	C++	C++
문서화 지원 기능	×	×	반자동	반자동
정보일치성 비교 방법	×	변수의 수 비교	×	문서화 정보/ 원시코드 분석 정보

〈표 3〉에서 알 수 있는 것처럼 Marovac[6]은 절차 지향 언어의 문서화 방법만을 연구하였고, 송후봉[18]은 프로그램 문서화의 업무 부담을 줄이기 위한 문서화 지원 기능이 없고, 또한 원시 프로그램과 문서화 정보의 일치성을 검사할 때 모듈에 대해서 추출된 변수의 수와 개발자가 주석의 형태로 제시한 해당 모듈의 수가 일치하는가의 여부에 따라 일치성을 결정하기 때문에 모듈의 이름이나 상속 정보 등과 같은 다른 정보가 변경되면 일치성 비교가 불가능하다. ObjectManual[19]은 문서화에 필요한 정보를 원시 프로그램의 컴파일 시에 얻기 때문에 문서화 정보가 정확하여 원시코드와 문서화 정보의 일치성 검사가 필요 없는 반면, 클래스만을 문서화 대상으로 하기 때문에 문서화 정보가 빈약하다.

본 논문의 DT는 객체지향 프로그램의 모든 성분을 문서화 대상으로 하여 프로그램 문서화의 비효율성을 없애고 장점을 살리기 위해 반자동으로 문서화를 지원

하는 기능을 갖추고 있으며, 원시코드와 문서화 정보간의 일관성을 유지하기 위해 서로를 비교하는 기능을 갖추고 있다. 따라서 DT는 다른 연구에 비해 더욱 효율적인 문서화 지원 도구라 할 수 있다.

또한 본 논문에서는 다른 도구에서 제공하지 않는 객체지향 설계 및 프로그래밍 스타일 지침 안내 도구를 제공함으로써 설계 및 프로그래밍 시에 참조하도록 하였다.

5. 결 론

본 논문에서는 객체지향 프로그램의 유지보수 지원을 위한 역공학 도구의 한계와 주석 표현의 단점을 극복하기 위해 C++를 기반으로 개발한 기존의 역공학 도구를 확장하였다. C++-MT라고 명명된 이 도구는 문헌 [3]의 결과인 복잡도 측정 도구(CT: Complexity measuring Tool)와 시각화 도구(VT: Visualization Tool)에 본 논문에서 연구한 프로그램 문서화 지원 도구(DT: Documentation Tool)와 객체지향 설계 및 프로그래밍 스타일 지침 지원 도구(OOD/P-GT : OOD /OOP Guidelines Tool)가 추가된 4개의 서브모듈로 구성된다. DT는 확장된 도구의 주요 서브모듈인데, 이 모듈을 구현하기 위해 본 논문에서는 C++ 프로그램 문서화 방법을 연구·제안하였다.

DT는 제안된 문서화 방안을 바탕으로 프로그램 문서화가 잘 되었는지를 조사하여 오류를 찾아주고, 빠뜨린 문서화 정보를 보여주며, 필요에 따라서 빠뜨린 부분을 자동 삽입해 주어 프로그램 문서화가 잘 이루어질 수 있도록 해준다. 본 논문에서 제안한 문서화 정보들은 프로그램 이해, 변경, 테스팅 그리고 재사용 성분 추출을 용이하게 한다. OOD/P-GT는 이용하기 편리하도록 다양한 관점에서 분석하고 분류된 기존 연구들에서 제안한 설계 및 프로그래밍 스타일 지침들을 사용자가 원하는 그룹별로 손쉽게 찾아볼 수 있도록 해준다.

향후 연구과제는 DT에 문서화 정보 플래그 철자의 오류를 자동으로 수정해 주는 기능과 OOD/P-GT에 선정된 프로그램의 지침 준수 여부를 체크하여 그 결과를 보여주는 기능 등을 추가하는 것이다.

참 고 문 헌

[1] M. Lejter, S. Meyers, and S. P. Reiss, "Support for Maintaining Object-Oriented Programs," *IEEE Transactions on Software Engineering*, Vol.18, No.12, pp.1045-1052, Dec. 1992.

[2] N. Wilde and R. Huitt, "Maintenance Support for Object-Oriented Programs," *IEEE Transactions on Software Engineering*, Vol. 18, No.12, pp.1038-1044, Dec. 1992.

[3] 문양선, 김재웅, 조혜경, 유철중, 김용성, 장옥배, "C++ 프로그램의 이해도 증진을 위한 역공학 시각화 도구", 정보과학회 논문지(C), 제 1권 제 2호, pp.160-171, 1995.

[4] P. G. Zvegintzov N., *Techniques of Program and System Maintenance, Second E야tion*, QED Information Sciences, Inc., 1988.

