高品質 소프트웨어 開發·確保를 위한 複雜度 測定技法

1 9 9 0

經濟企劃院 調査統計局
電算擔當官室

目　　次

Ⅰ. 소프트웨어 品質 測定의 매트릭스 ····· 1

1. 소프트웨어의 品質 ····· 1

2. 品質 매트릭스와 체크리스트 ····· 13

Ⅱ. 소프트웨어 複雜度의 測定·評價方法 ····· 29

1. 소프트웨어 品質을 測定하기 위한 複雜度 ····· 29

2. 프로그램의 사이즈에 의한 複雜度 測定方法 ····· 33

3. 제어 흐름에 의한 複雜度 測定方法 ····· 39

4. 데이타의 흐름에 의한 複雜度 測定方法 ····· 45

5. 複雜한 測度의 問題點 ····· 50

6. 複雜度의 改善方案 ····· 53

Ⅲ. 소프트웨어 構造度의 測定·評價方法 ····· 58

1. 프로그램 構造度의 測定方法 ····· 58

2. 프로그램의 非構造度 ····· 64

3. 끝맺음 ····· 70

Ⅳ. 品質保證技術의 現狀과 課題 ····· 73

1. 테스트의 技術 ····· 73

2. 品質保證 技術 ····· 81

3. 끝맺음 ····· 86

Ⅴ. 소프트웨어 品質評價 TOOL(ESQUT) 紹介 ····· 88
1. 소프트웨어 品質評價의 要因 項目 ····· 89
2. 끝맺음 ····· 107

「자료 : 한국정보산업연합회, 세미나자료, 1990」

I. 소프트웨어 품질 측정의 매트릭스

1. 소프트웨어의 품질

1.1 소프트웨어 품질(SQ) 의 정의

(1) Quality 의 정의

fitness for use ; conformance to requirements

absence of decfcts or error (JOHNS)

(2) Software Quality 의 정의

- IEEE 의 정의 :

  S/W 가 가지고 있어야할 여러 속성들의 정도

- Fisher and Baker :

  S/W 생산물이 갖고있는 목적을 충족시켜주는데 필요한 일련의 규정된 속성

- DDD 표준 :

  규정된 최종물을 사용하도록 해주는 S/W 의 속성의 정도

- SQA 핸드북 :

  The fitness for use of the total software product

1.2 소프트웨어 품질보증 (SQA) 의 정의

(1) IEEE 의 정의

항목이나 제품이 설정된 기술적 요구사항에 맞는지를 확인하기위해

적절한 증거를 확보하는데 필요한 모든 행위를 계획하고 체계화한 것.

(2) Fisher and Baker :

S/W 품질보증은 S/W 품질척도 및 측정을 수행하는 기능적 실체임.

(3) SQA 핸드북

모든 S/W 생산물을 사용하는데 있어 적절한 확증을 준비하는 체계적 행위

1.3 IEEE 정의에 대하여

(1) It is a comprehensive view rather than a restrictive one.

재한적 관점보다는 이해적인 관점임.

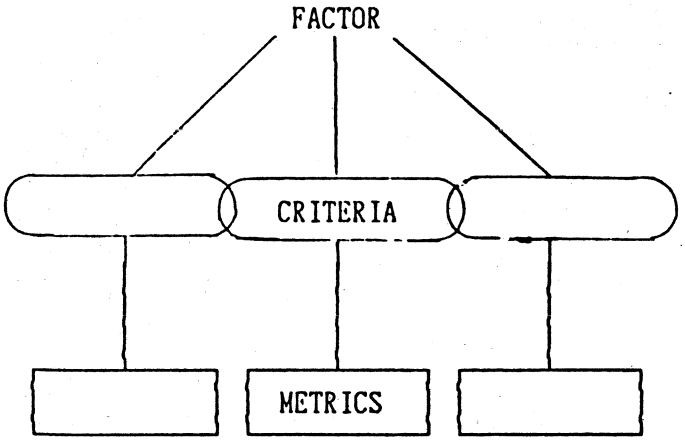
(2) The emphasis is on a plan and its systematic implementation to achieve the ohjectives of software quality.

품질목표 달성을 위한 계획과 체계적 실현에 중점을 둠.

(3) The notion of quality is relative to some prespecified requiremant rether than to some absolute sence of quality.

품질의 개념에 대한 절대적 감각보다는 사전의 요구사항에 치중.

(4) The purpose is not to gurantee 100 percent reliability or zero defects: It is rather of increase confidence that ecery reasonable step has been taken during the development phase to assure that quality of the end product.



소프트웨어 품질 보증을 위한 접근

품질보증의 목적이 최종제품에 대한 완전무결에 대한 보장이 아니며, 개발 과정의 각 단계별 타당한 절차로 최종제품의 품질에 대한 신뢰성 증가에 둠.

1.4 품질보증의 정의 및 목표

소프트웨어는 그것이 개발되는 과정이나 개발환경 자체에서 에러를 포함하게 되는데 품질보증이란 이러한 내포된 에러들을 찾아내어 수정하는 활동을 말하며 더 나아가서 는 에러가 존재하지 않도록 대비하는 일이기도 하다. 그러므로 소프트웨어 품질보증 의 필수적인 과정은 테스팅이다.

제품의 품질향상을 저해하는 것은 제품의 설계과정에서 발생한 에러가 코딩과 테스팅 과정에서 계속 발생하여 Life Cycle마지막 단계에까지 이어지는 경우로 품질보증 작업은 Life Cycle전반에 걸쳐 시행되며

무엇보다도 제품공정 초기에 에러를 검출하는 것이 최상책이다.

(1) S Q A 의 이슈

a) S. Q의 정의에 대한 이해와 동의가 쉽지 않으며, S.Q의 측정방법이 쉽지 않다.

b) 품질을 생산물에 부여하기 때문에 S/W 개발의 각 단계에서 보증되어야 한다.

c) 대다수 요원이 참여하고 장기간에 걸친 프로젝트에서 효과적 프로젝트 관리가 없이는 실현이 어렵다.

d) TOOL 의 가용성면에서 어떤 작업단계에서든지 성공적으로 실행하는데 애로점이 있다.

e) 이상의 모든 문제점을 해결하기 위한 SQA는 계획수립과 실행(구현)이 필요하다.

(2) S Q A 의 효과

a) 어느 품질요인이 중요한지 프로그램 관리자에게 Mechanism을 제공.

b) 설정된 품질목표의 효율적 실행을 위한 정량적 방법을 제공.

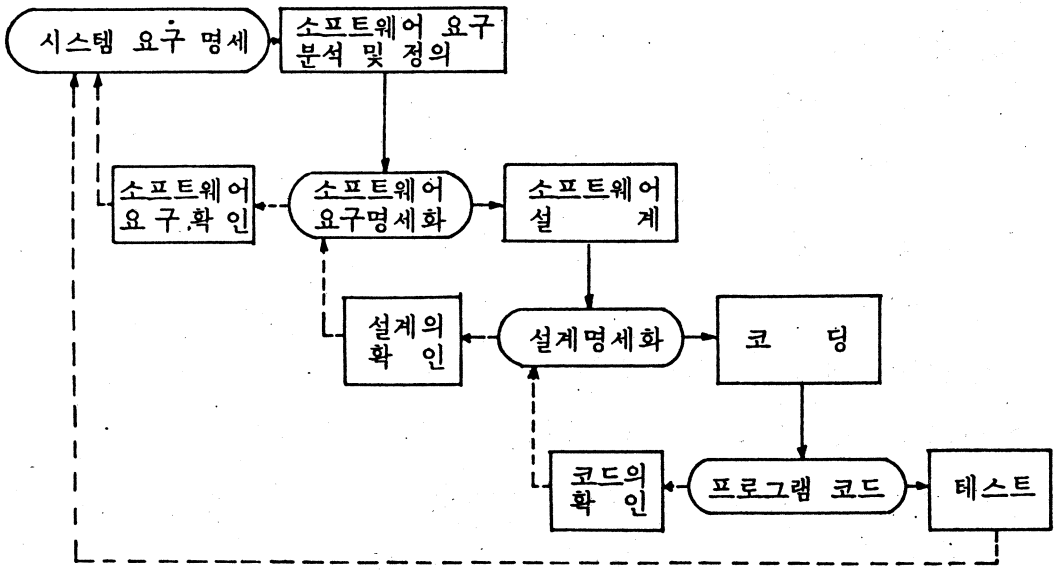
c) 개발기간동안 QA 담당자에 의해 Interaction 을 제공.

d) 저질화를 초기에서 지적하여 다른 작업에 영향을 미치지 못하게 함.

1.5 소프트웨어 라이프 사이클(Software Life Cycle)

소프트웨어 라이프 사이클에는 여러가지 형태가 있을 수 있는데 그림에서

볼 수 있듯 이 소프트웨어 요구분석 및 정의, 명세서 작성, 설계, 코딩, 테스트가 있다. 그리고 소프트웨어 확인과정으로는 요구확인, 설계확인, 프로그램, 코드확인, 프로그램 테스트의 순으로 이루어진다



[도1] 소프트웨어 품질의 확인 단계

1)요구분석 및 정의

이 단계의 근본 목표는 사용자의 요구를 정확히 정의하고 이를 명세서에 명료하게 반 영하여 소프트웨어 개발팀에 맡기도록 하며, 이 명세서에 의거하여 생산하였을 때 사용자 요구에 부합되는 것이다.

이과정에서 시행해야 하는 활동은 다음과 같다.

(1) 요구 분석의 계획 수립

.문제 분석 : 문제의 상황 조사

.작업 체제 : 요구 분석의 작업 요원 및 체제 결정

.작업 방침

.작업 일정

(2) 현행 시스템 조사

(3) 요구 분석

. 요구의 추출

. 요구의 정리

. 요구의 평가

2) 명세서(Specification) 작성

종래에는 소프트웨어에 대한 요구의 기술 및 명세는 서술적인 문서에 의존해 왔기 때문에 그 기술은 부정확하고 사용자와 개발자 사이의 오해요소가 많았었다. 또한 소프트웨어의 복잡한 기능은 자연어로 표현하기에 어려운 점이 많았으므로 명세서를 작성하는 작업도 쉽지 않았었다.

Parnas는 명세서가 갖추어야할 요건을 다음과 같이 제안하였다.

(1) 사용자가 프로그램을 올바르게 사용할 수 있도록 작성한다.

(2) 개발팀이 프로그램을 완성하는데 필요한 모든 정보가 중복없이 최소한으로 제공되어야 한다.

(3) 충분한 형식성을 가지고 일관성, 완전성이 검토되어야 한다.

(4) 사용자와 개발자의 공통언어의 사용으로 애매모호성을 배제한다.

이와같은 방법들을 적용하여 종래의 명세서가 내포하는 문제점을 시정하기 위한 요구분석 기법은 다음과 같다.

(1) 도식에 의한 명세방법

(2) Prototyping

(3) 요구정의 언어에 의한 명세법

3) 설계

(1) 기본 설계와 상세 설계

소프트웨어 설계는 요구분석과 정의로부터 요구명세를 기초로하여 소프트웨어 속성 (기능,성능)의 집합을 가장 적합하게 실현하는 과정으로 문제를 해결하기위한 시스템의 전체구조를 다음과 같이 형성하여 이를 문자 및 그림으로 표현한다.

. 시스템 조직이나 이에 필요한 데이타를 추상화한다.

. 시스템 각 부분사이에 있는 인터페이스를 확립하고 제어와 데이타의 연결을 명확히 한다.

. 목표한 시스템의 품질을 달성하기 위한 설계상의 여러조건과 trade-off를 찾 는다.

. 여러가지 설계안에서 가장 적합한 안을 선택한다.

(기본 설계)

요구분석 결과를 바탕으로 문제의 구조에 적절한 시스템의 전체적인 설계를 한다.

(상세 설계)

식별된 시스템의 각 부분을 상세화 시킨다.

(2) 설계의 기본 액티비티와 요인

a) 분석과 합성

시스템이 어떠한 요소로 되어 있는가(분석)를 결정하는 작업과 시스템의 각 특성과 각각 관계되는 요소들로 부터 시스템의 집합을 산출하는 과정(합성)으로 이루어진다.

b) 구현화

무엇인가를 기능적으로 사용가능하게 하는 특성

c) 추상화

복잡한 것을 감소시키기 위해 구현대상의 처리순서나 데이타들의 흐름을 알지 않고서도 사용할 수 있도록 한다.

d) 모듈화

복잡도를 경감시키고 다수의 설계자에 의한 시스템의 병행 생산으로 고품질의 향상을 노력한다.

e) 설계의 표현

설계 내용을 이해하기 쉽고 명확하며 변경하기 용이한 형식으로 표현한다.

(3) 설계 방법:소프트웨어를 설계하는데 다음의 두가지 방법이 있다.

. 프로그램을 처리하는 흐름을 중심으로 설계하는 설계방법

. 프로그램으로 처리되는 데이타에 주목하는 설계방법

전자의 제어중심형 설계법은 단계적 상세화, 기능분할,모듈러 프로그래밍 등이 강조되고 있으며, 후자의 데이타중심형 설계법은 jackson 법이나 warnier-orr 법에서 대표 되는 설계법으로 풀이하는 구조를 데이타 구조에 따라 결정하는 특징을 갖고 있다.

(4) 테스팅

소프트웨어에 관한 테스팅 문제는 다음의 세가지 범주에서 생각할 수 있다.

1) 소프트웨어 라이프 사이클의 각 단계별 효과측정.

2) 사용자의 요구에 맞는 최종 시스템 실행의 확인.

3) 표본 테스트 데이타에 의한 시스템 실행의 확인.

