소프트웨어 유지보수를 위한 리테스팅 방법론과 테스트 케이스 재사용에 관하여

황 선 명† 진 영 택††

요 약

리테스트는 수정된 코드를 확인하는 과정으로 본 논문에서는 수정된 코드를 실행하는데 필요한 리테스팅의 영역과 횟수를 명시적으로 나타내는 방법론을 제시한다. 목적함수를 최소화하기 위하여 제어 흐름 분석을 통한 연결 매트릭스, 도달가능 매트릭스와 테스트 케이스/참조 매트릭스를 사용할 뿐만 아니라 데이터 의존성 분석에 의한 Set/Use 매트릭스를 이용하였다. 목적함수의 최소값에 의하여 재사용되는 최소갯수의 테스트 케이스를 얻을 수 있으며, 이를 근거로 하여 수정시의 신뢰도 및 비용효과를 가져올 수 있는 리테스팅 툴을 개발한다.

On the Retesting Methodology and Reusing Test Cases for Software Maintenance

Sun Myung Hwang† and Yong Taek Jin††

ABSTRACT

Retest arises when attempting to validate code modifications. This paper proposes a retest methodology which explicitly defines the amount of retesting to be performed for any given code change. In order to minimize the objective function, we use the set/use matrix through analysis of data dependence as well as the connectivity, reachability, and test case/reference matrix through program control flow analysis. The value of objective function will give the minimum number of test cases necessary to assure the proposed methodology, we develope the retesting tool for generating mininum test cases based on the function.

1. 서 론

테스팅과는 별도로 유지보수는 전체 라이프싸이클 개발비용의 40~60%를 차지하고 있음에도 불구하고 이를 위한 기법과 도구의 연구는 아직도 국내.외적으로 미약하기 그지없다. 본 연구는 유지보수의 많은 활동 중 가장 비중이 많은 코드의 수정 및 변화에 따른 검증을 위한 리테스팅 기법과 도구에 대한 내용이다. 리테스팅이란 수정뒤에 변화된 소프트웨어의 테스팅과정을 말하는데, 이는 앞에서 언급한 소프트웨어 라이프싸이클에서 전체 시스템을 확인하고 검증하는 테스팅 활동과는 구별된다. 리테스트는 다음과 같은 문제를 다룬다.

(1) 코드의 수정으로 인하여 영향을 받는 영역은 어디인가[9, 10]?

(2) 이전에(테스트단계) 사용된 테스트 케이스들을 재사용할 것인가? 만일 그렇다면 어떤 테스트 케이스를 몇개나 사용할 것인가?

(3) 수정으로 인하여 새로운 테스트 케이스를 생성해야 하는가?

· 본 연구는 한국과학재단의 '93 핵심 전문 연구 지원에 의해 수행됨.

†정 회 원 : 대전대학교 컴퓨터공학과 조교수
††정 회 원 : 대전산업대학교 전자계산학과 조교수

논문접수 : 1994년 9월 9일, 심사완료 : 1994년 12월 9일

만일 그렇다면 몇 개를 만들어야 하는가? 이러한 문제에 관한 해답을 얻기 위하여 보다 체계적이고 타당한 리테스팅기준(retesting criteria)을 설정하고 이러한 기준에 맞는 자동화 도구를 개발함이 필요하다.

이러한 필요를 충족시킬 때 과다한 유지보수의 비용을 절감함은 물론 신뢰성있는 품질의 소프트웨어를 유지할 수 있으며 생산성 향상을 도모할 수 있다[7, 8, 9].

테스트 단계나 유지보수단계에서 체계적인 리테스팅방법 또는 기법에 대한 연구는 아직까지 극히 미흡하다. 이제까지 주로 사용된 리테스팅 방법은 정량 분석(quantitative) 방법과 임기응변적인 방법으로 이전에 완성된 테스트 케이스를 부분적으로 이용하여 재실행하였다.

이 방법이외에 다음과 같은 방법들이 연구되고 있다[5, 6].

(1) random retest : 임의로 선택한 테스트 케이스의 재실행
(2) modified code : 수정된 코드를 실행하는 모든 테스트케이스를 재실행
(3) confidence retest : 프로그램의 주요 기능을 모두 실행하는 새로운 테스트 케이스의 실행

각 방법들은 부분적인 장점이 있기는 하나 신뢰할 수 있는 해결방법으로는 아직 부족한 점이 많다.

본 연구의 목적은 가장 효율적이고 신뢰할 수 있는 리테스팅방법을 발견하고 이후에 이를 자동화된 도구로 설계하는 것이다. 연구될 리테스팅 방법과 기준들은 다음과 같은 문제를 해결할 수 있다.

(1) 테스트 위치 : 소프트웨어가 수정된 후 리테스트가 필요한 부분은 어느 곳인가?
(2) 리테스트 횟수 : 얼마나 많은 리테스팅이 필요한가?
(3) 최적의 리테스팅 방법 : 가장 효율적인 리테스팅 방법은 무엇인가?
(4) 테스트 케이스 선정 : 리테스팅에 필요한 테스트 케이스는 무엇인가?

