[illegible]

소프트웨어 역공학에서 기능성 검증 기법

황 선 명† · 진 영 택††

요 약

기존의 절차중심 패러다임으로 개발된 시스템들은 사용자들의 새로운 요구와 변화 그리고 개발 당시의 기술보다 뛰어난 새로운 기술이 개발됨에 따라 수정되어야할 필요가 있다. 특히 재공학 및 역공학 기술로부터 품질이 높은 기존 시스템이 객체지향 시스템으로 변환되었을 때 변화된 시스템의 기능이 변환이전의 시스템과 동일한지 여부의 평가가 필요하다.

본 논문에서는 함수 커버리지를 제안하여 절차 중심의 소프트웨어와 변화된 객체지향 소프트웨어간의 기능적 일치성을 검증하기 위한 방법과 도구를 개발하였다. 이때 동적 분석에 필요한 계측도구의 삽입과정과 검증절차를 통하여 함수 커버리지의 만족 여부를 판단하여 테스트 데이터를 선정하였다.

A Functional Verification Techniques in Software Reverse Engineering

Sun-Myung Hwang† · Young-Taek Jin††

ABSTRACT

Existing Systems need to be modified due to the change of environment, the emergence of new technology and the requirement of change.

One of the features of system transformation is how the functions of the old system is preserved in the transformed system after reverse-engineering. But, the techniques to evaluate and verify the transformed system have proceeded more partially and fragmentarily.

In this paper, we have proposed the functional coverage and developed the tool and techniques for verifying functional equivalence between a procedural software and a transformed object-oriented software. The tool and methods are efficiently used to verify whether the transformed program preserves the same functionality as the existing program.

1. 서 론

소프트웨어 역공학은 시스템 요소들과 그들 사이의 관계를 명시하고 높은 수준의 추상화 및 다른 형태로 표현하기 위해 시스템을 분석하는 과정으로 정의되며 이를 위한 여러 가지 방법이 제시되고 있다[1]. 객체지향 개념은 재사용 및 유지보수성을 증가 시키기 때문에 기존 절차적 시스템을 객체 지향 패러다임으로 변환하는데 이러한 역공학 기법이 적용되고 있다. 역공학 과정의 촛점은 객체 및 서비스의 인식이며 이를 위한 다양한 기법이 제시되고 있다[2,3,4].

이 기법을 크게 네범주로 분류해 보면 첫째, 수작업으로 기존 시스템을 재작성하는 방법이 있다. 이 기법은 시스템을 번역하고 시스템 구조를 변경하는데 따른 융통성은 있으나 원래 시스템의 스타일을 그대로 유지

† 종신회원 : 대전대학교 컴퓨터공학과 교수
†† 정 회 원 : 대전산업대학교 전자계산학과 교수
논문접수 : 1998년 1월 15일, 심사완료 : 1998년 7월 21일

라고 오류가 발생할 가능성이 높다. 둘째, 소스언어로 작성된 소프트웨어를 분석하여서 목표 언어로 새로운 소스코드를 생성하는 도구를 이용하는 방법으로 이 경우 언어 번역자체에만 치중하고 프로그램 구조 변경 문제는 다루지 못한다. 셋째, 시스템을 재설계하고 재구현하는 방법으로서 목표 언어로 구현을 위해 현 시스템을 재설계하기 때문에 그 언어의 기능은 잘 사용하게 되나 초기 비용이 높고 적절한 요구사항이 없어서 재설계가 가능하지 않는 경우가 존재한다. 넷째, 시스템을 완전히 재개발하는 경우가 있으나 기존 시스템의 수가 너무 많고 부정적인 결과를 가져올수도 있으므로 시스템을 교체 또는 재개발하는 것에 대한 대안으로서 종래의 시스템을 객체 지향 구조로 변경을 시도하게 된다[5]. 역공학 과정에서 소스코드로부터 추출 가능한 정보와 비구조적인 유지보수 작업에 의한 변경 정보가 포함되어 있는 경우에는 문제점이 발생한다. 따라서 자동화된 역공학 방법과 영역 전문가로부터 필요한 지식을 통합한 혼합 방법이 적절한 해결책으로 제시되고 있다[6,7].

그러나 변환 과정은 위에서 제시된 방법을 적용하는 과정에서 시스템 요소사이의 다양성 , 객체로 대응되지 않는 부분의 처리, 좋은 역공학 지원 도구의 유용성, 자동화 여부등 많은 요인에 따라 변환 결과가 달라지며 이 과정에서 기존 절차적 시스템이 가지고 있는 기능이 누락될 수 있다. 기존의 테스팅 기법[8,9,10,11,12,13]은 순공학(forward engineering)이나 역공학을 통해서 작성된 객체 지향 소프트웨어를 테스팅 하기 위한 방법이며 테스팅의 대상은 클래스,클래스 사이의 관계,메소드 및 객체 상태를 기반으로 테스팅을 수행하고 있다. 그러나 다른 패러다임으로 작성된 소프트웨어들에 대한 기능성 보존의 검증 연구는 거의 없는 실정이다[14]. 이러한 기능성 검증은 그 수준이 소스 코드 ,설계 문서, 정형화된 명세서인지 간에 이루어져야 하며 따라서 변환 과정에서 기능의 누락 또는 삭제를 방지할 수 있다.