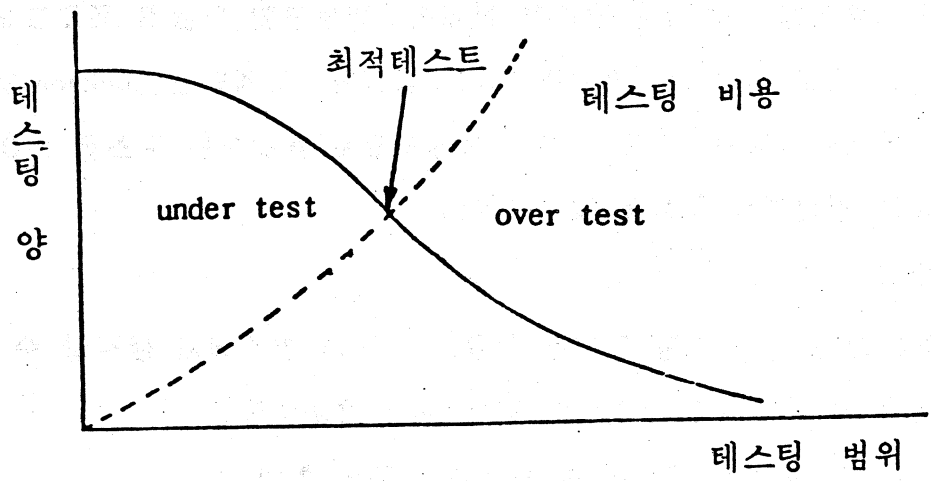
(1) 테스트 기법

- Path testing (Path slection, Loop testing)
- Transaction-flow testing
- Input validation and syntax testing (Test case generation, Ad-lib test)
- Logic-based testing (Test case design, Decision table
- State trasition testing (State graphs)

(2) Testing 구분

- Unit testing (Element testing)
- Integration testing
- System testing
- Configuration, recovery and security testing

다음 그림은 테스팅에 관한 효율적인 비용곡선으로 얼마만큼 테스팅을 하는것이 좋은 가를 설명하고 있다.



[도2] 테스팅 비용 곡선

일반적으로 널리 사용되고 있는 소프트웨어 테스트를 위한 기술로 다음과 같은 것이 있다.

(1) 알고리즘의 평가 테스트

설계하기 전에 알고리즘에 대한 평가 목적으로 테스팅의 속도, 정밀성 및 대상 프로그램의 크기를 중심으로 하여 분석하고 평가 한다.

(2) 분석 모델링 : 고리적 모델링

(3) Auditing

프로그램과 그 Documentation을 인스팩션에 의해서 확인하고 테스팅 하는 기술로 Scheduled Auditing과 Unscheduled Auditing 으로 구분한다.

(4) Code Inspection

코드 검증과 에러의 확인을 위한 기술

(5) Correctness Proofs

수학적인 이론으로 프로그램의 정당성을 증명하는 기술

(6) Design Inspection

설계에 대한 Inspecting과 에러를 찾아내는 기술

(7) Error-Prone Analysis

프로그램의 변경과 수정이 자주 일어날 수 있는 부분을 코딩중에 구분하여 유지 보수를 용이하게 해 주는 기술

(8) Equivalence Classes

테스트할 프로그램에 대한 테스트 케이스를 만드는 기술

(9) 실행 분석

Program Behavior를 조사. 분석하는 기술로 프로그램 실행에 관한 여러가지 정보를 만들어 준다. 동적 테스팅툴은 이 기술을 자동화 시켜 놓은 것이다.

(10) 기능 테스팅

소프트웨어 기능을 테스트하여 그 양식이 요구정의에 합당한지를 테스팅함.

(11) 논리 테스팅

계산결과가 정확한지를 테스팅한다. 정밀도, 에러핸드링, 초기치 모듈간 인터페이스, 타이밍 등을 분석한다.

(12) 패스 테스팅

실행된 문장의 수, Branch Path수, 서브루틴 호출회수등 프로그램의 Control Topology를 기초로한 효율을 분석한다.

(13) Post-Function 분석

기능 테스트후에 기능이 약한 부분을 보완하기 위한 분석.

(14) Reviewing

프로그램 Documentation에 관한 Inspection을 통해서 검증한다.

(15) 시뮬레이션(Simulation)

시스템의 한계내에서 타이밍, 시스템의 성능과 한계 및 제한 조건등을 평가한다.

(16) 표준화

Documentation, 언어, 설계 및 구조적인 프로그래밍의 제반단계를 표준화 한다.

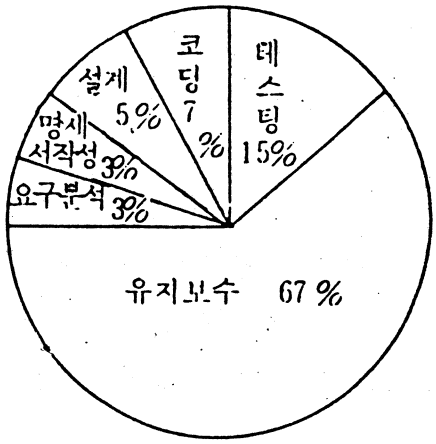
(17) Stress 테스팅

코딩 결과가 시스템의 Spec.을 만족하는가를 검증한다. 이상과 같은 기술이 지원할 수 있는 품질보증의 기능은 다음표에서 설명할 수 있다. 각 기술이 갖는 기능을 X표로 Check 하였으며 어떤 기술은 이미 도구로 개발되었거나 몇가지 기술을 통합한 한개의 도구로 개발이 가능하다.

기능 / 기술	계획	work tasking	구성관리	테스팅	correct-ing Action	문헌관리	설계표준화	분석표준화	Reviews 와 audits	하청관리
1. 알고리즘 평가				×						
2. 분석 모델				×						
3. Auditing	×	×	×	×	×	×	×	×	×	×
4. Code inspection							×	×	×	×
5. Correctness proof				×						
6. Design inspection							×	×	×	×
7. Error-prone analysis				×					×	
8. Equivalence classes				×						
9. 실행분석				×					×	
10. 기능 테스팅				×						
11. 논리 테스팅				×						
12. Path 테스팅				×						
13. Post-funcitional 분석				×					×	
14. Reviewing	×	×	×	×	×	×	×	×	×	×
15. 시뮬레이션				×			×		×	
16. 표준화	×	×	×	×	×	×	×	×	×	×
17. 정적분석				×					×	
18. Stress 테스팅				×						
19. Symbolic execution				×						
20. Walk-throughs							×	×	×	×

5) 유지.보수

소프트웨어의 전체 라이프 사이클을 통해보면 유지보수의 경제적 소비는 다른 것에 비해 월등히 높다고 하는 것이 일반적인 사항이다. 그림에서 보듯이 어떤 연구를 통 해 보면 유지보수의 경제적 비용이 전체의 3분의 2가 넘고 있다. 물론 모든 경우가 이렇다는 것은 아니지만 이는 다른 경우에서도 유사하리라 고 본다.



이와같은 사실은 논리적인 구분화를 진전시킴으로서 보수문제에 따른 오류의 연쇄적 반응을 근원적으로 감소시킬수 있을 것이다.

2. 품질 매트릭스와 체크리스트

2.1 품질 매트릭스의 개념

요인(factor)을 제품의 품질에 관한 관리적인 관점으로 볼 때, 평가기준은 품질을 보 증해 줄 속성을 지닌 소프트웨어적인 관점이며,

매트릭스(Metrics)는 이들 속성을 계속적으로 측정할 수 있는 프로젝트 관리수법이다.

모든 소프트웨어 제품의 가치는 사용자의 요구에 얼마나 부합하느냐에 달려 있으므로 제품을 인수하기 위한 평가기준은 테스팅할 평가기준이 몇개이고 특성이 무엇인가를 정의하는 것이 곧 매트릭스이다. 따라서 인수를 위한 테스트 기준은 사용자가 정하고 이 기준을 테스트할 수 있는 조건으로 바꾸어야 한다. 인수조건을 정하는 것은 인수를 위한 테스팅에 대한 중요한 평가기준의 설정이다.

품질 매트릭스를 테스팅 요소로 분류해 보면 다음과 같다.

(1) 테스팅 횟수

Verification, Validatiion및 Certification 에 따라서 테스팅 단계와 단위를 정하여 모듈, 프로그램의 인터페이스 및 소프트웨어 시스템의 요구정의에 대한 테스팅을 하 기 위해서 평가기준을 설정한다.

(2) 테스팅 경로(path)

테스팅 경로의 수는 경로회수로 정해진다. 각 시스템은 유한개의 논리적인 경로로 이루어 짐으로 매트릭스는 테스트할 논리경로의 한계를 정한다. Program Behavior를 조사할 수 있는 여러가지 평가기준을 설정한다.

(3) 테스팅 비용

소프트웨어의 개발단계별 및 기능별로 분류된 예산에 관한 평가기준을 정한다. 개발 비용에 대한 테스팅 비용을 평가할 수 있으나 테스팅에 소요된 시간과 그 효율에 관 한 관련성은 알기 어렵다.

McCall 과 Boehem 이 제안한 소프트웨어 품질요인에 관한 평가 기준을 설정하기 위해서 기준에 관한 정의를 내리면 다음과 같다.

소프트웨어 품질요인에 대한 기준의 정의

기 준	정 의	관련 요인들
추적 가능성	전문개발과 작용환경에 관한 요구에서 시행까지의 제공하는 SW 특성들.	정 확 성
완 전 성	요구된 기능(함수)에 대한 충분한 수행을 제공하는 SW의 특성들.	정 확 성
일 관 성	일정한 실제와 실행기법, 그리고 표시방법을 제공하는 SW의 특성들.	정확성, 신뢰성, 유지. 보수성
정 밀 성	계산값과 결과치들의 요구된 정밀도를 제공하는 SW의 특성들.	신 뢰 성
에러 인내성	실제의 조건하에서 운영(작업)의 계속성을 제공하는 SW의 특성들.	신 뢰 성
단 순 성	가장 알기쉬운 방법으로 기능들을 수행하는 SW의 특성들(보통 복잡성이 증가되는 실행들의 회피)	신뢰성,유지. 보수성, 테스트용이성
모 듈 화	고도로 독립된 모듈들의 구조를 제공하는 SW의 특성들.	유지.보수성, 유연성, 테스트용이성 이식성, 재사용효율성 결합성
일 률 성	실행되어진 기능들에게 여유폭을 제공하는 SW 특성들.	유연성, 재사용효율성
확 장 성	수치적 기능들이나 Data storage요구들의 확장에 관한 사항을 제공하는 SW 특성들.	유 연 성
자기 표현성	어떤 하나의 기능의 수행에 관한 설명을 제공하는 SW의 특성들.	유연성, 유지 보수성, 테스트 용이성,이식성, 재사용 효율성

기 준	정 의	관련 요인들
인스트루맨테이션	사용방법의 척도나 에러들의 확인을 제공하는 SW특성들.	테스트용이성
실 행 효 율 성	최소처리시간을 위해 제공하는 SW 특성들.	효 율 성
메모리 효 율 성	(운영)작동하는중 최소메모리 요구에 관한 사항을 제공하는 SW 특성들.	효 율 성
엑세스 콘 트 롤	SW의 엑세스의 제어와 데이터에 관해 제공하는 SW의 특성들.	안 정 성
엑세스 오 디 트	SW의 엑세스에 관한 감사와 데이터에 관해 제공하는 SW의 특성들.	안 정 성
운 영 성	SW의 운영에 관한 작업계획과 과정을 결정하는 SW의 특성들.	유 용 성
트 레 이 닝	현재의 작업계획(운영)이나 초기지식에서부터의 변이를 준비하는 SW의 특성들.	유 용 성
전 달 성	서로 유사하게 될 수 있는 유용한 입력. 출력들을 준비하는 SW의 특성들.	유 용 성
SW 시스템 독립성	SW 환경에서 그것의 의존도를 결정하는 SW의 특성들 (SW환경은 운영체제, 유틸리티, 1/O두린 등을 말함)	이 식 성 재사용효율성
기 계 독 립 성	SW 시스템에서 그것의 의존도를 결정하는 SW의 특성들.	이 식 성
전 달 공 통 성	인터페이스후린들과 표준프로토콜의 사용에 관한 제공하는 SW의 특성들.	결 합 성
데 이 타 공통성	표준데이타 표현의 사용을 제시하는 SW의 특성들.	결 합 성
간 결 성	코드의 양이 최소인 한기능의 수행에 관한 SW의 특성들.	유지. 보수성

1)McCall의 품질요인의 정의

- 안정성(integrity):

권한 없는 사람에 대한 소프트웨어 및 자료의 접근을 제어할 수 있는 정도로서, 프로그램이나 정보를 사용하고 검색하기 위한 견고성을 가지고 완전하게 보증한다.

- 유용성(usability):

프로그램의 교육, 조작, 입력 준비, 출력의 이해에 필요한 노력도로서, 사용법이 쉽고, 입출력이 용이하며 출력내용을 쉽게 이해할 수 있게 한다.

- 유지.보수성(maintainability):

프로그램내의 오류를 발견하고, 수정하는데 필요한 노력도로서, 에러수정을 편리하게 한다.

- 유연성(flexibility):

프로그램 변경에 소요되는 노력도로서 프로그램을 수정하여 새로운 시스템 요구에 적용할 수 있게 한다.

- 테스트 용이성(testing) :

기능 수행을 보증하기 위한 시험에 소요되는 노력도로서,소프트웨어의 품질을 향상시킬 수 있도록 테스팅을 한다.

- 이식성(portability):

프로그램을 다른 S/W, H/W 환경에 이식하는데 필요한 노력도로서, 하드웨어를 바꾸어서도 프로그램을 수행할수 있도록 한다.

- 재사용성(reusability):

프로그램을 다른 업무에 활용 가능한 정도로서, 프로그램의

· 표준화를 통해서도 모듈의 재사용이나 다른 페케 지에 적용할 수 있게 한다.

- 결합성(interoperability):

다른 시스템과의 결합에 소요되는 노력도로서, 인터페이스 기능을 강화하여 프로그램 및 시스템간의 결합을 쉽게 해준다.

- 정확성(correctness):

프로그램이 사양을 만족시키고 사용자의 목적을 만족시키는 정도로서, 소프트웨어가 사용자 요구와 그명세에 부합하거나 부합하도록 보완시킨다.

- 신뢰성(reliability):

요구된 정밀도로 소기의 기능 실행을 기대할수 있는 정도로서, 정확하고 예상된 기능을 발휘할 수 있도록 한다.