이 같은 연구 목적을 위하여 리테스트 기준과 커버리지가 연구되어야 한다. 실제 시스템으로 구현함은 유지보수단계뿐만 아니라 이전 단계인 테스팅단계까지도 지원할 수 있어 그 효과가 크다. 또한 관리자는 하나의 모듈에 대한 수정이 시스템전체를 다시 실행하게 되어 많은 시간과 비용을 소비하는 파급효과를 최소화시킬 수 있다. 또한 선정된 리테스트방법을 통하여 자동적으로 리테스팅활동을 분석하고 지원하는 도구를 개발한다. 개발된 리테스팅도구는 프로그램의 제어구조와 데이타구조를 통하여 D-D(Decision-Decision) 경로 분석과 최소의 테스트 케이스를 생성한다.

2. 리테스트 방법

리테스팅은 주어진 소프트웨어 유지보수환경하의 목적과 제약조건들에 따라 그 접근방법이 다양하다. 이제까지 전형적인 리테스팅방법으로는 주어진 스케듈과 비용조건하에서 가능한 많은 리테스팅을 실행토록하는 '샷건(shotgun)'방법이 사용되어왔다.

본 연구에서는 코드의 변경이 발생되었을 때 D-D경로를 이용하여 리테스팅되어질 범위와크기를 명시적으로 정의할 수 있는 6가지 전략과 이들 중 가장 효율성이 높은 리테스트 전략 6을 중심으로 그 알고리즘을 제시하고자 한다[14].

(1) 리테스트 전략 1

이전에 사용된 테스트 케이스들을 모두 재실행하는 가장 간단한 리테스팅 전략으로 이 테스트 케이스들은 모듈의 기능에 대한 전체 테스트 커버리지를 제공한다.

(2) 리테스트 전략 2

이 전략은 각 테스트 가능 경로에 대한 테스트 케이스가 설계되어야 한다[11, 15].

그러나 프로그램 내의 반복문이 포함되어 있을 경우 테스트 가능 경로의 수는 반복 횟 수에 따라 기하 급수적으로 증가하기 때문에 본 전략은

고려대상에서 제외시킨다.

(3) 리테스트 전략 3

만일 어떤 테스트 케이스가 수정된 코드를 실행시킬 수 없다면 그것은 변경된 코드 에 아무런 영향을 주지 못할 뿐 아니라 재 실행시킬 필요가 없다. 이 전략은 수정된 코드를 실행하는 테스트 케이스로만 리테스팅을 제한하는 방법이다. 이때 테스트 케이스 크로스 참조 매트릭스를 이용한다. 주어진 테스트 케이스들이 어떠한 D-D 경로 들을 포함하는가를 표시하는 테스트 케이스/ 참조 매트릭스로 부터 리테스트시의 필요 한 테스트 케이스를 추출해 내는 방법으로, 수정된 D-D 경로에 해당하는 테스트 케이스/ 참조 매트릭스의 행에 '1'로 표시된 테스트 케이스 만을 선택한다. 그러나 이 방법은 수정된 D-D 경로를 실행하는 테스트 케이스만으로 선택이 제한되어 있는 것이 문제다.

(4) 리테스트 전략 4

이 전략은 프로그램을 보다 관리하고 처리하기 쉬운 D-D경로로 분해하여 프로그램 내의 모든 D-D경로를 최소한 한번 이상 실행하도록 하는 방법이다. 이를 실행하기 위하여 D-D 경로 도달가능 매트릭스 목적함수가 이용된다[14, 15].

(5) 리테스트 전략 5

전략 4에서는 수정된 코드로부터 도달하는 D-D경로에 대한 리테스트만을 다루었으나 수정된 코드에 도달하는 D-D경로를 포함하여 리테스트하는 보다 효율적인 방법이다. 이를 실행하기 위하여 D-D경로 도달가능 매트릭스와 목적함수가 이용한다.

(6) 리테스트 전략 6

수정된 코드에 도달하고 수정된 데이타를 정의하는 경우와 수정된 코드로부터 도달하고 수정된 데이타를 사용하는 경우를 동시에 만족하는 D-D 경로를 리테스트 한다. 가장 핵심적인 제안인 전략 6은 이전의 5가지 리테스트 전략이 리테스트 집합을 결정하는데 프로그램 제어 구조만을 고려한 것과는 달리 프로그램의 데이타 흐름 구조까지를 고려한다. 프로그램의 데이타 의존 구조(data dependency structure)는 데이타들간의 논리적 관계를 설명한다.