본 논문은 절차적 프로그램인 C 프로그램을 객체 지향 프로그램인 C++로 변환 했을 때 변환된 C++ 프로그램이 기존 C 프로그램의 기능을 보존하는가를 소스 코드 수준에서 검증하기 위해 함수 커버리지를 제안하고 이를 바탕으로 기능성 검증을 위한 자동화된 도구를 구현하였다. 이 도구는 테스트 데이터로부터 수행되는 함수를 추적하는 기능과 기능의 불일치 시에 테스터에게 도움을 주는 정보를 제공한다.

2. 관련 연구

2.1 역공학 관련 연구 및 검증 도구

최근 각종 객체지향 방법론 연구가 진행되면서 기존 시스템의 변환에 관심이 모아지고 있는데, Jacobson은 기존 시스템을 수동으로, 그리고 점차적으로 객체지향 시스템으로 재구성하는 방법을 제안하였으며, Feldman은 Fortran 프로그램을 자동으로 C나 C++로 변환해 주는 방법을 제안하였다. 이 방법은 객체를 인식하거나 생성할 수 없으며, 따라서 객체지향 코드를 생성할 수 없다. 자료 추상화 장치(Data Abstraction Facility)를 제공할 수 있도록 포트란 언어를 확장하는 방법이 Miler에 의해 시도된 바 있으며, Dietrich는 기존 시스템들을 객체지향 시스템으로 변환하기 위해 기존 시스템들 위에 객체지향 인터페이스를 연결시켜 기존 시스템을 객체지향 시스템으로 변환하는 시도가 있었다. Waters는 코드 추상화를 통해 변환 과정을 시도하였다.

이와 같이 기존의 시스템에서 객체지향 시스템으로 변환시키는 재공학·역공학적 접근 방법에서 가장 중요시되는 부분은 기존 시스템과 변환된 시스템의 기능적, 품질적 적합성 여부를 판단하는 것으로 기능적, 품질적 적합성을 검증 받지 못한 변환 시스템은 그 가치를 인정받지 못한다. <표 1>은 과거의 외국에서 연구되었던 소프트웨어 테스팅 방법이나 도구로 프로그래밍 언어별로 각각 별도의 방법 및 도구가 개발되었는데, 주로 테스팅, 디버깅 및 유지보수의 CASE 도구로서의 품질 향상을 목표로 하고 있다.

이외에도 개발된 역공학 도구로는 COBOL 코드를 대상으로 한 Intersolv Cor.의 Design Recovery, Scan/COBOL과 C 코드를 입력으로 하는 Teamwork/rev, CIAS, Fortran 코드를 입력으로 하는 VIFOR, EPOS/RE-SPEC 등이 프로그램으로부터 설계정보 또는 그래픽 정보등을 생성한다.

따라서 이제까지의 연구는 단일 언어를 지원하는 도구나 방법론이 추가되었으나 본 논문에서와 같이 source 프로그램과 target 프로그램을 모두 검증하기 위한 방법이나 도구가 재공학적 관점에서 반드시 필요하게 되었다.

〈표 1〉 테스팅 관련 도구 사례
〈Table 1〉 Testing Tools and Coverage

[illegible]	특 징	대상 언어
AdaTest	branch, statement, exception, boolean operator, boolean operand coverage	Ada
MaCabe Visual Testing toolSet	branch, statement, path, condition/decision coverage	C, C++
CodeTEST	branch, statement coverage	C, C++
CoverTEST	statement coverage	C++
Object Coverage	branch coverage	
QC Coverage	statement, path, function call, exception	C, C++
TestCenter	function statement coverage	C, C++
PiSCES Java Tracker	function, branch	Java
PiSCES Coverage Tracker	function, branch, condition/decision, multiple condition coverage	C, C++
STW/ COVERAGE	branch, call-pair	C, C++, Ada, F77
Proteum	mutation anlysis	C
ATAC	function, block, decision, c-uses, all-uses	C

2.2 기능성 검증을 위한 함수 커버리지

테스팅 기법의 종류를 보면 기능의 누락, 기능의 비정상을 검출해 내는 기능 테스팅과 프로그램의 논리구조상의 오류를 발견하는 구조 테스팅으로 구분된다. 기능 테스팅의 대상은 요구 정의서, 문제가 기술되어 있는 명세서, 설계명세 등 추상화 단계의 자료이며, 구조 테스팅의 대상은 프로그램이다.

본 논문에서는 변환 이전의 C 프로그램과 변환 이후의 C++ 프로그램의 적합성 여부를 기능의 일치성을 통하여 검증하고자 한다. 그러므로 C 소스 프로그램은 테스팅 도구의 입력으로 나타나는데 이는 구조 테스팅이라고 할 수 있다. 그러나 C 코드와 C++ 코드의 기능성을 분석하는 일은 기능적 테스팅의 작업영역이다.

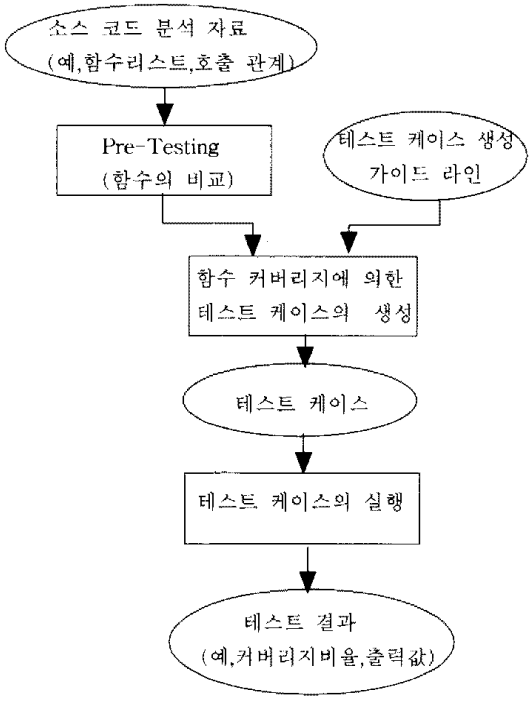
따라서 기능성 검증은 구조 테스팅적인 코드를 기반으로하여 C 코드와 C++ 코드간 기능 테스팅에 해당하는 기능 일치성을 찾기 위한 정보를 제공한다.