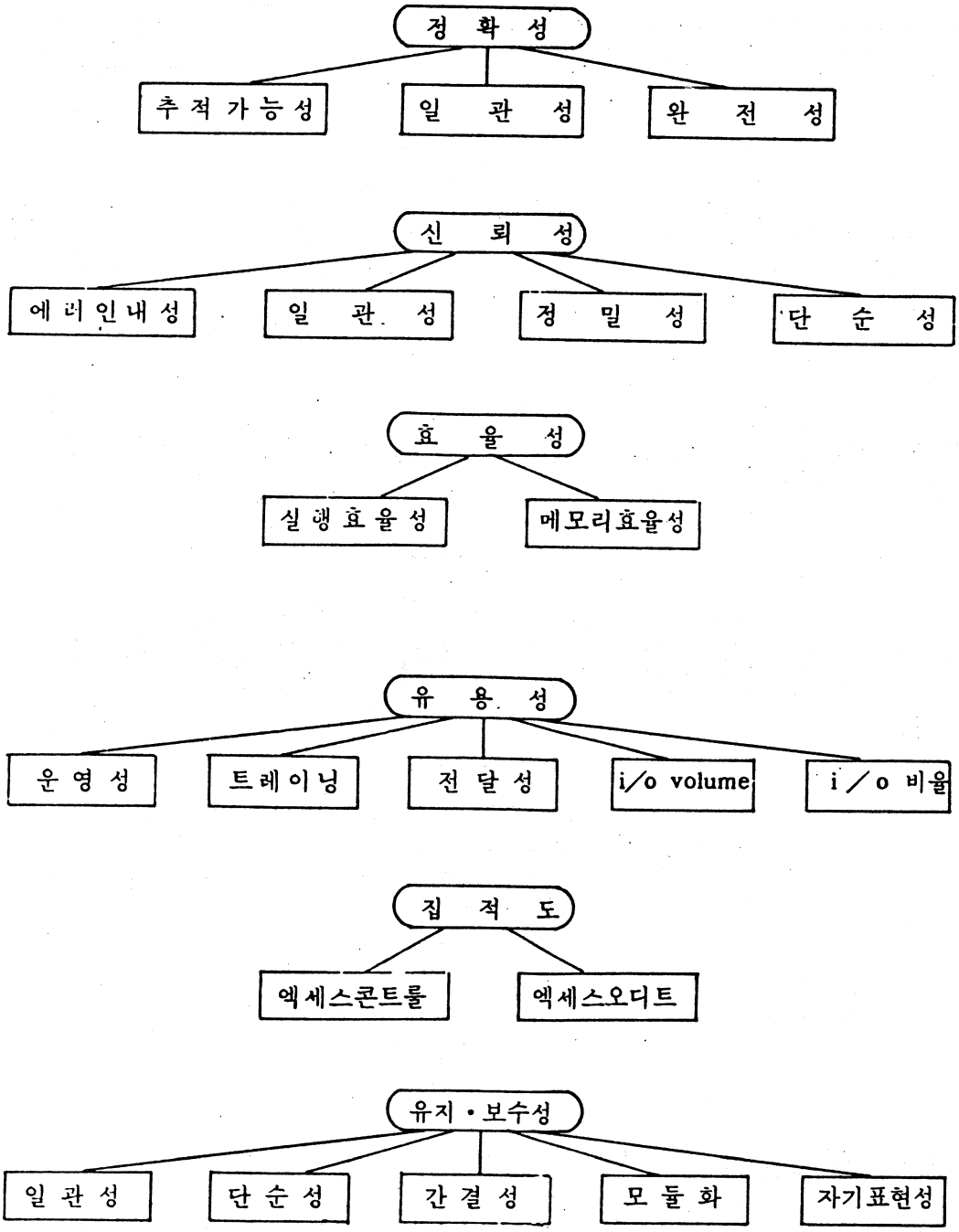
- 효율성(efficiency):

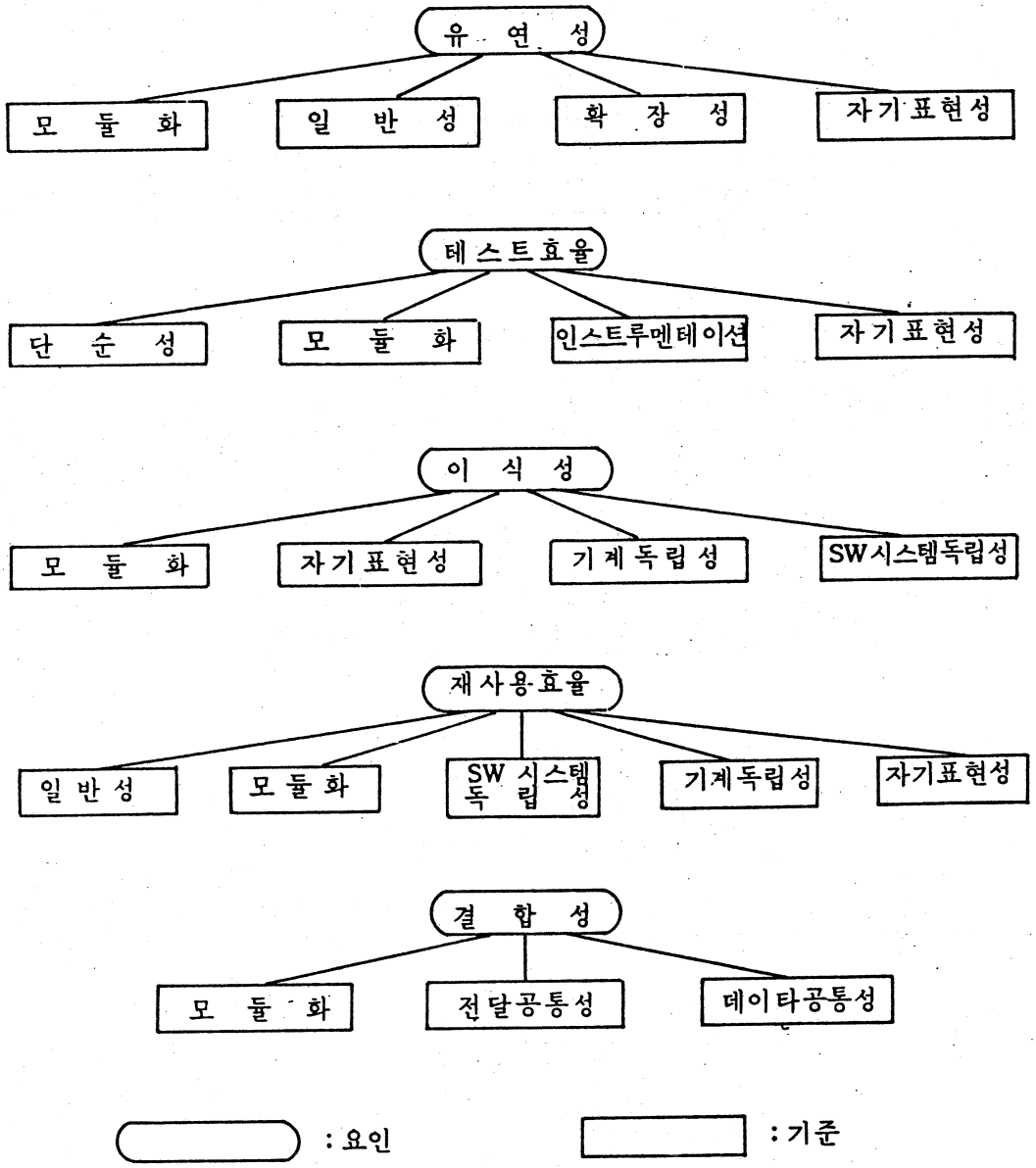
기능실행에 필요한 자원의 적절한 사용량으로서, 하드웨어를 포함한 시스템적인 입장에서 소프트웨어 제품이 충분한 기능을 발휘할 수 있게 해준다.

이와같은 품질요인은 품질보증을 위한 평가기준을 설정해 준다. 평가기준을 소프트 웨어의 독립적인 속성으로서 품질요인을 정의하고 요인들간의 관련성을 설명해 준다.

이것은 하나의 평가기준이 여러개의 품질요인에 관련될 수 있기 때문이다.

평 가 기 준


정 확 성
추 적 가 능 성
일 관 성
완 전 성
신 뢰 성
에 러 인 내 성
일 관 성
정 밀 성
단 순 성
효 율 성
실 행 효 율 성
메모리효율성
유 용 성
운 영 성
트 레 이 닝
전 달 성
i/o volume
i / o 비율
집 적 도
엑세스콘트롤
엑세스오디트
유지 • 보수성
일 관 성
단 순 성
간 결 성
모 듈 화
자기표현성



2.2 품질 목표 설정

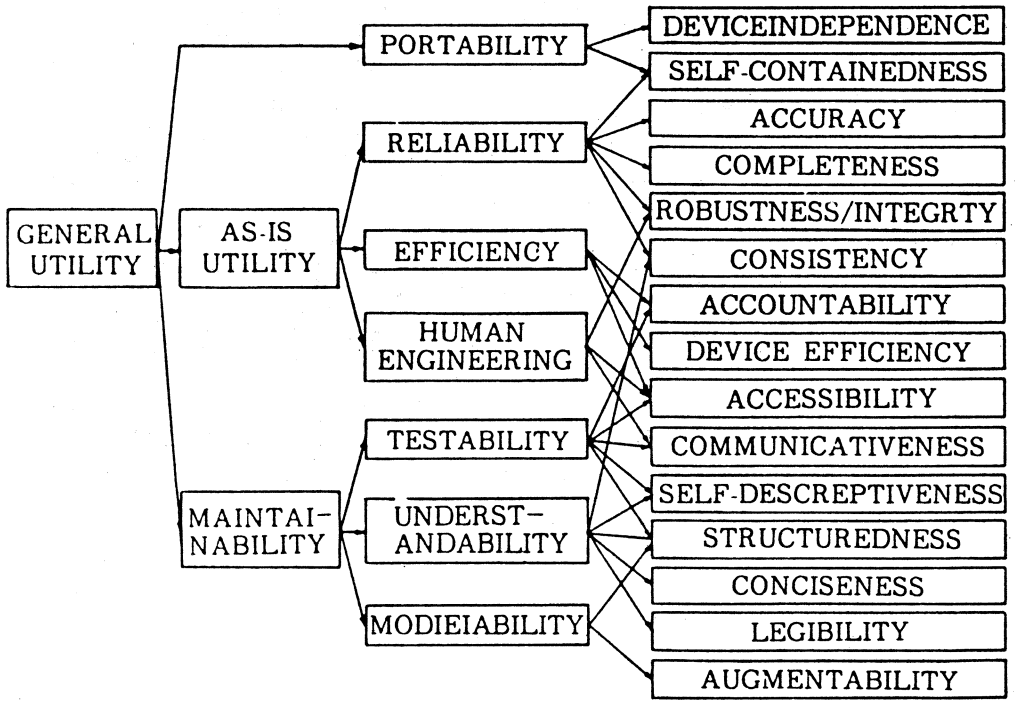
고품질 소프트웨어의 생산의 첫 단계는 프로젝트 관리자의 책임이며, 소프트웨어 라이프 사이클의 초기에 이루어져야 한다.

1) 시스템 특성에 대한 고려

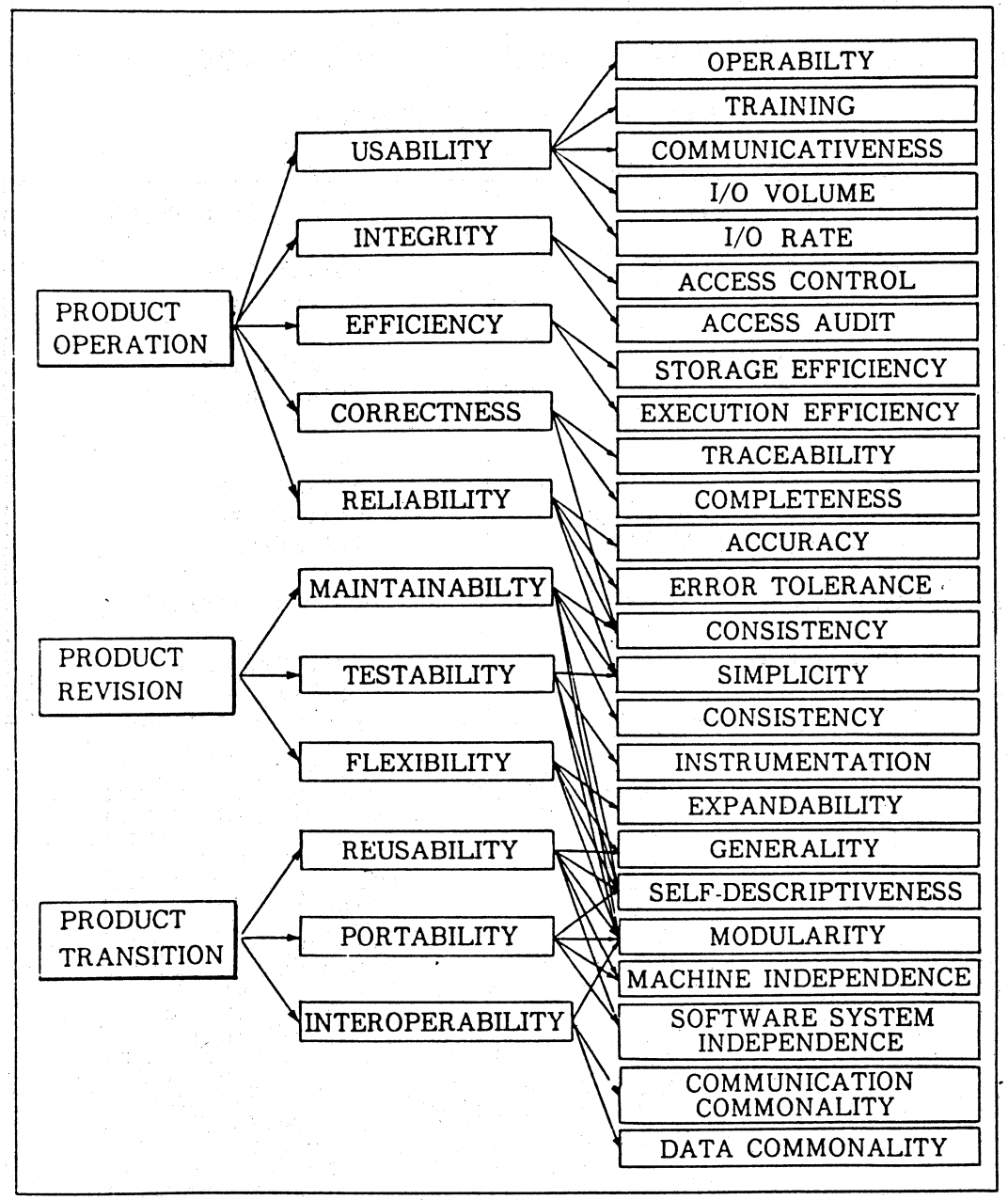
주어진 품질목표의 중요성은 그 시스템의 큰 부분을 차지한다.