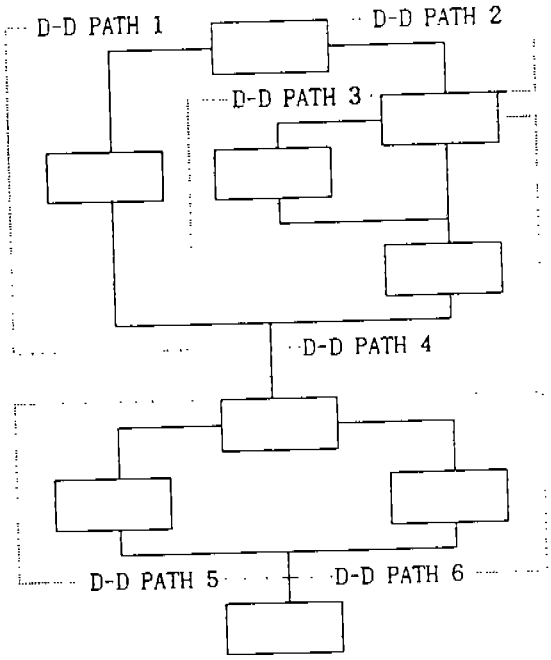
3. 목적함수를 이용한 리테스팅 (전략 6 중심)

3.1 프로그램 제어구조 분석 및 D-D경로

수정된 프로그램을 리테스팅하는 가장 초기 단계로서 프로그램의 제어 구조를 분석하여 D-D경로를 찾아야 한다. 이때 제어구조상에서 D-D 경로는 다음과 같이 정의한다.

- 엔트리(Entry) 노드에서 처음 만나는 결정(Decision) 노드까지
- 결정 노드에서 다음 나타나는 결정 노드까지
- 결정 노드에서 종료 노드까지

정의된 D-D경로를 찾아서 고유번호를 주면 수정이 어느 경로상에서 발생하는지 쉽게 판단할 수 있다. (그림 1)은 프로그램의 제어구조와 D-D 경로를 나타낸다.



(그림 1) 제어구조와 D-D경로
(Fig. 1) Control flow and D-D paths

3.2 테스트 케이스/참조 매트릭스

수정되기 이전 프로그램에서 사용한 테스트 케이스들 각각이 제어 구조상의 어떠한 D-D경로들을 통과하는가의 정보를 제공하는 매트릭스이다. (그림 1)에 대한 테스트 케이스/참조 매트릭스의 예를 〈표 1〉에서 볼 수 있다.

〈표 1〉 테스트 케이스/참조 매트릭스
〈Table 1〉 Test Case/Reference Matrix

D-D경로 번호	테스트 케이스					
	1	2	3	4	5	6
1	1	1	0	0	0	0
2	0	0	1	1	1	1
3	0	0	1	1	0	0
4	0	0	0	0	1	1
5	1	0	1	0	1	0
6	0	1	0	1	0	1

만일 수정이 D-D경로 3에서 발생되었다면 이 매트릭스의 셋째 행(row)이 '1'인 테스트 케이스 3과 4에 관심을 기울여야 한다. 이와같이 수정이 발생되면 기존의 모든 테스트 케이스를 재실행하지 않더라도 테스트 케이스/참조 매트릭스로 부터 재 실행해야 할 필수 테스트 케이스를 찾을 수 있다.

3.3 연결 매트릭스와 도달가능 매트릭스

수정된 D-D경로와 연결 가능한 D-D경로는 어느 것인가? 이를 알기 위하여 임의의 D-D경로에서 직접 연결 가능한 모든 D-D경로의 정보를 제공하는 〈표 2〉와 같은 연결 매트릭스가 필요하다.

D-D경로 간에 직접적인 연결 상황은 연결 매트릭스를 통하여 알 수 있는 반면,임의의 D-D경로에서 직접 또는 간접적으로라도 연결 될 수 있는 모든 경로를 찾을 수 있는 것이 도달 가능 매트릭스이다. 이때 자기 자신의 D-D경로는 연결 가능하다고 가정한다(〈표 3〉).

〈표 2〉 연결 매트릭스
〈Table 2〉 Connectivity Matrix D-D경로

D-D경로	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	0	1	1	0	0
3	0	0	0	0	1	1
4	0	0	0	0	1	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

〈표 3〉 도달 가능 매트릭스
〈Table 3〉 Reachability Matrix D-D경로

D-D경로	1	2	3	4	5	6
1	1	0	0	0	1	1
2	0	1	1	1	1	1
3	0	0	1	0	1	1
4	0	0	0	1	1	1
5	0	0	0	0	1	0
6	0	0	0	0	0	1

3.4 목적함수 (Objective Function)

리테스트의 관점은 유지보수의 환경적인 측면과 기술적인 측면으로 나눌 수 있는데 본 연구에서는 기술적인 측면만을 다루도록 한다. 리테스트 전략 4에서 전략 6까지는 리테스트를 위한 테스트 케이스를 최소화하기 위한 최적화 기법이 필요하다. 최적화 기법은 다음과 같은 함수의 최소화 모델로 구성된다.

$$Z = C_1X_1 + C_2X_2 + \ldots + C_nX_n$$

constraints :

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \geq b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \geq b_2$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m$$

(Z : 목적함수, C_i : 목적함수의 비용요소,
a_{ij} : 상관계수, b_i : i행 최저치, x_j : 변수)

리테스트 모델 형식(함수 Z)은 참조 매트릭스, 연결 매트릭스, 그리고 도달 가능 매트릭스를 통하여 축소법칙이 적용되어 최소화된다. 만일 n개의 테스트 케이스와 m개 D-D경로를 가진 소프트웨어의 테스팅을 가정하면, 위의 목적함수에서 각 Xj는 n개 테스트 케이스 중 하나와 유일하게 대응한다.