이러한 구조 테스팅에서 테스트 케이스를 생성하는 방법은 크게 3가지 유형으로 구분할 수 있으며 이것은 각각 메트릭 기반 테스팅(Metric-based testing), 커버리지 기반 테스팅(Coverage - based testing),복잡도 기반 테스팅(Complexity-based testing)[15]이다. 이들 중 가장 일반적인 방법은 테스트 데이터에 의해 실행되는 프로그램내의 커버되는 문(statement), 분기(branch), 경로(path)를 기초로하여 데이터 집합을 구하는 커버리지 기반 테스팅이다. 그러나 기능성 검증에서 적용되는 기능 요소는 문이나 분기 또는 경로와 관계가 깊지 않기 때문에 새로운 기준인 함수 커버리지(function coverage)를 제안한다.

(정의) 함수 커버리지(function coverage)

프로그램내에 존재하는 각 함수가 최소한 한번 이상 실행될 수 있도록 테스트 데이터를 설정한다.

코드 분석과정으로 부터 생성된 C프로그램내의 입출력 함수와 사용자 정의 함수들의 리스트로 부터 이들 함수를 최소한 한번 이상 커버(cover)하는 테스트 데이터 집합을 정의한다. 이때 하나의 테스트 데이터에 의해 커버되는 함수를 함수리스트에서 제외시켜 가면서 전체 함수가 모두 실행되도록 입력 변수 값인 테스트 데이터를 추가시켜 간다. 이때 테스트 데이터의 설정값은 테스트 담당자의 역할이며 주어진 커버리지를 최소의 테스트케이스 설정을 위해 가이드 라인이 필요하다.



(그림 1) 기능성 검증 절차
(Fig. 1) Verification Process for Functionality

2.3 기능성 검증 과정

기능성 검증을 위해 본 논문에서 고려하는 요인은 첫째, 기능성 검증 과정은 완전한 자동화 보다는 테스터와의 인터액션을 통하여 이루어진다는 것이며 둘째, C 프로그램은 소스 코드이외에는 관련 정보가 없다고 고려한다. 셋째, 함수 커버리지를 중심으로 테스팅을 수행하는 과정에서, C와 역공학하여 생성된 C++의 함수명 및 변수이름과 같은 이름 명명(naming) 문제이다.

기능의 누락 여부를 검증하기 위해 본 논문은 기능 테스팅 기법을 채택하며 (그림 1)에서 제시된 바와 같이 세 단계의 테스팅 절차를 수행한다.

사전 테스팅(pre-testing) 단계는 변환이전의 C 프로그램과 역공학으로 생성된 C++ 프로그램 사이의 함수 갯수를 비교하여 테스팅 할 함수를 선정한다. 입력 프로그램(C, C++)으로부터 추출된 함수 중 사용자 정의 함수를 비교한 후 테스트 할 함수를 결정하기 위한 것으로서 다음과 같은 단계에 의해 수행된다.

사전조건 : C와 C++ 프로그램의 함수 이름은 동일

단계1) 함수의 이름과 함수의 갯수를 비교한다.

단계2) C의 함수 갯수가 C++ 함수 갯수보다 많은 경우 나머지 C 함수를 나열하고 C++의 폴리몰피즘등 과 같은 특성에의한 함수 갯수의 감소를 검사한다.

단계3) C의 함수 갯수가 C++ 함수 갯수보다 적은 경우 여분의 C++ 함수를 나열하고 기능의 추가 및 언어 특성에 의한 함수의 증가(생성자 함수등)를 검사하고 테스트 할 함수 커버리지에서 이 함수들을 제외한다.

사후조건 : C와 C++ 프로그램의 함수 갯수는 동일

사전 테스팅 단계는 테스트 케이스에 의한 함수 커버리지의 비교를 위해 필요하며 함수의 이름이 동일하게 유지된다고 하더라도 C++에서 제공하는 폴리몰피즘과 같은 다양한 기능에 의해서 함수의 갯수가 감소할 수 있다. 목적은 기존의 C 프로그램으로부터 추출된 함수와 역공학으로 생성된 C++ 함수사이에 대응관계를 설정하기 위한 것으로서 이에 대한 관계를 표로 나타내면 <표 2>과 같다. (단, 함수 갯수는 가정치로 하고 함수이름은 동일한 것으로 가정한다.)

〈표 2〉 함수의 매핑 관계
〈Table 2〉 Mapping Relation

	기존 프로그램 (C)	변환된 프로그램 (C++)	관 계
사용자 정의 함수의 갯수	50	50	함수 커버리지 동일
	50	52	1) 함수의 추가 2) 언어의 특성(생성,소멸자)에 의한 추가
	50	48	1) 함수의 누락 2) 폴리몰피즘과 같은 언어 특성에 의한 함수의 감소

두번째 테스팅 단계는 사전 단계를 거쳐 선정된 함수를 대상으로 전체 함수를 커버하는 테스트 케이스를 선정한다. 코드 분석 과정으로부터 추출된 C프로그램의 함수 중 입력함수의 리스트로부터 전체 함수를 최소한 한 번이상 커버하는 테스트 케이스 집합을 정의한다. 이 과정은 실제 프로그램의 실행을 통해서 이루어지며 입력 받는 데이터 변수의 이름이 동일하다고 가정할 때(실제로는 데이터 변수의 이름이 동일하지 않음) 입력은 시스템에서 제공되는 함수를 이용하게 되며 입력데이타의 타입과 갯수는 C++로 변환되더라도 같게 된다. 테스트 데이터의 의미를 파악하기 위해 입력자료와 연관된 출력 텍스트를 참조한다.