시스템 특성 및 관련 품질목표

시스템 특성	중요한 품질 목표
. 인간 생활에의 영향	. **Reliability** . **testability**
. 긴 life cycle	. **Flexibility** . **Maintainability** . **Portability**
. 실시간 응용	. **Efficiency** . **Reliability**
. 수행될 정보가 분류됨	. **Integrity**
. 관계시스템과의 Interfacing	. **Interoperability**
. Interactive	. **Responsiveness** . **Usability**



Boehm의 품질 모델



McCall의 품질 모델

초기에 품질 목표들을 설정하려면,

(1) 주어진 시스템 특성을 읽고 적용될 필요가 없는 요인들은 삭제한다.

(2) 수행될 기능, 요구되는 시스템 성능 및 그 시스템의 기술적 요구사항 등에 기초하여, 적용될 다른 시스템의 특성을 추가한다.

(3) 품질요인들이 리스트를 사용하여, 각 시스템의 특성에 중요한 품질 목표를 확정한다.

(4) 품질목표 각각의 중요도를 표시한다.

2) Cost / Benefit Tradeoffs에 관한 고려

	개 발			평 가	개 발 후			비 고
라이프싸이클 요인 과정	요구 분석	설계	코드와 디버그	시스템 테스트	작동	보수 유지	변형	기대비용효과 : 투자비용
정 확 성	V	V	V	X	X	X		높 음
신 뢰 성	V	V	V	X	X	X		높 음
효 율 성		V	V		X			낮 음
완 전 성	V	V	V		X			낮 음
유 용 성	V	V		X	X	X		중 간
보수.유지성		V	V			X	X	높 음
테스트효율		V	V	X		X	X	높 음
유 연 성		V	V			X	X	중 간
이 식 성		V	V				X	중 간
재사용효율		V	V				X	중 간
결 합 성		V			X		X	낮 음

- V : 측정되어야 할 품질요소
- X : 실제로 나타나는 poor quality 영향

위 표는 정해진 소프트웨어 품질목표들과 라이프 사이클간의 관계 및

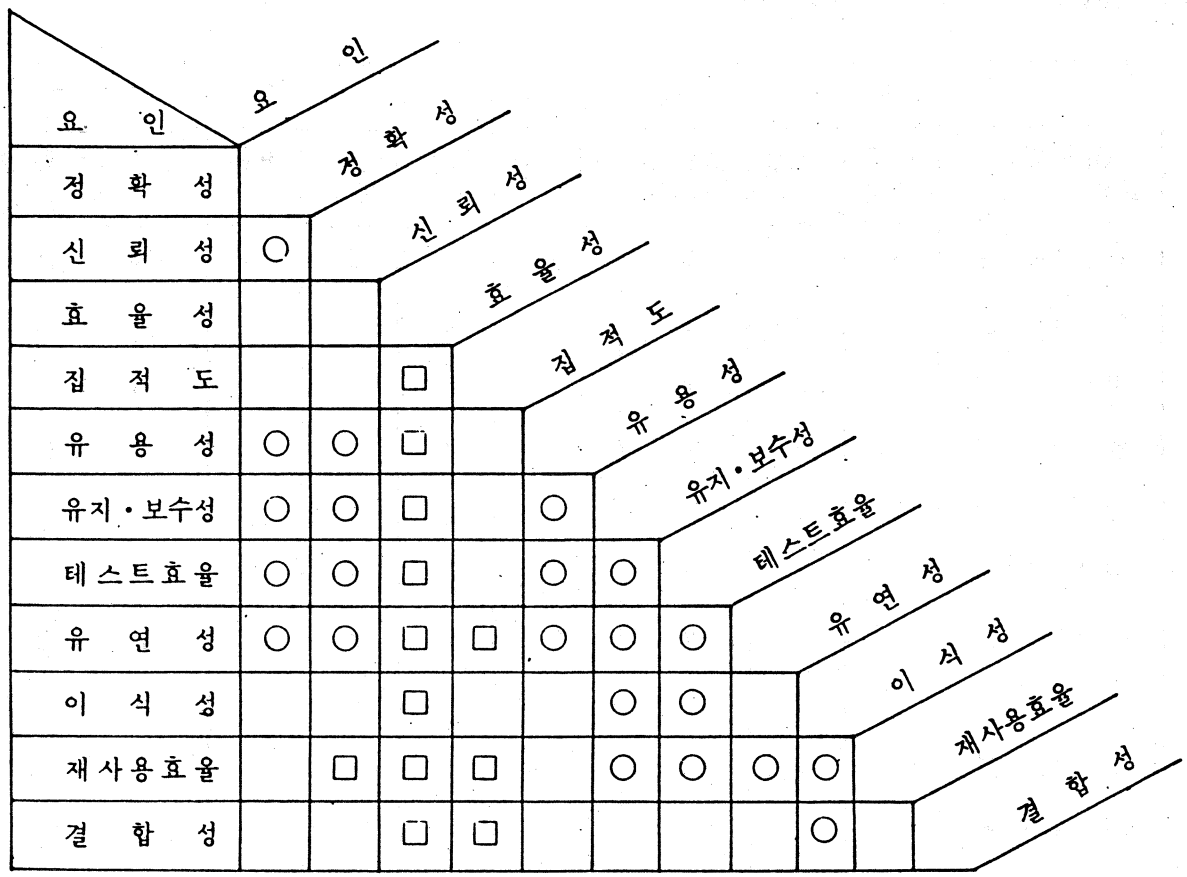
기대비용 효 과에 대한 내용을 나타낸 것이다.

위의 표로부터,

. 주어진 목표를 실행하는데 있어서의 실패효과와

. 각 품질 목표에 따른 비용의 기대효과를 알 수 있다.

3) 품질목표들 간의 Tradeoffs에 대한 고려



○ : 높은 관계 □ : 낮은 관계 또는 모순 blank : 무관계

이들의 관련성이 중요한 이유는 상충되는 목표들은 구현하는 것이 어려울 뿐 아니라 비용도 많이 들기 때문이다.

4) 지각 조사에 대한 행동

품질목표를 설정하는데 있어 필수적은 아니지만 유용한 것으로 시스템의 품질목표의 중요도에 따른 요원비치를 생각할 수 이다.

이들 인식은 시스템을 관리하고 개발하며 유지.보 수하고 사용하는데 배치될 요원들에 대한 조사를 통해 얻어질 수 있다.

조사될 요원들은 위에서 논의된 세가지 고려사항에 대하여 간결한 지시를 받거나 또는 그들 자신의 독자적인 기준에 근거하여 응답을 요청받을 수 있다.

5) 목표의 종결

품질목표를 설정하는 최종단계는 고려되는 시스템 특성, Cost / Benefit Tradeoffs 및 품질목표 Tradeoffs에 의해 수행된 결과로써 조사결과를 조정하는 것이다. 품질 목표는 1에서 11등급으로 나누어질 수 있고 프로젝트 관리자는 시스템 개발자에게 명백한 지침을 제시할 수 있다.

2.3 품질 체크리스트

체크 리스트들은 소프트웨어 개발의 각 과정에서 각각의 품질기준에 대한 구체적인 지침을 제공하게 된다.

다음의 단계들은 수행될 활동에 적합한 체크 리스트를 어떻게 선택할 것인가를 결정해주는 지침이다.

1) 프로젝트 각 단계에 따라 다음을 결정한다.

-요구사항 단계의 체크리스트 결정과 실행
-설계단계에서의 체크리스트 결정과 작성
-코팅단계에서의 체크리스트 결정과 작성
-테스팅 단계에서의 체크리스트 결정과 작성

2) 그 시스템에서 설정되어진 품질목표가 무엇인지를 찾아낸다.

3) 선택된 품질목표를 지원하는 품질기준을 결정한다.

4) 설립된 품질기준의 체크리스트를 설정한다.

요구사항 단계 체크리스트(예)

1. 일관성

점 검 항 목	평가				
	YES	NO (결합도)			비고
		상	중	하	
1. 시스템 기능을 설명하는 요구는 다른 것들과 일치하는가?					
2. 입력요구는 다른 요구와 일치하느가?					
3. 출력요구는 다른것들과 일치하느가?					
4. 데이타 베이스 요구는 시스템의 다른 요구와 일치하는가?					
5. 외부의 Interface 요구는 다른 요구와 일치하는가?					
6. 수행요구는 시스템의 다른 요구가 일치 하는가?					
7. 보안성 요구는 시스템의 다른 요구와 일치하는가?					
8. 애러 복구 요구는 시스템의 다른 요구와 일치하는가?					
9. 데이타 베이스에 관한 요구는 데이타 베이스 문서와 일치하는가?					

(2) 테스트 저장성

점 검 항 목	평가				
	YES	NO (결합도)			비고
		상	중	하	
1. 시스템에 대한 요구가 항목화 되어 있는가?					
2. 각 요구에 대한 검사는 가능한가? 즉 요구를 만족시키는지의 결정을 연구할 수 있는 검사와 방법이 정해져 있는가?					
3. 시스템의 수행여부를 Test하는 동안 주어진 시간에 발생하는 애러의 수에 대한 명시된 요구가 있는가?					

체크리스트들은 소프트웨어 개발자에 의해 여러 방향으로 사용될 수 있다.

- 작성한 문서 및 소스코드를 계획하는 것을 도와 준다.
- 작성중인 문서 및 코드들에 대한 남아 있는 문제들을 보조해 준다.
- 문서 및 코드에 대하여 Self-check 하도록 해준다.
- 문서 및 코드들의 Inspection 및 Walk-through 에 대한 지침을 제공한다.

소프트웨어 문서 및 코드를 계획하고 준비하여 재조사하는데 체크리스트들을 적절히 사용하여, 이들 생산물들이 선정된 품질기준을 만족하며, 궁극적으로 이미 설정된 프로젝트의 품질목표를 완수하게 된다.

참 고 문 헌

1. 奈良隆正他 ,"소프트웨어의 품질보증 활동",ENGINEERS,pp.6-11 ,1983.10
2. 野木兼六他, "소프트웨어 테스트 항목작성 지원 시스템",日立評論, Vol. 66, No.3, pp 29-32,1984.
3. 石井康雄, "소프트웨어의 검사와 품질보증", 日科技連出版社,1986.
4. 東基衛他, "소프트웨어의 품질계측/보증기술"(SQMAT),품질,Vol.16, No. 1, pp.79-84,1986.
5. Goel,A.I,"Software Reliability Models:Assumptions, Limitations and Applicability", IEEE Trans. on Software Engineering, Vol. SE-11, pp. 1411-1423, 1985.
6. 古賀惠子他,"소프트웨어 품질평가 시스템의 개발", ENGINEERS, pp. 18-21, 1986年 8月.
7. Curtis,Bill,"Measurement and Experimentation in Software Development", IEEE, 1981.
8. 石正康雄,"소프트웨어의 제조", 日科技連出版社
9. 梁 海述,"컴퓨터 프로그램의 품질보증 기준과 실제",한국정보산업협회,『情報 産業』,1988.11.
10. 梁 海述,"테스트 및 품질보증 기술의 현상과 과제", 컴퓨터 정보사, 『컴퓨터 월드』,1988.6.
11. 송 재형,"SOFTWARE 품질보증", 소프트웨어 엔지니어링, 산업경영정보연구원,1990.
12. 총무처,"S/W 품질보증",전산전문교육교재, 총무처 정부전자계산소, 1989.

Ⅱ. 소프트웨어 복잡도의 측정.평가 방법

1. S/W 품질을 측정하기위한 복잡도

최근에 소프트웨어의 다양한 특성을 측정 평가하기 위한 방법으로 프로그램의 복잡도(Program Complexity)가 정보과학 분야에서 중요한 문제로 다루어져 왔다 ([도 1] 참조).

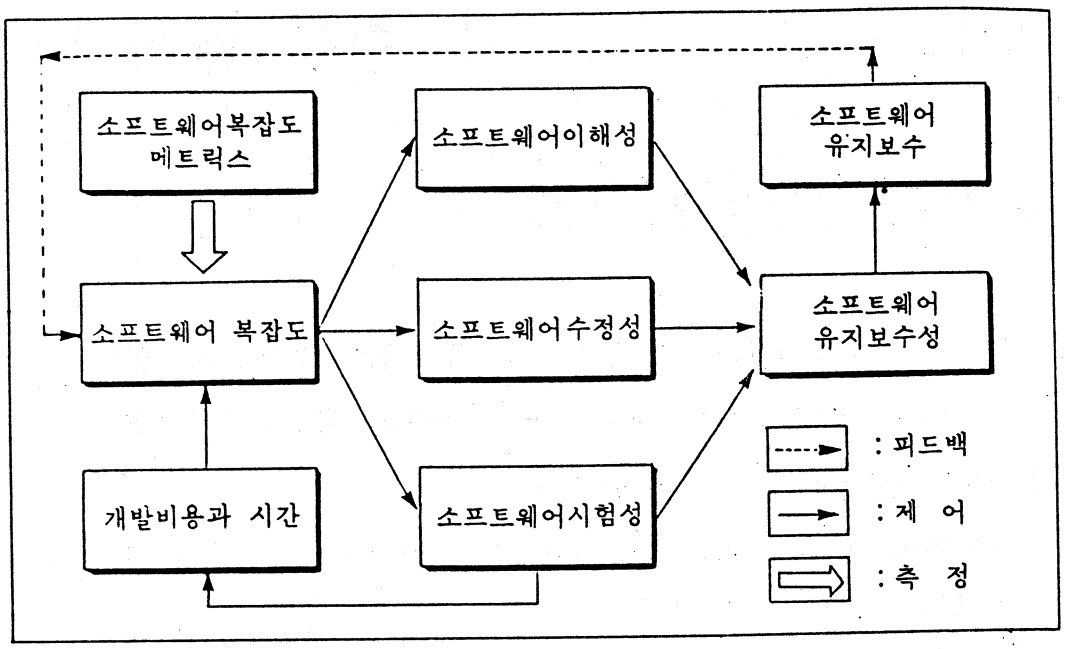
복잡도의 의미는 보는 관점에 따라 다양하나, 다음과 같이 정의 할수 있다.[7]

- 프로그램을 이해할 수 있는 정도.
- 프로그램의 오류를 수정, 유지할 수 있는 정도.
- 프로그램을 다른 사람에게 설명할 수 있는 정도.
- 프로그램을 변화시킬 수 있는 정도.
- 설계 명세에 따라 프로그램을 작성하는데 드는 노력의 정도 등으로 정리할 수 있다.