최종결과 $X_j = 1$ 인 경우, j번째 테스트 케이스는 리테스트 되어야 하며 $X_j = 0$인 j에 대해서는 대응되는 j번째 테스트 케이스는 기본 모델에 포함시킬 필요가 없다.

목적함수의 비용요소(Cj)는 각 테스트 케이스를 재 실행하는데 관련된 실제 비용을 뜻한다.본 연구에서는 각 테스트 케이스의 재 실행 비용이 동일하다고 가정한다.($C_j=1$)

제한 상관계수(a_{ij})는 테스트 케이스/참조 매트릭스로 부터 직접 구할 수 있다. 만일 테스팅이 테스팅 툴에 의해 이루어진다면 제한 상관계수는 자동적으로 구해질 수 있다. bi의 값은 D-D경로 i가 테스트 되어지는데 필요한 것인지 여부를 반영한 것으로 도달 가능 매트릭스를 이용하여 프로그램의 분기구조를 검사함으로 결정한다. 만일 k번째 D-D경로가 수정되었다면 도달 가능 매트릭스의 k번째 행과 k번째 열이 b값을 결정하는 역활을 한다. 목적 함수의 최적 해법은 수정된 코드로부터 도달가능한 모든 D-D경로들을 최소한 한번이상 실행가능하도록 하는 최소의 테스트 케이스를 구하는 것이다. 이때 X_i의 결과가 1일 때 리테스트 대상 테스트 케이스로 결정된다.

4. 함수의 단순화와 데이타 의존성 분석

4.1 함수의 단순화

함수의 기본적인 모델대로 처리하기 위해서는 많은 기억장소가 필요하고 데이타의 양이 많을 때는 오버플로우도 발생할 수 있으므로 다음 4가지 방법으로 데이타의 크기를 감소시키는 단순화 법칙이 필요하다.

〈법칙1〉 수정된 D-D경로를 실행하지 않는 테스트 케이스는 리테스트에서 제외한다. 즉 수정된 D-D경로와 대응되는 행에서 0을 포함하는 테스트 케이스(열)를 제거한다.

〈법칙2〉 수정된 코드로 부터 도달 가능하거나 수정된 코드까지 도달하지 못하는 D-D경로와 대응되는 제한조건들을 제거한다. 그러므로 bi값이 0인 제한조건은 제거된다.

〈법칙3〉 다른 제한조건과 중복되는 제한조건은 모델로 부터 제거된다.

〈법칙4〉 목적함수(Z)안의 모든 요소가 공통으로 포함하는 제한조건은 모델로 부터 제거된다. 만일 D-D경로 3에서 수정이 이루어 졌다면, 〈표1〉에서의 테스트 케이스/참조 매트릭스 요소는 함수식의 a_{ij}값인 제한 조건 수식의 상관계수를 나타내고, b값은 도달 가능 매트릭스(〈표3〉)의 [3행]과 $[3열]^{-1}$의 논리 OR 연산에 의해 결정된다. 도달가능 매트릭스의 세번째 행은 D-D경로 3에서 부터 도달가능한 임의의 D-D경로를 의미하고, 세번째 열은 D-D경로 3까지 도달가능한 D-D경로를 뜻한다. 즉 수정된 D-D경로와 연결 가능한 모든 D-D경로를 논리 OR에 의해 표시하면 다음과 같이 b값이 결정된다.

도달 가능 매트릭스 [3행] : 001011
도달 가능 매트릭스 $[3열]^{-1}$: 011000
논리 OR 011011(b값)

최소화 함수 $Z=X_1+X_2+X_3+X_4+X_5+X_6$

조건
$$\begin{cases} X_1+X_2 \geq 0 \\ X_3+X_4+X_5+X_6 \geq 0 \\ X_3+X_4 \geq 0 \\ X_5+X_6 \geq 0 \\ X_1 \quad X_3 \quad X_5+ \geq 1 \\ X_2 \quad +X_4 \quad +X_6 \geq 1 \end{cases}$$

위와 같은 모델에 단순화 법칙을 적용하여 다음과 같이 간단하게 변형 시킬 수 있다.

D-D경로 3 이 수정되었다면 법칙1이 적용된 후 D-D경로 3을 실행하는 테스트 케이스(X_3)을 발견하기 위하여는 테스트케이스/참조 매트릭스의 3행에서 X_3와 X_4를 결정한다. 그러므로 테스트 케이스 1, 2, 5, 6은 함수와 조건식에서 모두 삭제된다.