테스트 케이스를 유도하기 위한 지침으로서 테스트 케이스의 생성을 위해 하나의 테스트 데이터에 의해 커버되는 함수를 함수리스트에서 제외 시켜가면서 전체 함수가 모두 실행되도록 입력변수 값인 테스트 데이터를 추가 시켜간다. 이때 테스트 데이터의 설정 값은 테스터의 역할이다. 세번째 테스팅 단계는 두번째 단계에서 선정된 테스트 케이스를 가지고 C및 C++에 적용하여 커버되는 함수의 리스트와 출력값을 비교하여 누락된 기능이 없는가를 검사한다.

커버되지 않은 함수가 없을 경우 두 프로그램의 가능성은 일치하는 것으로 고려될 수 있다.

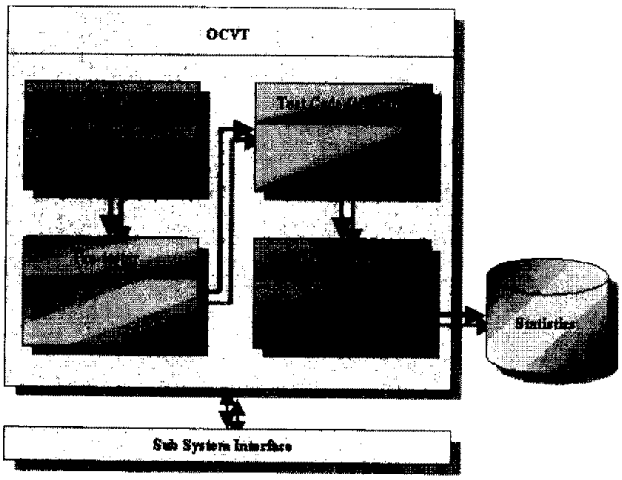
이상과 같은 단계를 통해서 본 논문의 검증 기법과 기존의 기능테스팅 기법과 비교할 때, 두 방법 모두 테스터가 입력데이터 집합에 대한 출력결과를 이미 알 수 있다는 점에서는 동일하나 기존 방법에서는 어떠한

기능을 수행하는지 파악하기 어렵다. 본 검증 기법은 입력데이타에 대한 함수 커버리지 개념을 도입함으로서 어떤 기능이 수행되는지 파악할 수 있다.

3. 기능성 검증 도구의 설계 및 구현

3.1 검증 도구의 구성

(그림 2)에서 제시된 검증 도구는 기존의 C프로그램을 역공학하여 생성된 C++ 프로그램이 변환 이전의 C프로그램이 가지고 있는 기능과 일치하는지를 검증하기 위한 검증 도구이다. 검증 도구는 함수 커버리지를 기반으로 하여 기능의 일치성을 검증하기 위한 정보를 제공한다. 이 도구는 소스코드의 분석을 위한 코드 분석기, 사전 테스트기, 테스트 케이스의 생성을 위한 테스트 케이스 생성기, 테스트 결과를 검증하기 위한 비교기와 통계 및 기타 정보를 제공하기 위한 출력기로 구성된다.



(그림 2) 객체 적합성 검증 도구의 구조
(Fig. 2) System Architecture of Verification for object

코드분석기는 기존 C프로그램과 역공학으로 생성된 C++ 프로그램을 입력으로 받아 각 소스 프로그램에 포함된 함수를 추출하는 기능, 추출된 함수 사이의 호출 관계를 보여주는 기능을 수행한다. 분석기는 입력된 소스프로그램(C, C++)을 토큰별로 구분하여 파싱하고 기능을 나타내는 함수를 찾는다. 소스프로그램의 구성은 C, C++ 모두 다수의 파일과 구현환경에서 제공하는 시스템 함수를 제공하는 헤더 파일로 구성된다. 분석기를 통해 추출된 함수 중 시스템에서 제공하는 함수를 제외한 사용자가 정의한 함수를 (반환값, 함수명, 인자타입1...n, 인자1,...n)의 형태로 표현하여 저장한다. 다수의 인자가 있는 경우는 그 순서대로 인자타입과 인자를 표현한다. 함수 사이의 호출 관계는 다이아그램 형태로 표현된다.

사전 테스트기는 기존 프로그램과 변환 프로그램 사이에 함수 갯수가 동일하지 않은 경우 테스트 할 함수를 조정한다. 테스트 케이스 생성기는 기존의 C 프로그램으로부터 추출된 입력자료를 이용하여 함수 커버리지를 위한 테스트 케이스를 유도한다. 테스트 케이스의 유도는 전체 함수가 커버될 때까지 계속적으로 이루어지고 전체 함수 커버리지가 달성될 때의 테스트 케이스를 생성한다. 테스트 결과 비교기기는 C프로그램의 추출된 테스트 케이스를 이용하여 변환된 C++ 프로그램의 함수 커버리지를 검사하고 그 결과를 비교하는 기능을 제공한다. 출력기는 테스트 케이스에 따라 커버되는 함수리스트 및 출력값을 나타낸다. 이를 위해 계측을 위한 별도의 코드가 원래 코드에 삽입되기 때문에 기존 및 변환된 프로그램의 컴파일과 수행과정이 필요하다.