이러한 프로그램의 복잡도는 테스팅 노력의 스케쥴, 프로그래밍 시간의 측정, 프로그래머의 디버깅 능력 및 소프트웨어의 생산성 평가, 프로그램의 비구조도(Unstructuredness) 측정, 신뢰성(Reliability)과 유지보수성(Maintainability), 소프트웨어 개발 요구사양에 대한 예측, 자동관리제어(Automatic Management Control)등의 분야에 응용되고 있다.

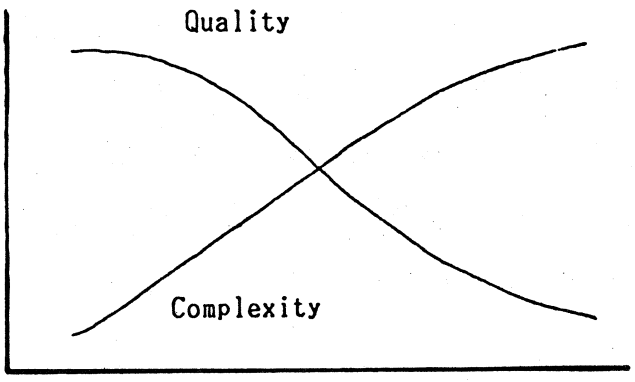
1.1 소프트웨어의 품질향상을 위한 복잡도 측정

복잡도는 IEEE 용어집에 복잡도란,"데이타 구조의형태, Nesting된 정도,



〔도 1〕 소프트웨어 복잡도의 중요성

조건 분기(condition branches)의 수(number)와 복잡성, 인터페이스의 수와 복잡성, 그리고 다른 시스템 특징(characteristics)등과 같은 요소로서 결정되는 시스템이나 시스템이나 시스템요소의 복잡한 정도를 말한다."



[도 2] 소프트웨어 품질과 복잡도의 상호관계

앞에서 정의된 것과 같이 소프트웨어의 복잡성은 소프트웨어의 품질과 밀접한 관계가 있다. [도 2]는 소프트웨어의 복잡성과 소프트웨어의 품질과의 상호관계를 나타내고 있다. 그림에서 나타난 바와같이, 소프트웨어의 복잡도가 증가할수록 품질이 떨어지는 것을 알 수있다.

일반적으로 개발되어진 소프트웨어의 복잡도 메트릭스의 소프트웨어 라이프 사이클에 대한 적용의 유용성은 다음과 같다.

첫째, 만약 시스템 설계단계에서 만들어진 명세(specification) 로부터 복잡도 측정이 가능하다면, 시스템 설계자는 그 메트릭스로써 막대한 비용으로서 시스템의 구현이 되기전, 품질이 떨어지는 시스템 요소나 설계를 찾아서 변경시키거나, 대체를 가능하게한다.

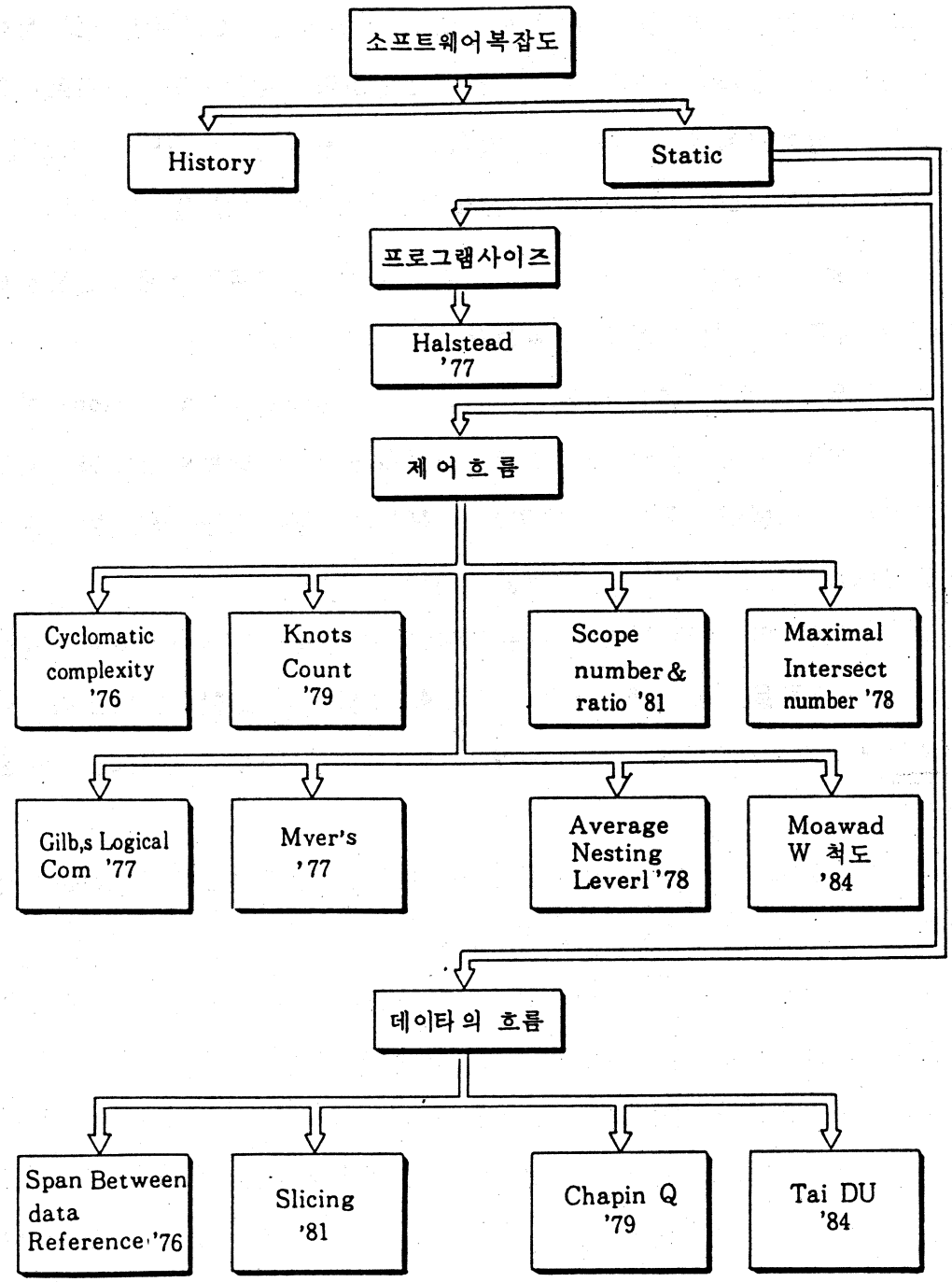
둘째, 프로젝트 메니저는 소프트웨어 복잡도 메트릭스로써 생산될 소프트웨어 제품을 좀더 잘 평가, 관리할 수있다. 또한 그 메트릭스로써 프로젝트에서의 스케쥴링, 비용등의 할당같은 의사 결정에 도움을 줄것이다.

세째, 시스템의 구현자는 소프트웨어 복잡도 메트릭스를 툴(tool)로써 품질좋은 코드 생성에 도움을 줄 것이다.

네째, 시스템의 테스터(tester)는 그 메트릭스로써 어떤 부분이 좀더 테스팅이 필요한 가를 결정하는 데 유용할 것이다.

마지막으로, 소프트웨어 복잡도 메트릭스는 유저와 시스템 개발자에게 시스템의 품질에 대한 좀더 정확한 요구(requirment)를 만들어 줌으로써, 그요구가 엄격히 수행되었는지를 정량적으로 제시해 줄 수있다. 위와 같은 유용성으로 소프트웨어 복잡도 측정도은 소프트웨어 품질을 보증시켜 주는 중요한 방법론(methodology)이 되는 것이다.

또한 일반적으로 프로그래머의 관점에서 볼 때 소프트웨어의 품질을



〔도 3〕 소프트웨어 매트릭스의 분류

나타내는 프로그램의 복잡도는 다음과 같이 분류할 수 있다.

1) 프로그램 사이즈에 기반을 둔 복잡도
2) 제어의 흐름에 기반을 둔 복잡도
3) 데이타 흐름에 기반을 둔 복잡도(데이타 구조 포함)
4) 혼합적 방법에 의한 복잡도

이와같은 네가지 관점으로 지금까지 [도 3]과 같은 많은 복잡도 척도가 제안되고 있다.

그러나 여기에서는 가장 신뢰할 만한 척도들을 중심으로 그 적용방안 과 문제점을 살펴보기로 한다.

2. 프로그램의 사이즈에 의한 복잡도 측정방법

프로그램의 복잡도 측정에는 보통 다음과 같은 척도(scale) 가 채택된다.

1) Norminal 2) Oridinal 3) Interval 4) Ratio scale

1) Norminal 척도 방법을 예로 들어 설명하면 프로그램을 이해하기 "어렵지 않다"," 약간 어렵다"," 어렵다" 등으로 분류하는 것이다.

2) Oridinal 척도 방법을 예를 들어 설명하면 프로그램 A,B,C 모두가 " 약간 어렵다" 말할 수 있을 뿐 아니라 B가 A보다 "조금 더 어렵다" 라고 말할 수 있는 것이다.

3) Interval 척도로서는 가령 "프로그램 A가 B보다 10 단위만큼 어렵다"

라고 말할 수 있는 것이다.

4) Ratio 척도로써는 두 프로그램을 비율로서 나타낼 수 있다. 가령 "A가 B 보다 2배 어렵다." 등으로 나타낼 수 있다.

Ratio 척도는 융통성(flexible)이 있었으나 실제적(pratical)이지 못하고 Interval 척도는 단위(unit)의 정의를 필요로 한다. 일반적으로 Ordianl 척도가 측정하기 편리하며, 실제적이어서 소프트웨어 복잡도 측정 척도에 가장 적합하고 좋은 척도라고 한다.

소프트웨어 복잡도 측정의 가장 기본적인 측정이 프로그램의 크기에 기반을 둔 것이다. 프로그램의 크기 측정은 계산하기 쉽고 여러경우에 적용가능하다. 이런 프로그램 크기에 기반을 둔 것중, 대표적인 것이 라인의 수(Lines of code) 와, Halsted의 소프트웨어 사이언스이다[13].

(1) 라인의 수(Lines of code)

프로그램의 라인의 수로써 소프트웨어 복잡도를 측정하는 것은 다음과 같은 실험적 결과를 제공하고 있다. 즉 프로그램이 클 수록 작은 프로그램보다 유지비용이 증가한다는 것이다. 그러나 다른 성격을 가진 프로그램을 단지 라인의 수로써 복잡도를 측정한다는 것은 문제가 있다.

(2) 할스테드의 소프트웨어 사이언스 공식

현재, Halstead 의 이론, 즉 Software science 는 소프트웨어 복잡도 메트릭스 중 가장 신뢰받을 만한 측정으로 주목 받고 있다. 학계와 산업계의 연구소에서 soptware science 의 적용과 검증을 한 결과, 놀랍게도 이 메트릭스는 정확성이 뛰어 난것으로 알려졌다[13]. 이 메트릭스는 프로그램의 연산 자(oprerator) 와 피 연산자(operand) 측정에

기반을 두고 있다. 이러한 프 로그램을 작성하는데 요구 되는 시간과 정신적인 노력을 예측하는 데에도 상당 한 관련이 있음이 밝혀졌다.

어떤 프로그램의 soptware science 메트릭스는 다음과 같은 4가지 요소의 계 산으로 부터 유도 된다.

n1: 프로그램에서의 구별되는(unique) 연산자의 수

n2: 프로그램에서의 구별되는(unique) 피연산자의 수

N1: 프로그램에서의 총 연산자의 수

N2: 프로그램에서의 총연산자의 수

이러한 4지 요소로부터 다음의 Halstead의 software science 방정식이 유도 된다.