최소화 함수 $Z = X_3+X_4$

$$\text{조건} \begin{cases} X_3+X_4 \geq 0 \\ X_3+X_4 \geq 1 \\ X_3 \geq 1 \\ X_4 \geq 1 \end{cases}$$

b_i 값이 0인 모든 제한 조건은 제거하는 법칙의 적용후의 함수는 다음과 같다.

최소화 함수 $Z= X_3+X_4$

$$\text{조건} \begin{cases} X_3+X_4 \geq 1 \\ X_3 \geq 1 \\ X_4 \geq 1 \end{cases}$$

법칙 4에 의하여 목적함수 Z를 구성하는 모든 항을 갖고있는 제한 조건을 제거한다. 위의 함수에서 Z는 X_3와 X_4로 구성되고 첫번째 조건 또한 X_3+X_4 이므로 이 조건은 삭제가 가능하다.

최소화 함수 $Z=X_3+X_4$

$$\text{조건} \begin{cases} X_3 \geq 1 \\ X_4 \geq 1 \end{cases}$$

그러므로 단순화 법칙 1-4까지 적용한 이후의 함수 Z를 최소화 하는 X_3와 X_4 는 각각 1이며 이때 목적 함수값은 2가 된다. 이는 두개의 테스트 케이스가 리테스트를 위하여 다시 재 실행되어야 한다는 의미이고 이때의 테스트 케이스는 3과 4이다.

4.2 Set/Use 매트릭스의 생성

D-D경로를 이용한 리테스트 방법은 제어흐름에 기초한 방법으로서 수정의 대상이 되는 데이타에 관한 정보를 리테스트 과정에서 반드시 반영해야 할 필요가 있다. 프로그램내의 데이타 흐름을 나타내는 데이타 의존성 분석 자료로 Set/Use 매트릭스를 사용한다. Set/Use 매트릭스 안의 데이타는 각 D-D경로 내에서 어떤 변수가 정의 또는 사용되었는지를 나타낸다. 이때 'S'는 set, 'U'는 used, 'X'는 set and used를 의미한다.

〈표 4〉 Set/Use 매트릭스
〈Table 4〉 Set/Use Matrix D-D경로

변수	1	2	3	4	5	6
A	X	U	U	U	U	U
B	U	U	U	U		
X		U	X	X	X	U
Y			U		U	
Z			U		U	

(그림 1)에 대한 Set/Use 매트릭스가 위와 같을 때 제어흐름에 의한 목적함수와 제한조건의 bi값을 결정하는 것을 다음 절차에 의하여 변경하여야 한다.

단계 1) Set/Use 매트릭스 〈표 4〉의 수정된D-D경로의 row 0를 1로 set한다. 이때 D-D경로 3열에 U 또는 X인 변수들을 찾아 기억한다.

단계 2) 기억된 U,X에 해당되는 변수가 D-D경로 1열에서 D-D경로 6열까지 S 또는 X가 존재하는지를 조사하고 해당되면 row 0의 각 열에 1로 set한다.

단계 3) 도달 가능 매트릭스의 3열을 찾아 row 0 값과 논리곱(AND)을 하여 이를 기억한다.

〈표 5〉 Set/Use 매트릭스에서의 row 0값
〈Table 5〉 Row 0 value of Set/Use matrix D-D경로

변수	1	2	3	4	5	6
A	X	U	U	U	U	U
B	U	U	U	U		
X		U	X	X	X	U
Y			U		U	
Z			U		U	
row 0	1	0	1	1	1	0

위의 단계 1), 2)는 수정된 D-D경로상에서 used되는 변수들에게 영향을 미치는 set변수들이 어느 D-D경로상에 존재하는가를 찾는 단계이다.

그러므로, 도달가능 매트릭스와 row 0 값을 AND하면 다음과 같다.

도달 가능 매트릭스 3열 : 011000
row 0 101110
논리 AND 001000

논리곱 AND의 결과 값은 수정된 D-D경로까지 도달하는 임의의 D-D경로와 이들 경로상에 존재하는 데이타 의존 관계를 동시에 반영한 것이다.

단계 4) Set/Use 매트릭스의 수정된 D-D경로의 row 1을 1로 set한다. 이때 D-D경로 3열에 S 또는 X인 변수들을 찾아 기억한다.

단계 5) 기억된 S, X에 해당되는 변수가 D-D경로상에서 U 또는 X 되었는지 조사하고 존재하면 해당 row 1의 각 요소를 1로 set한다.

단계 6) 도달 가능 매트릭스의 수정된 경로(3행)를 찾아 row 1 값과 논리곱(AND) 한다.