3.2 함수커버리지 적용을 위한 계측도구 삽입

본 도구에서는 Turbo C환경을 고려했고 검증 도구 자체에서 C의 컴파일 및 수행 환경을 제어할 수 없으므로 데스다가 그 실행 환경에서 직접 컴파일 하는 것으로 하였다. 계측을 위한 소스코드 변경 과정은 코드를 실행시킨 때 입력에 관한 값, 출력에 관한 값, 입력에 따른 실행함수의 값을 출력하기 위해 3개의 파일을 생성시키는 코드를 삽입하는 것으로 시작된다.

- in-buf.txt : 입력에 관한 값
- out-buf.txt : 출력에 관한 값
- func-buf.txt : 입력에 따른 실행 함수값

1) 메인 함수의 계측

전처리기 파일의 정의가 끝나면 전역변수로 3개의 파일 변수를 생성한다. main 함수가 시작되면 main함수의 변수 선언이 끝난후 3개의 파일을 열고 main함수가 끝나기 바로전 열었던 3개의 파일을 닫는다. 이때 3개의 File 변수는 다음과 같다.

- create-file1 → in-buf.txt
- create-file2 → out-buf.txt
- create-file3 → func-buf.txt

2) 입력함수의 계측

입력함수를 만나면 입력함수에 쓰인 입력 변수의 타입과 변수 명, 변수 값을 계측하는 코드를 삽입한다.

예1) scanf("%d", &num);
fput("int num", create_file1);
fprintf(create_file1, "%d", num);
fputc('\n', create_file1);

3) 출력 함수의 계측

출력함수를 만나면 출력함수에 쓰인 출력 변수의 타입과 변수명, 변수 값을 계측하는 코드를 삽입한다.

예2) printf(" %d %s", num, name);
fputs("int num" create_file2);
fputs("char name", create_file2);
fprintf(create_file2," %d %s", num, name);
fputc('\n', create_file2);

4) 사용자 함수의 계측

함수를 정의한 코드의 시작부분, 즉 함수안에서 지역 변수의 정의가 끝나면 바로 함수명 문자열을 계측하는 코드를 삽입하며 그 예는 아래의 코드에서 제시된다.

예3)
```
sortclass (st, nst)
 STUDENT st[];
 int nst;
{
 int i, j, pick;
/****** instrument code begin: *******/
printf("%s\n","sortclass");
fputs("sortclass\n",create_file3);
/****** instrument code end:  ********/
    for (i=0; i < (nst-1); i=i+1) {
        pick = i;
        for (j=i+1; j < nst; j=j+1) {
          if (st[j].grade < st[pick].grade) {
                pick = j;
            }
        }
        swap(&st[i], &st[pick]);
    }
}
```

이러한 계측을 위한 코드의 변경으로 인한 컴파일을 성공적으로 마치면 테스트 케이스의 생성을 위해 기존 프로그램 및 변환된 프로그램을 수행할 수 있다.

3.3 검증 도구의 구현 및 사용자 인터페이스

검증 도구의 프로토타이핑 구현은 윈도우환경에서 Visual C++로 작성되었고 C는 Turbo C로 작성되었다고 가정하였다. 검증 도구는 10개의 클래스로 구성되며 각각은 다음과 같다.

1) CEvnDlg 클래스

파싱할 환경을 설정하는 클래스로 다음과 같은 선택사항이 있다.

(1) 파싱할 파일이 C code 인지 C++ code 인지를 선택한다.
(2) 파싱할 파일인 단일파일인지 프로젝트 파일인지를 선택한다.
  i) 단일파일은 하나의 파일로 구성된다.(예, main.c)
  ii) 프로젝트 파일은 여러개의 파일로 구성된다.(예, main.c exam1.c exam2.c .)
(3) 파싱할 파일을 선택한다.
(4) 시스템 헤더파일이 있는경로를 설정한다.

2) CParser 클래스

이 프로그램의 주된 기능을 하는 클래스로써 다음과 같은 기능이 있다.

(1) 토큰을 얻는다.
(2) 함수를 찾아 함수에 대한 정보를 얻는다.
  (함수 반환값, 함수 명, 함수 인자 타입, 함수 인자 변수 명)
(3) 함수들의 관계를 구한다.
(4) 파싱하는 파일의 코드를 편집한다.

3) CViewDlg 클래스

CPaser에서 구한 함수에 대한 정보를 보여주고 함수들의 관계를 트리형식으로 보여준다.

4) CMatch 클래스

C code와 C++ code를 파싱했을 때 서로 매칭되는 함수를 보여준다.

5) CMatchNo 클래스

C code와 C++ code를 파싱했을 때 서로 매칭되지 않는 함수를 보여준다.

6) CCreateDlg 클래스

C code를 실행시켰을 때 입력값에 대한정보, 출력값에 대한정보, 입력값에 따라 실행한 함수를 보여준다.

7) CKey 클래스

파싱파일이 C code일 때 그 C code의 함수에 대한 정보를 보관하는 클래스

8) CKeyP 클래스

파싱파일이 C++ code일 때 그 C code의 함수에 대한 정보를 보관하는 클래스

9) CInclude 클래스

시스템 헤더 파일이 위치한 곳을 지정하는 클래스이다. 드라이브명과 디렉토리명을 지정한다.

10) CParameter 클래스

함수의 인자값으로 들어가는 인자타입과 변수명을 저장하기 위한 클래스로써 CKey와 CKeyP의 멤버로 들어간다.