Vocabulary : n1 = n1 + n2

Length : N1 = N1 + N2

Estimated length : N = n1log n1 + n2log n2

Volume : V = Nlog n

Level : L = v*/V

Program effort : E = V/L

● Vocabulary , n은 프로그램의 구별되는 연산자와 피연산자로써 구성된다. 큰 Vocabulary 값은 프로그래머가 프로그램 Code 를 이해하는 데 좀더 많은 word 를 알아야 한다는 것을 의미한다. 프로그램에서의 문제는 Vocabulary 가 프로그램마다 개성에 따라 달라질 수 있다. 이런 의미에서 자료 사전 (data dictionary)의 사용은 유용한 것이다. 프로그래머는 프로그램 개발을 위해 Vocabulaty 를 피연산자 구성에 참조하여 사용할 수 있다.

```
      SUBROUTINE SORT (X, N)
      DIMENSION X(N)
      IF (N .LT. 2) RETURN
      DO 20 I = 2,N
         DO 10 J = 1,I
           IF (X(I) .GE. X(J))GO TO 10
           SAVE = X(I)
           X(I) = X(J)
           X(J) = SAVE
   10 CONTINUE
   20 CONTINUE
         RETURN
         END
```

	Operator	Count
1	End of statement	7
2	Array subscript	6
3	=	5
4	IF ( )	2
5	DO	2
6	,	2
7	End of program	1
8	.LT.	1
9	.GE.	1
$n_1 = 10$	GO TO 10	1
		$28 = N_1$

	Operand	Count
1	X	6
2	I	5
3	J	4
4	N	2
5	2	2
6	SAVE	2
$n_2 = 7$	1	1
		$22 = N_2$

begin

i := 1 ;

1 : if A [i] ≦ A [i + 1] then goto 2 ;
swap (A [i] , A [i + 1]) ;
if i = 1 then goto 1 ;
i := i − 1 ;
goto 1 ;

2 : i := i + 1 ;
if i ≠ n then goto 1
end

長円：operator begin と end の対，:=，;，[と] の対，if，≦，+，then，goto 2，swap，(と) の対，,，=，goto 1，−，≠

$\eta_1 = 16 \quad N_1 = 41$

正方形：operand i，1，A，n

$\eta_2 = 4 \quad N_2 = 22$

$\eta = \eta_1 + \eta_2 = 20$

$N = N_1 + N_2 = 63$

$V = 63 \log_2 20 = 63 \times 4.32 = 272.2$

$L = \frac{2 \times 4}{16 \times 22} = 0.02$

$\lambda = L^2 \times V = (0.02)^2 \times 272.3 = 0.11$

$E = \frac{V}{L} = \frac{272.3}{0.02} = 13610$

$T = \frac{E}{s} = \frac{13615}{18} = 756.1$

● Volume ,V 은 프로그램 Vocabulary 에서 N items 의 독특한 Designator를 제공하는 데 필요한 비트의 수로 측정된다. Volume 은 가장 함축적인 설계 구현할 수 있는 크기이다.

● 프로그램 effort , E 는 N 과 N2 보다 프로그램 이해성 관점에서 좀더 좋은 측정이다. 여러 프로그래밍의 검증 결과 이 프로그램 effort 는 실제 프로그래밍 노력과 거의 밀접하게 일치되는 것이 발견되었다.

(3) 지프의 법칙

통계적인 자연언어의 연구를 통해 지프(Zipf)는 영어, 중국어, 라틴어 를 중심으로 사용빈도가 높은 순으로의 단어 서열과 사용빈도 사이의 관 계를 다음과 같이 표현하였다.

fr * r * 2 = Constant = C . . . (1)

여기서 단어의 종류 r의 절대 빈도를 nr, 전체빈도를 n이라 하면 fr은 nr/n으로 나타나는 상대빈도이다. a = 1 일 때 식(1)은 다음과 같이 쓸 수 있다.

fr * r =10 . . . (2)

식(2)가 바로 지프의 법칙이며, t개의 단어의 종류가 있다고 가정하면 다음과 같은 공식을 유도할 수 있다.

n = t(0.5772 + 1nt) . . ..(3)

여기서 n은 프로그램의 길이이다.

(3) 위 공식들의 문제점

프로그램 사이즈에 기반을 둔 복잡도는 프로그램의 라인수와 명령문 및

연산자의 수 등에 의해 복잡도를 측정하는 방법이다.

이 방법은 사이즈가 긴 프로그램이 짧은 프로그램에 비해 반드시 더 복잡하다고 말할 수 없다는 약점을 가지고 있다. 이를 좀더 구체적으로 살펴보면 다음과 같다.

- 실제로 대규모의 프로그램에서 적용이 불가능하다
- 대상 언어에 따라 다르며 특히 파스칼 프로그램에 적용하기에 부적합하다.
- 수치해석등과 같은 기술계산 문제에 부적당하다.
- 입력변수, 출력변수, 매개변수는 오퍼랜드로서 설계 초기 단계에는 예측이 가능하나 오퍼레이터 타입의 예측은 어렵다.

3.제어의 흐름에 의한 복잡도 측정방법

(1) 맥케이브의 Cyclomatic 척도

프로그램의 복잡도를 양자화하고자 하는 많은 제안들 중에서 프로그램 의 논리적인 복잡도를 나타내주는 척도는 맥케이브(McCabe)의 Cyclomatic 수가 대표적이다.

이 척도는 일반적으로 프로그램에 내재되어 있는 제어흐름의 전체 경로(Paths)를 계산할 수 없기 때문에, 선형 조합으로서 프로그램내에 가능 한 모든 경로들을 만들어 이 기본 경로들의 수를 의미한다.

따라서 맥케이브의 Cyclomatic 수 척도는 하나의 프로그램 P에 대해 그래프 G = (V, E)를 만든다. 여기서 V는 정점(Node)으로서 프로그램의 각

명령문에 해당되고, E는 에지(Edge)로서 프로그램의 제어흐름에 해당 한다.

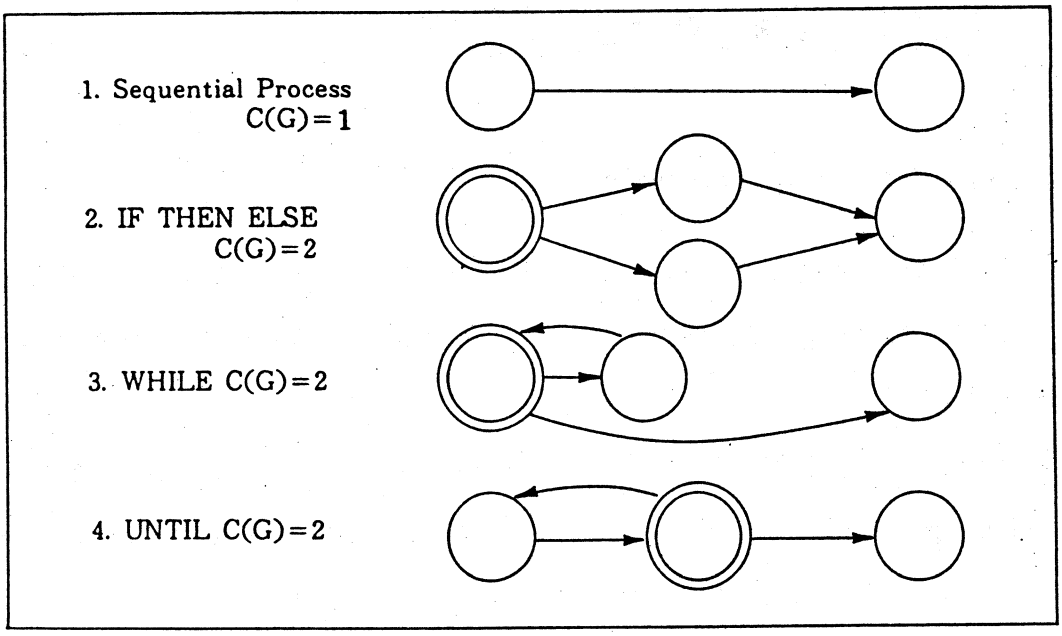
이렇게 프로그램으로부터 변환된 제어흐름 그래프 G상에서 Cyclomatic 수는 C(G) =e-n +2P로 계산한다.

여기서 e는 에지수, n은 정점의 수, P는 연결성분의 수이며 2는 그래프의 다면체 공식에서 구한 값을 적용한 것이다.

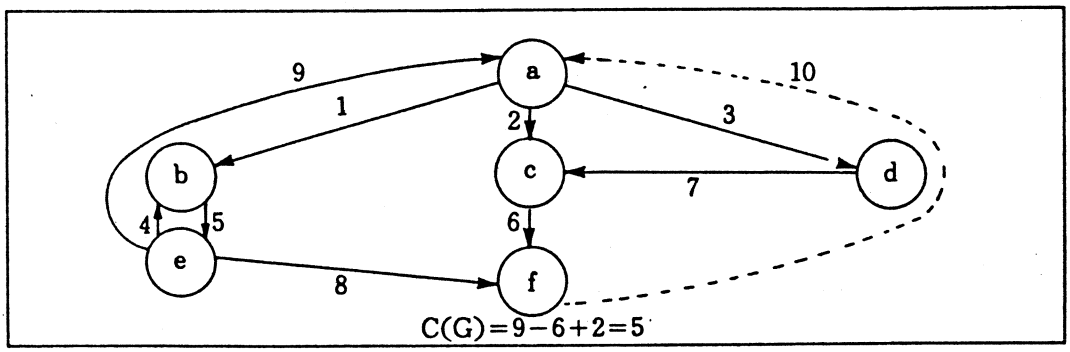
또한 맥케이브는 P=1인 경우 다음 식과 같이 프로그램내의 프레디케이트(Predicates)의 수에 1을 더한 값과 같다는 것을 보임으로써 복잡도를 쉽게 계산할 수 있게 하였다.

C(G) = 프레디케이트의 수 + 1

[도 4]는 제어 흐름그래프에서 처리구조에 따라 오퍼레이터를 분류한 것이며 [도 5]는 맥케이브의 Cycolmatic 척도의 예이다.



〔도 4〕 제어구조의 제어흐름 그래프



〔도 5〕 C(G)=5인 제어흐름 그래프

제어흐름 그래프에 대한 기본경로

기본경로	1	2	3	4	5	6	7	8	9	10
(abefa)	1	0	0	0	1	0	0	1	0	1
(beb)	0	0	0	1	1	0	0	0	0	0
(abea)	1	0	0	0	1	0	0	0	1	0
(acfa)	0	1	0	0	0	1	0	0	0	1
(adcfa)	0	0	1	0	0	1	1	0	0	1

(2) 모아워드의 W척도

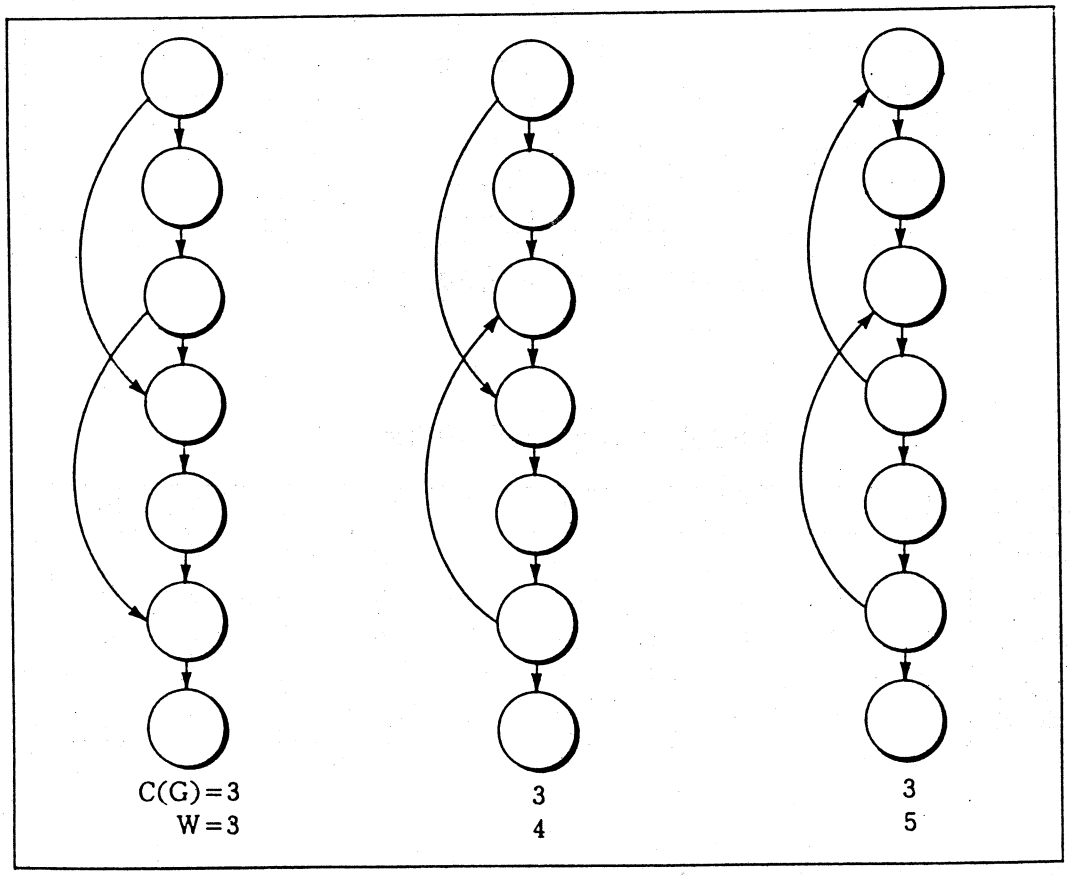
모아워드(Moawad)가 제안한 W수 척도는 맥케이브의 척도를 좀더 개 선한 방법이다. 이것은 프로그램내에 루프가 복잡도에 가장 많은 영향을 미치는 점에 착안하여 루프(Loop)의 수를 복잡도에 반영시켰다.

따라서 W수 척도는 제어흐름 그래프에서 구한 Cyclomatic 수에 루프 의 수를 더한 값으로 다음과 같이 프로그램의 복잡도를 표현하고 있다.

W =C(G) +루프의 수

그러나 W수 척도는 Cyclomatic 수 척도가 가지는 문제점을 완전히 해결하지는 못했으며, 특히 루프의 반복회수나 사이즈가 복잡도에 반영되지

않고 있음을 알 수 있다. [도 6]은 W 척도의 예이다.



〔도 6〕 제어흐름 그래프에서 W척도의 예

(3) Gilb'의 논리적 복잡도 메트릭스

Gilb은 논리적 복잡도(logical complexity)를 프로그램에서 얼마나 많은 dicision -making logic이 있는 지의 측정으로써 정의하였다. 그는 두가지 메트릭스를 제안했는데, 하나는CL 로써, 절대적 논리적 복잡도 (absolute logical complexity) 이고, 또 하나는 cL로써 기존의 프로그램 크기를 고려하 였고, 그곳에는 제어 메트릭스 관점이 들어가 있다.