단계 4), 5)는 수정된 D-D경로상에서 set된 변수들이 다음 어느 D-D경로상에서 used 되어 영향을 미치는지를 찾는 단계이다. 이 단계가 실행된후의 set/use 매트릭스의 row 1 값은 〈표 6〉과 같다.

〈표 6〉 Set/Use 매트릭스에서의 row 1값
〈Table 6〉 Row 1 value of Set/Use matrix D-D경로

변수 \ D-D경로	1	2	3	4	5	6
A	X	U	U	U	U	U
B	U	U	U	U		
X		U	X	X	X	U
Y			U		U	
Z			U		U	
row 1	0	1	1	1	1	1

row 1 값은 도달 가능 매트릭스의 3행과 논리 AND 연산을 하면 다음과 같다.

도달 가능 매트릭스 3행 : 001011
row 1 011111
논리 AND 001011

단계 7) 단계 3)과 단계 6)에서의 논리 AND한 결과값을 논리 OR 연산하여 최종 b값을 결정한다.

단계 3) 결과 001000
단계 6) 결과 001011
논리 OR 001011 ==> b값

논리 OR 연산의 결과 목적함수의 최종 b 값을 결정한 후 앞에서 설명한 단순화 법칙을 적용하여 함수를 최적화 시킨다. 이것은 최소화된 함수의 구성요소를 분석하여 테스트 케이스의 최소수를 파악하는데 도움이 된다.

테스트 베드(test bed) 상에서 존재가능한 테스트 케이스의 갯수를 N이라 하고 재 실행하기 위한 최소의 테스트 케이스들의 수를 P라 한다면 다음의 식을 얻을 수 있다.

① 존재 가능한 테스트 케이스의 조합 수 : $\frac{N!}{(N-P)!P!}$

② 타당한 테스트 케이스가 선택될 확률 :

$= \frac{\text{실행가능조합의 수}}{\text{존재가능한 조합의 수}}$

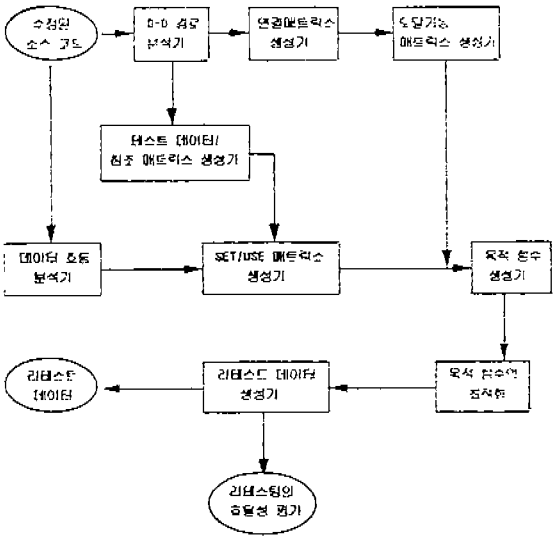
5. 리테스팅 툴과 사례연구

5.1 툴의 구성

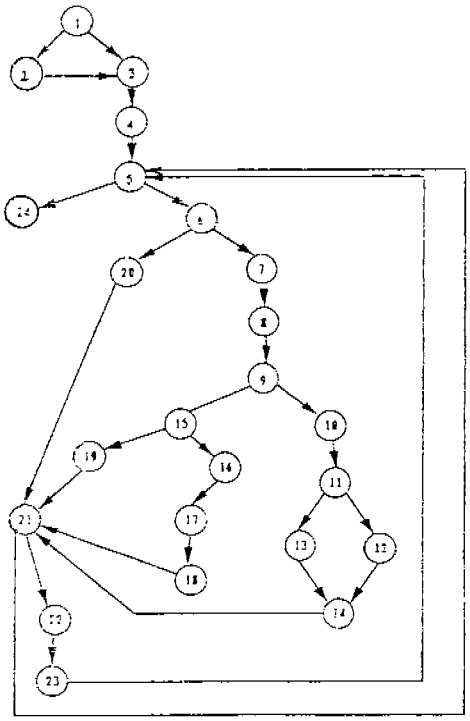
이상에서 설명된 D-D 경로를 통한 매트릭스 제시와 목적함수의 단순화 과정으로부터 리테스팅 데이타와 레포트까지 생성하는 시스템의 구성은 (그림 2)와 같다.

5.2 사례 연구

먼저 리테스팅 대상 프로그램의 제어 흐름 구조가 (그림 3)과 같다고 하자. 이때 그래프상의 각 노드는 프로그램상의 각 문장과 일대일 대응



(그림 2) 시스템 구성도
(Fig. 2) System Organization



(그림 3) 제어 흐름 그래프
(Fig. 3) Control Flow Graph

하도록 할 수도 있으나 본 연구에서는 제어 흐름 구조가 변하지 않는 문장들은 하나의 노드로 통합하여 표시하였다. 제어 흐름 그래프로부터 D-D 경로를 구별하고 각 D-D 경로에 포함되는 그래프상의 노드들은 〈표 7〉과 같다. 이때 D-D 경로 5번에서 수정이 발생한다고 가정하고 이 수정으로부터 기존의 테스트 데이타의 재사용 가능성을 확인하여 보자.