11) CCompareDlg 클래스

테스트 케이스에 대해 커버되지 않는 C++ 함수리스트를 생성한다.

12) CStatDlg 클래스

테스트케이스에 대한 C 및 C++ 프로그램을 테스트한 결과값, 함수리스트 등을 출력한다.

검증 도구의 사용자 인터페이스는 테스터가 편리하게 그 과정을 파악할 수 있고 테스팅 전차가 그대로 반영될 수 있도록 설계되었다. 검증 도구의 실행은 크게 세가지 범주로 나눌 수 있는데 첫째는 코드 분석이고 둘째는 테스트 케이스 생성, 셋째는 테스트 결과 비교이다.

코드 분석 부분에서 코드분석기의 수행과정은 (그림 3)의 화면에서 제시된 것처럼 4부분으로 나뉘어진다.

1) Environment

코드 분석의 수행은 이부분을 선택함으로서 시작되며 이 부분이 선택되기 이전에는 다른 부분을 선택할 수 없도록 그레이(gray)로 반전되어 표시된다.

2) Parse

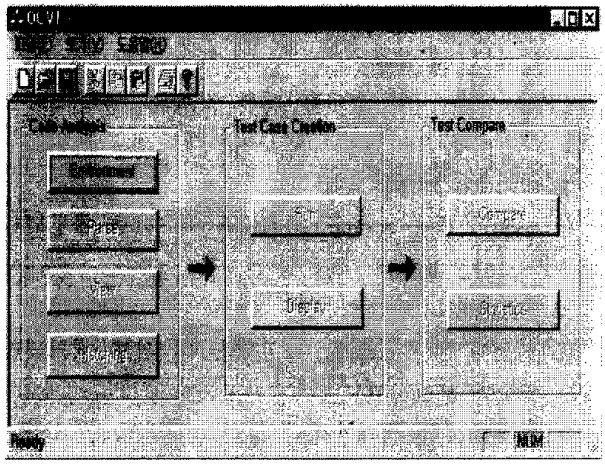
입력에 따른 코드(C, C++)를 토큰별로 구분하여 파싱하고 원하는 토큰에 따른 분석자료를 산출한다. C 및 C++ 코드의 사용자 정의 함수와 입·출력 함수리스트를 찾고 각 변수의 리스트를 산출한다. 분석이 끝났을 때만 "View" 부분을 선택할 수 있다.

3) View

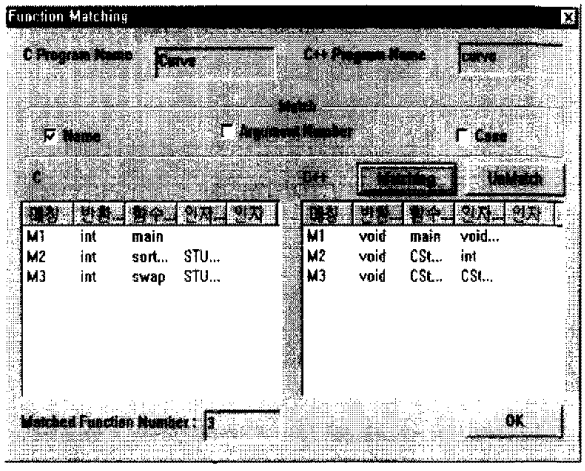
추출된 함수 관련 정보 및 함수 사이의 관계가 표현된다. 상단 부분은 함수에 대한 반환값, 함수명,인자타입,인자에 대한 정보를 제시하며 하단 부분은 함수 사이의 호출 관계를 간략한 형태의 다이아그램으로 표시하고 있다.

4) matching

이 영역은 사전 테스팅 기능을 수행하는 부분으로서 (그림 4)와 같이 나타나며 제시된 내용 중 "Match"부분은 함수를 비교하기 위한 옵션을 주기 위한 것으로서 디폴트는 "함수이름"과 "대소문자의 구별하지 않음"에 의한 비교이다. 사전조건으로서 C의 함수의 이름이 동일한 이름으로 C++로 변환된다고 가정했기 때문에 이와 같이 설정하였다.



(그림 3) 기능성 검증 도구의 초기화면
(Fig. 3) Initial Interface of OCVT



(그림 4) 사전 테스트기
(Fig. 4) Preprocessor for Function Mapping

제시되는 함수의 리스트는 함수의 이름으로 정렬되며 C 및 C++에서 서로 부합되는 함수는 각각 M1,M2.. 등으로 표현된다. 그 의미는 양쪽 첫번째 함수끼리 부합된다는 것을 나타내고 부합되지 않은 함수는 표시없이 나타나므로 그 결과를 쉽게 파악할 수 있다.

5) Run 부분

테스트 케이스의 생성 부분은 Run 및 Display영역으로 나누어 지고 Run부분을 선택하면 테스트 케이스를 생성하기 위해 실제 프로그램이 수행된다. Display 영역에는 실제 테스트 케이스에 대한 입력값, 커버되는 함수 목록및 출력값이 제시되며 다음과 같이 세영역으로 나뉘어진다.

(1) input variable 영역

이 영역은 전체 함수를 커버하는 테스트 케이스를 유도하기 위해 기존 프로그램의 실행을 위한 입력값이 제시된다.

(2) covered function list 영역

이 영역은 테스트 케이스가 수행될 때마다 커버되는 함수의 리스트가 나열된다. 이 과정은 전체 함수가 커버될 때까지 계속된다.

(3) Output variable 영역

이 영역은 테스트 케이스에 따른 출력 변수와 출력 값을 나타낸다.