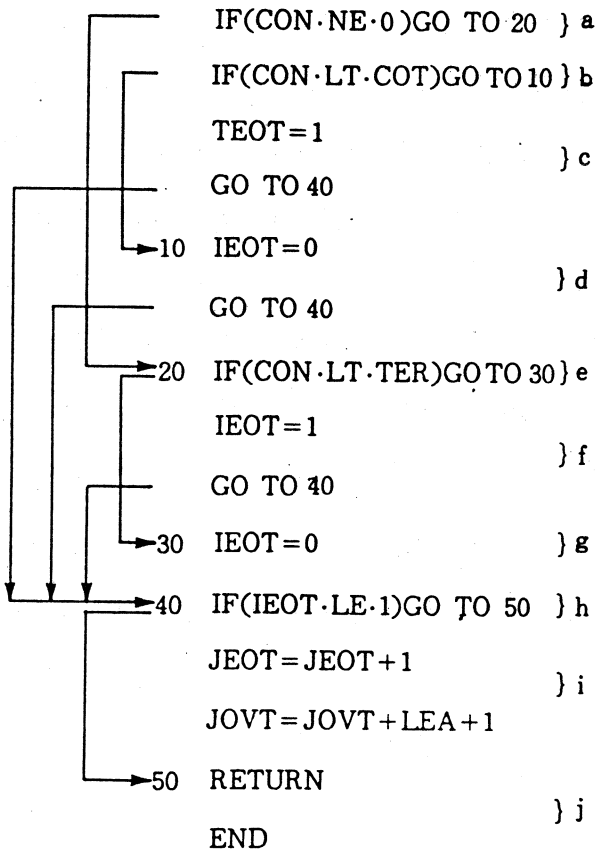
(4) 우드워드의 Knot 척도

우드워드(Woodward)는 프로그램 텍스트에서 제어전달의 수보다는 실제적인 위치에 근본을 두는 척도를 제안하였다.

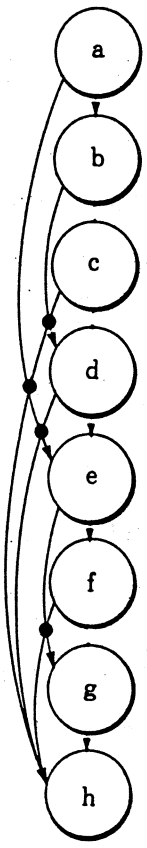
이 척도는 모든 정점(내부에 제어 전달이 없는 문들의 연속)들로부터 제어전달의 도달점까지 에지를 그렸을 때 이들 에지들의 피할 수 없는 교차점(Knot)들의 수를 측정함으로써 프로그램의 복잡도를 판정하는 방법이다.

이 방법은 [도 7]과 같은 원시 프로그램을 블럭화시킨 다음에 블럭화 된 프로그램의 제어흐름 그래프 [도 8]에서 교차점 수를 측정하는 방법 이다.

따라서 [도 8]은 4개의 교차점수를 가지는 프로그램 임을 알 수 있다.

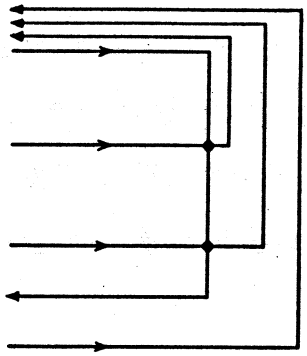


〔도 7〕 원시 프로그램의 블럭화



〔도 8〕 4개의 교차점을 갖는 제어흐름 그래프

```
    begin
    i:=1;
1:  if A[i]≤A[i+1] then goto 2;
    swap(A[i], A[i+1]);
    if i=1 then goto 1;
    i:=i-1;
    goto 1;
2:  i:=i+1;
    if i≠n then goto 1
    end
```



(5) 스코프 메트릭스

스코프 메트릭스는 제어 흐름을 측정하는 다른 메트릭스와 같이 프로그램을 flow graph 와같이 그래프이론적 표현에 기반을 두고있다[15]. 스코프 메 트릭스를 측정하기 위하여 그래프중 부그래프(subgraph)G' 를 만들고, 이 그 래프는 주어진 선택 노드(decision node) 에 바로 이어진 모든 node 로써 구 성된다. 즉 부프로그램의 선택 노드로부터 하나의 에지(edge)에 의해 연결된 모든 노드로서 구성된것이다. Outdegree 0나 1을 가진 노드는 Receiving 노드이다. 1보다 큰 Outdegree를 가진 것은 선택노드에 모두 이어지는 "Lower bound"노드이다. 모든 다른 Lower Bound node 보다 앞선 Lower bound 노드가 Great Lower Bound(GLB)노드이다. GLB에 앞서고 선택 노드가 뒤이어 있는 노드의 수 더하기 1을 한값이 선택노드의 Adjusted Complexity(AC) 값이다.

스코프 메트릭스값은 각 노드의 AC 를 합한 것이 된다. 스코프 비율

메트릭스(Scope ratio metrics ,SCORT)는 다음과 같이 정의된다.

SCORT = (1.0 - N/scope)* 100%

여기서 N은 terminal 노드를 제외한 flow graph 내의 노드의 수를 나타낸다. SCORT 는 복잡도가 증가할수록 100%에 가까이 접근한다.

4. 데이타의 흐름에 의한 복잡도 측정방법

(1) Chapin의 Q척도 측정방법

이 방법은 데이타의 아이템이 어떻게 사용되어지는지에 기반을 둔 측정이다. 데이타의 부류는 다음과 같이 4가지로 나눈다.

1. P 데이타 부류 : 세그멘트를 배출하는 데 필요한 입력 데이타.
2. M 데이타 부류 : 세그멘트내에 생성되거나 변화하는 데이타.
3. C 데이타 부류 : 세그멘트내에 "Controlling"에 사용되는 데이타.
4. T 데이타 부류 : 세그멘트내에 변화되지 않고 사용되는 데이타.

즉, Chapin은 각 데이타의 사용이 모듈에 각각 다른 비중을 갖고 복잡도에 기여한다고 생각했다. 각 모듈의 복잡도는 Q = R * W'로써 계산했으나 여기서 W'는 비중 값으로서,

C 데이타 : 3

M 데이타 : 2

R 데이타 : 1

T 데이타 : 0.5

값을 갖는다고 정의하였다. R은 반복-Exit 요소라고 정의하였으며, 프로그램의 전체 복잡도는 각 모듈의 복잡도를 전체 더하여 계산한다.

(2) Henry 의 정보 흐름(imfomation flow) 척도

이 방법은 시스템 요사이에서의 정보흐름에 기반을 둔 측정이다. 이 메트릭스는 다음과 같이 측정된다[12].

Length * (fan -in * fan -out)**2

Length는 원시 코드의 라인 수이며, fan -in 이란 가령 A라는 프로시듀어가 있을 경우, 이 프로시듀어에 들어오는 정보와 이 프로시듀어에서 정보를 retrieve 하는 수를 더한 값이된다. fan-out 은 이와는 달리 프로시듀어 A에서 다른 프로시듀어로 흘러나가는 정보와 이 프로시듀어에서 update 가 일어나는 수를 더한 값이된다.

이 외에도 데이타 흐름과 제어 흐름관점을 같이 고려한 Ovido의 복잡도 모델과 제어 흐름과 제어관점을 고려한 Hansen 의 복잡도 측정 방법등이 있다.

(3) D 척도

문헌[21]에서는 확장된 제어흐름 그래프에서 대입과 참조를 이용하여 테이타 흐름정보에 기반을 둔 복잡도를 제안하였다.

즉, 확장된 제어흐름 그래프상에서 변수 x가 참조되는 위치 q에 이르는 경로를 의미한다.

예를 들면, [도 9]의 (1)과 같은 IF구조에서 어떤 변수 x의 대입이 블럭내에 새로 대입되지 않으면 입력대입은 출구에서 계속 살아있게 된다.

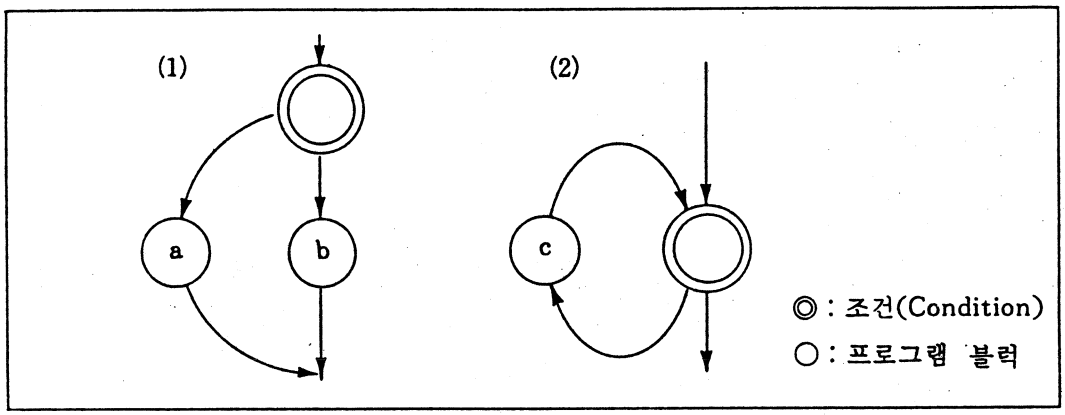
a와 b 모두 x가 대입되면 이 두 대입만이 출구에서 살아있게 된다. 또 a,b중에서 어느 한쪽에서만 x가 대입되면 이 대입과 초기의 입력 대입이

출구에서 살아 있게 된다.

따라서 IF 구조는 입력대입의 수에 1개의 대입를 추가할 수 있으며, IF 구조안에서 생성될 수 있는 출력대입의 수는 2개 이하이다.

[도 9]의 (2)와 같은 WHILE 구조의 예를 보면, 블럭 C에서 x가 다시 대입되지 않으면 모든 입력대입은 출구에서 살아있게 된다.

또, C가 x의 대입을 가진다해도 루프 본체를 실행하지 않고도 WHILE



〔도 9〕 IF 구조와 WHILE 구조의 제어데이타 그래프

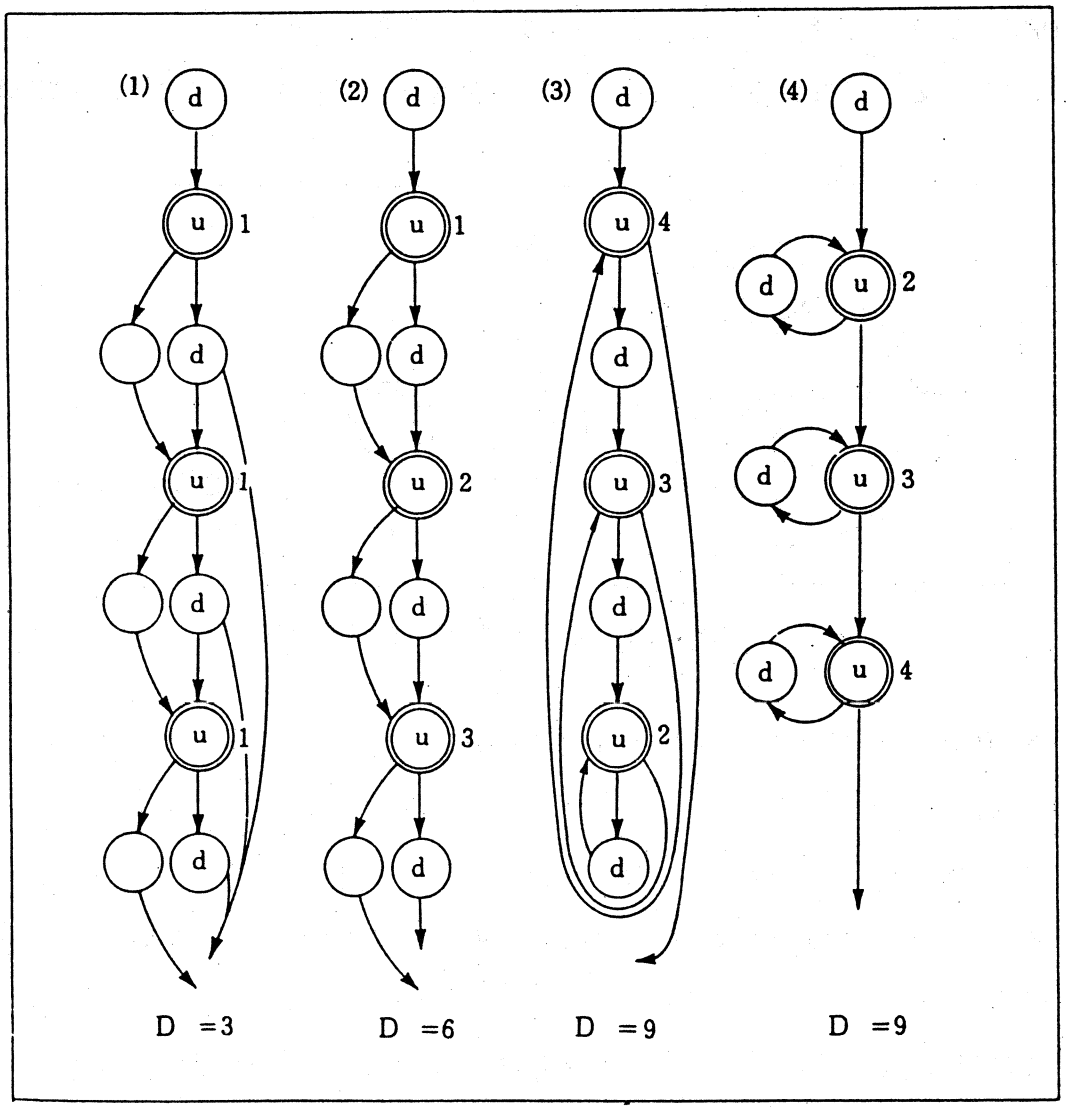
구조를 떠날 수 있기 때문에 이 대입과 모든 입력 대입은 출구에서 살아 있게 된다.

따라서 WHILE 구조에서도 x에 대한 대입정의의 수보다 1개 더많은 출력대입의 수를 가질 수 있고, 그중에서 WHILE 구조안에서 생성될 수 있는 것은 1개 이하이다.