테스트 케이스/참조 매트릭스, 도달 가능 매트릭스, Set/Use 매트릭스를 구하고 이들로 부터 목적함수와 제한 조건을 얻을 수 있다. 단순화된 목적 함수는 수정된 D-D경로와 관계가 있고 재실행할 리테스트 데이타를 생성하게 된다.

〈표 7〉 D-D 경로와 그래프 노드
〈Table 7〉 D-D paths and Nodes

D-D 경로	D-D 경로상의 문장
1	1, 2, 3, 4, 5
2	1, 3, 4, 5
3	5, 24
4	5, 6
5	6, 7, 8, 9
6	9, 10, 11
7	11, 12, 14, 21
8	11, 13, 14, 21
9	9, 15
10	15, 16, 17, 18, 21
11	15, 19, 21
12	6, 20, 21
13	21, 22, 23, 5
14	21, 5

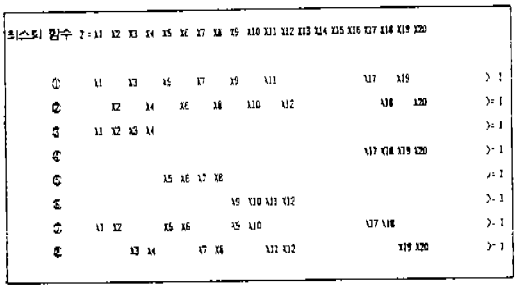
〈표 8〉 테스트 케이스/참조 매트릭스
〈Table 8〉 Testcase/Reference Matrix

PATH, CASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
5	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0
6	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
7	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
10	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
13	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0
14	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

〈표 9〉 도달가능 매트릭스
〈Table 9〉 Reachability Matrix

PATH..PATH	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1	1	1	1
5	0	0	1	1	1	1	1	1	1	1	1	1	1	1
6	0	0	1	1	1	1	1	1	1	1	1	1	1	1
7	0	0	1	1	1	1	1	1	1	1	1	1	1	1
8	0	0	1	1	1	1	1	1	1	1	1	1	1	1
9	0	0	1	1	1	1	1	1	1	1	1	1	1	1
10	0	0	1	1	1	1	1	1	1	1	1	1	1	1
11	0	0	1	1	1	1	1	1	1	1	1	1	1	1
12	0	0	1	1	1	1	1	1	1	1	1	1	1	1
13	0	0	1	1	1	1	1	1	1	1	1	1	1	1
14	0	0	1	1	1	1	1	1	1	1	1	1	1	1

〈표 10〉 목적함수
〈Table 10〉 Objective Function



〈표11〉 리테스트 데이타의 집합
〈Table 11〉 Set of Retest Data

(1,5,10,19)	(2,5,9,20)	(3,5,9,20)	(4,5,9,17)
(1,5,11,18)	(2,5,10,20)	(3,5,10,18)	(4,5,9,20)
(1,5,11,20)	(2,5,11,17)	(3,5,10,19)	(4,5,10,17)
(1,5,12,17)	(2,5,11,18)	(3,5,10,20)	(4,5,10,18)
(1,5,12,18)	(2,5,11,19)	(3,5,11,18)	(4,5,10,19)
(1,5,12,19)	(2,5,11,20)	(3,5,11,20)	(4,5,10,20)
(1,5,12,20)	(2,5,12,17)	(3,5,12,17)	(4,5,11,17)
(1,6,9,19)	(2,5,12,18)	(3,5,12,18)	(4,5,11,18)
(1,6,9,20)	(2,5,12,19)	(3,5,12,19)	(4,5,11,19)
(1,6,10,19)	(2,5,12,20)	(3,5,12,20)	(4,5,11,20)
(1,6,10,20)	(2,6,9,19)	(3,6,9,17)	(4,5,12,17)
(1,6,11,17)	(2,6,9,20)	(3,6,9,18)	(4,5,12,18)
(1,6,11,18)	(2,6,10,19)	(3,6,9,19)	(4,5,12,19)
(1,6,11,19)	(2,6,11,17)	(3,6,9,20)	(4,5,12,20)
(1,6,11,20)	(2,6,11,18)	(3,6,10,17)	(4,6,9,17)
(1,6,12,17)	(2,6,11,19)	(3,6,10,18)	(4,6 9 18)
(1,6,12,18)	(2,6 11,20)	(3,6,10,19)	(4,6,9 19)
(1,6,12,19)	(2 6,12,17)	(3,6,10,20)	(4,6,9,20)
(1,6,12,20)	(2,6,12,19)	(3,6,11,17)	(4,6,10,17)
(1,7,9,18)	(2,7,9,17)	(3,6,11,18)	(4,6,10,19)
(1,7,9,20)	(2,7,9,18)	(3,6,11,19)	(4,6 11,17)
.	.	.	.
.	.	.	.
.	.	.	.