6) Test Compare

C로 부터 추출된 테스트 케이스를 변환된 C++에 적용한 결과를 파악하기 위해 C++에 대한 계측을 실행한다. Coverage Ratio는 C++ 사용자 정의함수의 갯수를 C 사용자 정의 함수의 갯수로 나눈 백분율을 의미하며 이 값이 100%를 나타내면 변환된 C++의 전체 함수가 커버되었다는 것을 의미한다.

(1) C측의 커버안된 함수 리스트

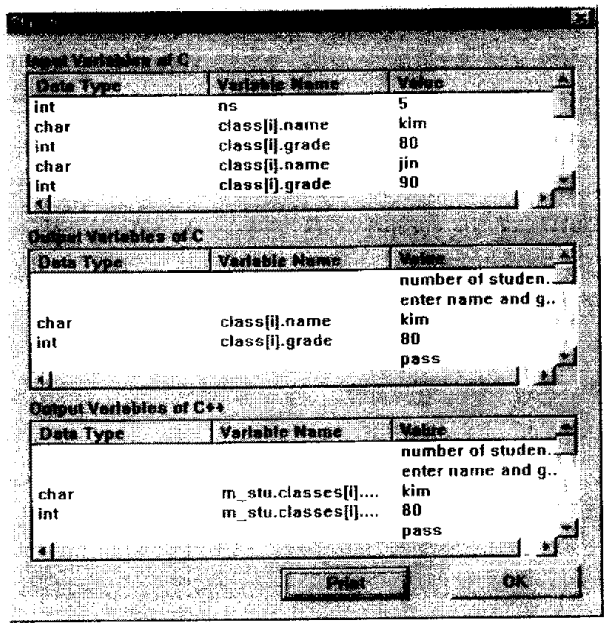
이 부분은 테스트 케이스를 C++에 적용한 결과 C++에 대응되지 않는 C측의 함수를 나타낸다.

(2) C++측의 커버 안된 함수 리스트

이 부분은 테스트 케이스를 C++에 적용한 결과 변환 과정의 다양함(언어의 특성, 기능의 분할, 기능의 추가 등) 때문에 C에 대응되지 않는 C++측의 함수를 나타낸다.

7) Statistics

이 부분은 (그림 6)에 제시된 바와 같이 테스트 케이스의 적용시 함수 커버리지와 관련한 출력값을 나타내고 있다. 양측의 출력 결과를 비교하기 위해 출력값



(그림 5) 테스트 케이스 적용에 따른 출력 결과
(Fig. 5) Comparison of Outputs

을 이용하는데 이때 출력 자료의 유형은 수치 자료(정수, 실수), 문자(열) 자료, 화일 자료(수치자료 + 문자자료), 기타 자료(event ,mouse click, touch panel, control signal등) 이 존재하며 기타 자료 유형은 보통 비트열(bit string)의 형태로 나타나지만 C프로그램의 실행 결과값(Test 결과값 (C))과 C++프로그램 결과값(Test 결과값 (C++))을 서로 비교하여 두 프로그램간의 기능의 일치성을 판단할 수 있다.

4. 실행결과의 고찰

함수 커버리지를 기본으로 한 기능성 검증을 위해 여러 종류의 C 프로그램을 수작업으로 C++ 로 역공학하였다. 역공학 하기 위한 방법은 앞서 서술한 역공학 기법을 적용하였고 변환자는 C 및 C++에 익숙한 점우로 고려하였다. 역공학 과정에서 기존 프로그램의 함수는 변환된 프로그램의 멤버 함수로 대응이 된다. 어떤 기능을 수행하는 것으로 간주하여 사례로 C로 작성된 학생들의 이름과 성적을 받아들여서 학급의 상위 70%에 해당하는 학생은 'pass'가 부여되고 하위 30% 학생에게는 'fail'이 부여되는 성적 처리 프로그램과 자료구조 및 알고리즘의 원리를 파악하기 위한 자료구조 처리프로그램을 역공학하여 기능성을 검증하였다. 실행 결과는 다음과 같이 요약될 수 있다.

1) 경우 1: 두 프로그램이 동일한 함수 커버리지를 가지며 테스트 결과가 같은 경우는 변환된 프로그램이 기존 프로그램의 기능성을 보존하는 것으로 판단되었다.

2) 경우 2: 두 프로그램이 동일한 함수 커버리지를 가지나 테스트 결과가 다른 경우는 실제로 어떠한 기능이 누락되었는지는 파악할 수 없었다. 성적처리 프로그램에서 C에서는 성적순으로 정렬되는 것을 C++에서는 이름순으로 정렬되도록 변경하고 자료처리 학습프로그램의 경우 알고리즘의 로직을 바꾼 결과 함수커버리지에는 영향이 없었으나 결과값은 다르다는 것을 알 수 있었다. 따라서 이경우는 본 검증 도구에서는 출력된 결과를 테스터가 비교함으로써 파악할 수 있었다.

3) 경우 3: 두 프로그램의 함수 커버리지가 다른 경우는 변환된 C++프로그램에서 함수커버리지가 C 프로그램의 함수커버리지보다 적거나 높은 경우이다. 기준이 되는 함수커버리지가 C 의 커버리지이기 때문에 C++ 커버리지가 적은 경우는 변환 과정에서 기능의 누락이 있음을 파악할 수 있었다. 높은 경우는 출력값을 비교하여 결과가 동일한 경우 기능성이 보존됨을 알 수 있었다.