[5] Barry W. Boehm, "Software Engineering," *IEEE Transactions on Computers*, Vol.C-25, No.12, Dec. 1976.

[6] Nenad Marovac, "Guidelines for Embedded Software Documentation," *ACM SIGSOFT, Software Engineering Notes*, Vol.19, No.2, pp.22-28, Apr. 1994.

[7] 유철중, "객체지향 프로그램의 주요 메카니즘을 이용한 정량적 복잡도 측정 모델", 전북대학교, 박사학위논문, 1994.

[8] T. Cargill, *C++ Programming Style*, Addison-Wesley Publishing Company, Inc., 1992.

[9] C. Horstmann, *Mastering Object-Oriented Design in C++*, John Wiley & Sons, Inc., 1995.

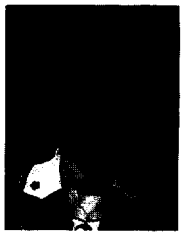
[10] FN/Mats Henricson and Erik Nyquist, "Programming in C++ Rules and Recommendations," Ellemtelecommunication Systems Laboratories, 1993.

[11] K. J. Lieberherr and I. M. Holland, "Assing Good Style for Object-Oriented Programs," *IEEE Software*, pp.38-43, Nov. 1989.

[12] S. Meyer, *Effective C++*, Addisons-Wesley Publishing Company, Inc., 1992.

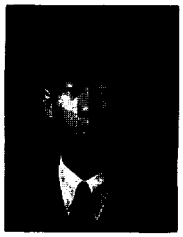
[13] G. Perry, *Teach Yourself Object-Oriented Programming with Turbo C++ in 21 Days*, Prentice Hall Inc., 1993.

[14] J. Sametiger, "A Tool for the Maintenance of C++ Programs," *IEEE Software*, pp.54-60, 1990.
[15] 김문희, 한재수, "객체지향 언어 C++를 위한 Information Viewer", 정보과학회지, 제 11권 제 2호, 1993. 4.
[16] Ian Sommerville, *Software Engineering, Fifth Edition*, Addison-Wesley Publishing Company, Inc., 1996.
[17] T. A. Corbi, "Program Understanding : Challenge for the 1990s," *IBM System Journal*, Vol.28, No.2, pp.294-306, 1989.
[18] 송후봉, 최종명, 유재우, "정형화된 주석을 이용한 프로그램 이해 도구의 설계", '95 봄 학술발표논문집, 한국정보과학회, 제 22권 1호, pp.707-710, 1995.
[19] ObjectSoftware, Inc., "ObjectManual™ - A Tool to generate documentation for C++ Classes in HTML format," "ftp://ftp.netcom.com/pup/ob/objsoft/htmlifo/Testing.html," Jan. 1996.



문 양 선

1986년 2월 전북대학교 전산통계학과 졸업(이학사)
1992년 2월 전북대학교 대학원 전산통계학과 졸업(이학석사)
1996년 전북대학교 대학원 전산통계학과 박사과정 수료
1995년~현재 서해대학 사무자동화과 조교수
관심분야 : 객체지향 소프트웨어 테스팅, 소프트웨어 품질평가 등임

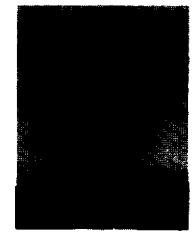


장 근 실

1995년 2월 전북산업대학교 전자계산학과 졸업(이학사)
1997년 2월 전북대학교 대학원 전산통계학과 졸업(이학석사)
1997년 3월~현재 전북대학교 대학원 전산통계학과 박사과정
1996년 9월~1997년 3월 전북대학교 전자계산소 조교
1997년 3월~현재 광양대학 전자계산과 전임강사
관심분야 : 소프트웨어 문서화, 품질평가, Rapid Prototyping 등임

유 철 중

1982년 2월 전북대학교 전산통계학과 졸업(이학사)
1985년 8월 전남대학교 대학원 계산통계학과 졸업(이학석사)
1994년 8월 전북대학교 대학원 전자계산학과 졸업(이학박사)
1982년 9월~1985년 3월 전북대학교 전자계산소 조교
1985년 4월~1996년 12월 전주기전여자대학 전자계산과 근무
1997년 1월~현재 전북대학교 자연과학대학 컴퓨터과학과 전임강사
관심분야 : OOSE, Multimedia, HCI, Distributed Objects Computing Environment, Cognitive Science 등임.



장 옥 배

1966년 고려대학교 수학과 졸업(학사)
1974년~1980년 죠지아주립대, 오하이오주립대 박사과정 수료
1990년~1991년 영국 에딘버러대학교 객원교수
1980~현재 전북대학교 컴퓨터과학과 교수
관심분야 : 소프트웨어공학, 전산교육, 수치해석, 인공지능등