최적화된 목적함수(〈표 10〉)의 제약 조건을 분석하면 제약 조건 ③, ④, ⑤, ⑥ 에 의하여 최소 4개의 테스트 데이타 조합을 구성함으로 D-D 경로 5번에 의한 수정을 검증할 수 있음을 알 수 있다. 이때 4개의 조합 중 실행 가능한 경우의 수는 196가지의 재실행 리테스트 데이타가 생성된다. (〈표11〉)

6. 결 론

소프트웨어의 수정에 따른 리테스팅 과정을 보다 체계적이고 타당성 있는 기준을 통하여 수행하기 위하여 본 논문은 제어 흐름 분석과 자료 흐름 분석에 의한 방법을 제안하였다. 제어 흐름 분석 자료로 제어흐름 그래프, 테스트 케이스/참조 매트릭스, 연결 매트릭스, 도달 가능 매트릭스를 구하고, 자료 흐름 분석 자료도 Set/Use 매트릭스를 구하여 수정이 이루어진 코드 영역과 영향을 주고 받는 관계를 발견하였다. 이러한 관계 자료로 목적 함수(Z)의 제한 조건을 결정하여 최소의 목적함수 값을 구하고 리테스트시에 재사용할 테스트 케이스와 그 갯수를 최소화 하였다. 또한 목적함수로 부터 리테스팅시에 타당한 테스트 케이스를 선택할 확률과 실행가능 테스트 케이스 수를 알 수 있다. 이러한 리테스팅 방법론과 자동화된 툴을 통하여 과다한 유지보수 비용을 절감함은 물론 신뢰성 있는 품질의 소프트웨어를 유지할 수 있다.

참 고 문 헌

[1] C.G. Gibson, and L.R. Railing, "Verification Guidelines, TRW Document 17618-H2000-RO-00", NAS 9-8166, August 1971.

[2] K. F.Fischer, "A Test Case Selection Method for the Validation of Software Maintenance Modification", Proceedings, COMPSAC '77, IEEE, Nov. 1977.

[3] S.Warshall, "A Theorem on Boolean

Matricies", Journal of ACM, IX, 1, January 1962.

[4] R. E.Davis, D. A. Kendrik, and Weitzman, "Branch-and-Bound Algorithm for Zero and One Integer Programming Problems", Operation Research, Vol. 19, pp. 1036-1044, 1971.

[5] R. E.Allenson, et.al., "Automated Verification System Programmer's Guide," TRW Systems, Note, FMT, No. 72.

[6] D.Teichroew, "ISDOS and Recent Extensions," Proceedings of the Symposium on Computer Software Engineering, Polytechnic Press, pp. 79, 1976.

[7] Report to the Congress of the United States, "Federal Agencies' Maintenance of Computer Program : Expensive and Undermanaged", February 1981.

[8] J. D.Dnnahoo, and D. Swearingen, "A Review of Software Maintenance Technology", RADC-TR-80-13, Rome Air Development Center, Griffiss AFB, NY, February 1980.

[9] S.S.Yau, and J. Collofello, "Some Stability Measure for Software Maintenance," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, November 1980.

[10] S.S.Yau, and J. Collofello, "Some Stability Measure for Software Maintenance," IEEE Transaction on Software Engineering, Vol. SE-6, November 1980.

[11] C.Gannon, "Error Detection Using Path Testing and Statistic Analysis," IEEE Transactions on Computer, Vol. No. pp. August 1979.

[12] U.Voges, Gmeiner, and Amscher, "SADAT, an Automated Testing Tool", IEEE Transactions on Software Engineering, Vol. SE-6, No. 3, pp. 286-290, May 1980.

[13] K.Tai, "Program Testing Complexity and Test Criteria," IEEE Transactions on Software Engineering, Vol. SE-6, No. 6, November 1980.

[14] B.Beizer, 'Software Testing Techniques,' 2nd ed., Van Nostrand Reinhold, 1990.

[15] D.M.Marks, 'Testing Very Big Systems,' McGraw Hill, 1992.

황 선 명

1982년 중앙대학교 전산과 졸업 (학사)
1984년 중앙대 대학원 전산과 졸업 (석사)
1987년 중앙대 대학원 전산과 졸업 (이학박사)
1988년 독일 Bonn 대학 Informatik III post doctor
1989~현재 대전대학교 컴퓨터 공학과 조교수
관심 분야 : 소프트웨어 품질 보증 및 평가, 소프트웨어 테스팅 기법 및 도구.

진 영 택

1981년 중앙대학교 전산과 졸업 (학사)
1983년 중앙대학교 전산과 졸업 (석사)
1992년 중앙대학교 전산과 졸업 (공학박사)
1982~90년 에너지기술연구소 연구원
1990~현재 대전산업대학교 전자계산학과 조교수
관심 분야 : 소프트웨어공학, 객체지향설계.