5. 결론 및 제안

절차중심 프로그램과 객체지향 프로그램간의 기능의 비교는 패러다임의 커다란 차이점때문에 쉽지 않다. 또한 역공학 과정에서 절차중심 프로그램에서의 단일 모듈이나 프로시져가 객체의 단일 서비스로 대응되지 않기때문에 단편적인 비교는 타당성을 고려해야 한다.

본 논문에서 설계하고 구현한 기능 적합성 검증 도구는 주로 기능의 일치성을 판단하는 차원에서는 기능 테스팅이며 테스팅의 대상이 소스코드라는 점에서는 구조 테스팅이라고 볼 수 있다. 프로그램 코드에서 기능 테스팅을 위하여 설정한 함수 커버리지는 본 연구에서 테스트케이스 생성의 기준이며 이를 통하여 기능 관련 정보를 코드로부터 산출할 수 있다. 또한 테스트 드라이버에 의해 수행된 두 코드의 결과를 비교할 때 해당되는 데이터에 의해 수행되는 함수를 추적하는 트레이스기능으로부터 결과 불일치시에 수정되어야할 영역에 대한 정보를 제공한다.

본 검증 시스템은 RESORT(REsearch on object oriented SOftware Reengineering Technology)의 마지막 과정이자 RESORT 시스템의 결과에 대한 검증과정으로 볼 수 있다.

적합성 검증 도구에 의한 검증은 완전한 자동화 처리가 불가능하며 사용자의 도움이 요구된다. 본 도구의 설계시 추후 고려해야 하는 사항은 최소의 테스트 케이스 생성 지침, 비교 결과의 판단에 대한 사용자의 지원, 트레이스 기능으로부터 오류 영역 탐색 이다. 아울러 기존 프로그램의 실행 환경과 변환된 프로그램의 실행환경이 다른 경우 검증 도구를 포함한 시스템 사이의 통합이 고려되어야 한다.

참 고 문 헌

[1] E. J. Chikofsky and J. H. Cross, "Reverse Engineering and Design recovery: A Taxonomy", IEEE software, 7(1), pp.13-17, 1990.

[2] Eric J. Byrne, "Software Reverse Engineering: A Case Study",software practice and experience, Vol.21(12), pp.1349-1364, 1992.

[3] T. J. Biggerstaff, "Design Recovery for Maintenance and Reuse", IEEE Computer, pp.36-49, 1989.

[4] I.Jacobson and F. Lindstrom,"Re-engineering of Old systems to an Object-oriented Architecture", OOPSLA'91 Proceedings, pp.340-350, 1991.

[5] Harald C. Gall and Rene R. Klosch and Roland T. Mittermeir, "Architectural Transformation of Legacy systems", ICSE-17 Workshop on Program Transformation for Software Evolution, Technical Report CS pp.95-418, 1995.

[6] Harald Gall and Rene Klosch, "Managing Uncertainty in an Object Recovery Process", 5th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems, 1994.

[7] 진영택, 황선명, 이헌기 "절차중심 소프트웨어에 대한 객체 지향 구조로의 변환기법", 97춘계학술발표대회 논문집, 한국정보처리학회, 1997.

[8] Scott Ambler, "Use-Case Scenario Testing", Software Development, Vol.3, No.6, pp.53-61, 1995.

[9] C. D. Turner and D. J. Robson, "Guidence for the Testing of Object oriented Programs", Technical Report No.TR 2/93, University of Durham.

[10] D. C. Kung, Jerry Gao and Pei Hsia et al, "Design Recovery for Software Testing of Object-Oriented Programs",Proceedings of the Working Conference on Reverse Engineering, pp.202-211, 1993.

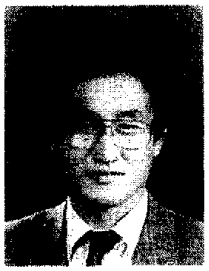
[11] Shekhar Kirani and W.T. Tsai, "Method sequence specification and verification of classes", JOOP, pp.28-38.

[12] John D. McGregor and Douglas M. Dyer, "Selecting Functional Test Cases for a Class", 11th Annual Pacific Northwest Software Quality Conference, PNQSC, pp.109-121, 1993.

[13] Ernst Siepmann and A. Richard Newton, "TOBAC: A Test Case Browser for Testing Object-Oriented Software", Proceedings of the 1994 International Symposium on Software Testing and analysis, pp.154-168, 1994.

[14] 황선명, 진영택, 고천재외, "기능성 검증을 위한 OCVT의 설계", 97추계학술발표대회 논문집, 한국정보과학회, 1997.

[15] Martin R. Woodward, David Hedley, and Michael A. Hennell, "Experience with path analysis and testing of programs", IEEE TOSE, pp.278-286, 1980.



황 선 명

1982년 중앙대학교 전자계산학과 졸업(학사)

1984년 중앙대학교 대학원 전자계산학과 졸업(이학석사)

1987년 중앙대학교 대학원 전자계산학과 졸업(이학박사)

1988년 독일 Bonn대학 Informatik III post doctor

현재 대전대학교 컴퓨터공학과 부교수

관심분야 : 소프트웨어 테스팅, 소프트웨어 재공학, 역공학, 품질보증, 표준화

진 영 택

1981년 중앙대학교 전자계산학과 (이학사)

1983년 중앙대학교 전자계산학과 (이학석사)

1992년 중앙대학교 전자계산학과 (공학박사)

1983년~1990년 에너지 기술 연구소 연구원

1990년~현재 대진산업대학교 전자계산학과 부교수

관심분야 : 소프트웨어공학(객체 지향 분석/설계, 설계 패턴, 역공학 등)