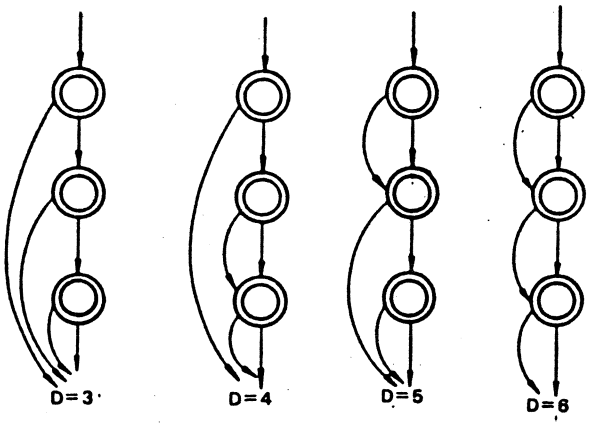
문헌[21]에서는 이와 같은 내용들을 임의의 그래프에 적용시켜 대입-참조(D) 의 전체수로써 프로그램의 복잡도를 정의하였다.

[도 10]는 문헌[21]의 D척도를 이용하여 프로그램의 복잡도를 측정한 예이다.(여기서, d는 대입, u는 참조을 나타내며, u옆의 수는 참조할 때의

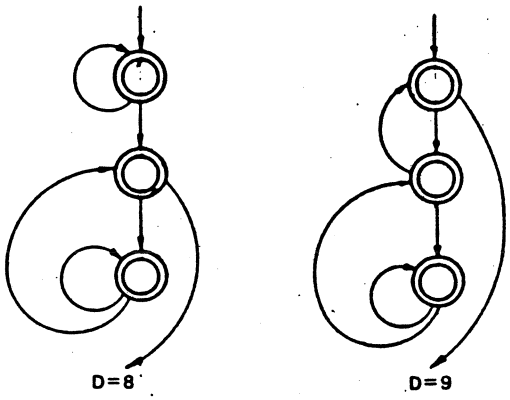
살아있는 대입의 수이다)



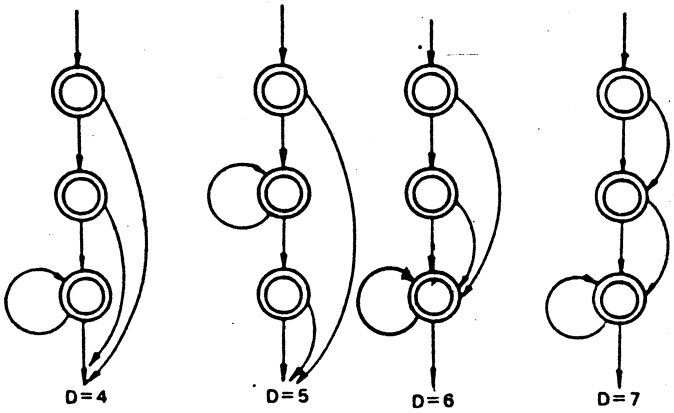
〔도 10〕 3개의 제어구조를 가지는 제어 데이타 그래프의 예



3개의 IF구조에 대한 제어 DATA
그래프의 계산 예



3개의 WHILE구조에 대한 제어 DATA
그래프의 계산 예



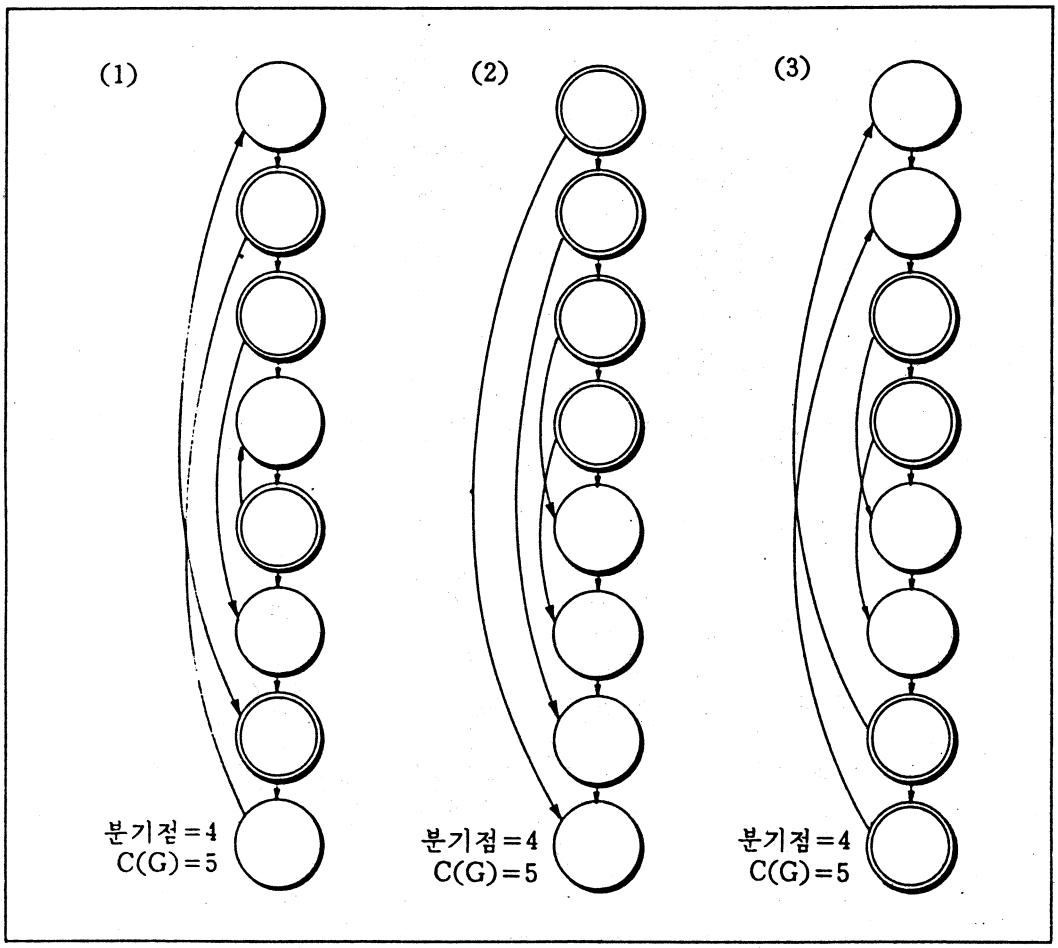
2개의 IF구조와 1개의 WHILE구조에 대한
제어 DATA 그래프의 계산 예

5. 복잡한 척도의 문제점

(1) 맥케이브의 Cyclomatic 척도

Cyclomatic 척도는 어떤 그래프에 상관없이 사용할 수 있고, 계산이 간단하며 프로그램에서 바로 구할 수 있다는 특징이 있다.

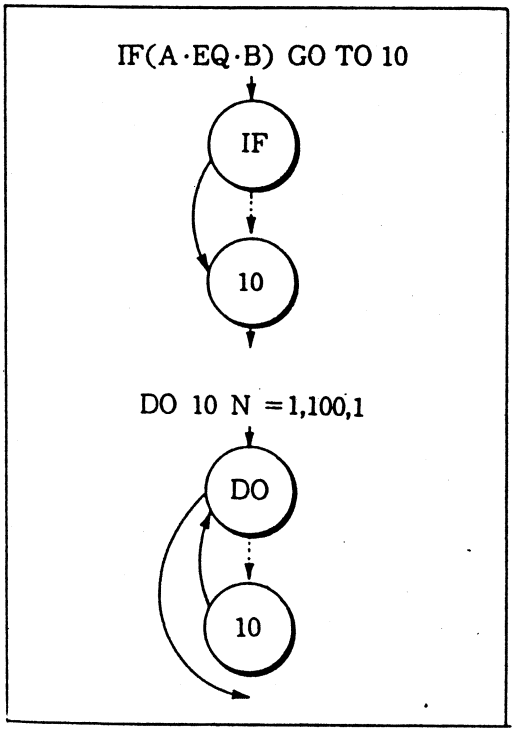
그러나 이 척도는 그래프내에서 분기점(Decision Point)의 배열과 관계없다. [도 11]에서 같은 수의 분기점을 갖는 프로그램들은 분 기점의



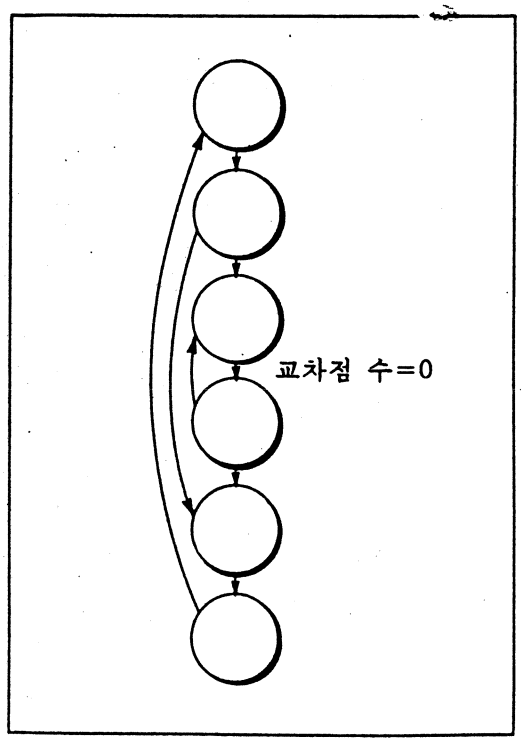
〔도 11〕 C(G)=6인 프로그램의 제어흐름 그래프

배열에 관계없이 항상 똑같은 복잡도를 가지게 된다.

이러한 점은 복잡도에 프로그램 이 구조를 충분히 반영하지 못한다는 것을 의미한다. 또한 프로그램의 구조를 반영하지 못한 경우 프로그램의 이해도와 신뢰도에 직접적인 영향을 미치므로 이해도의 관점에서 문제점 이 대두된다.



〔도 12〕 IF 구조와 DO 구조의 표현



〔도 13〕 구조화 프로그램 그래프

(2) 모아워드의 W 척도

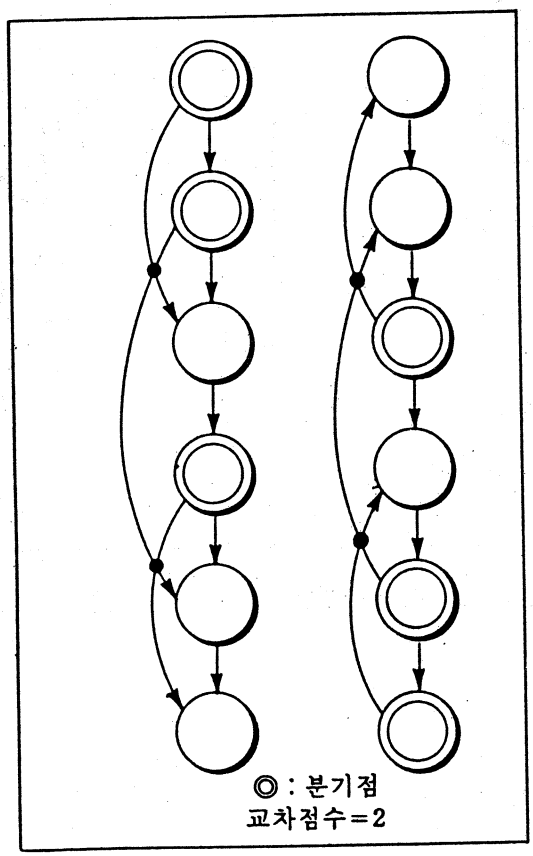
W 척도는 맥케이브의 척도와 거의 비슷한 문제점을 가지고 있다.

(3) 우드워드의 Knot 척도

이 척도는 프로그램내에 있는 제어 흐름들의 위치와 정점들의 배열에 따라 증속되는 척도이다. 이것은 제어흐름이 점점 더 많아질수록 프로그램의 복잡도가 증가되기 때문에 프로그램내의 구조도 정도를 잘 나타내 준다.

그러나 이 척도는 다음과 같은 몇가지 문제점을 안고 있다.

1. IF 문과 같은 선택문에 의해 초래되는 제어흐름 복잡도는 잘 반영



〔도 14〕 제어흐름의 그래프의 예

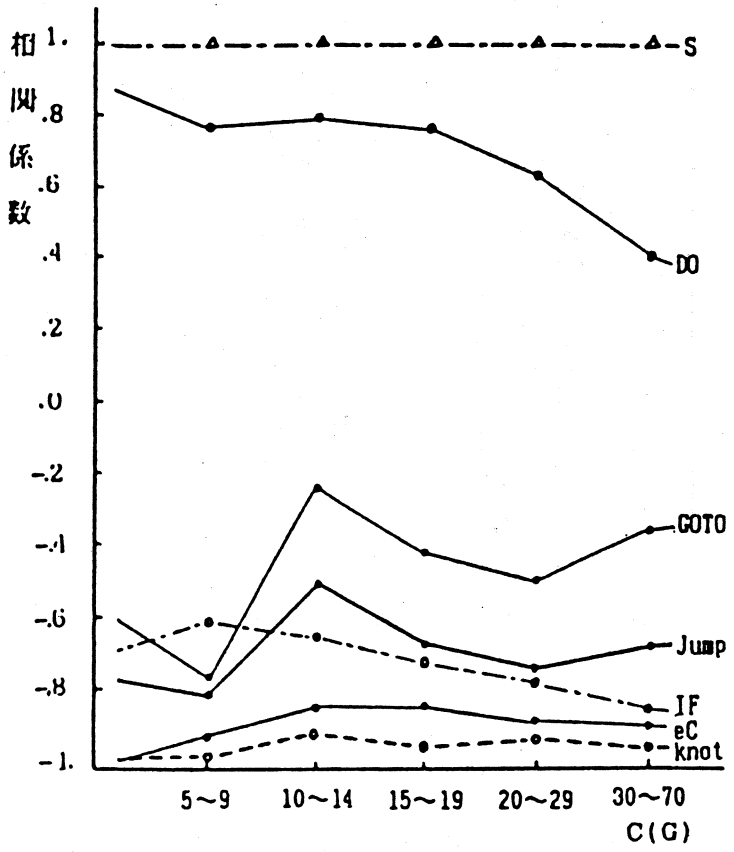
하지만, DO 문과 같은 반복문의 복잡도는 거의 반영하지 못한다 ([도 12] 참조).

2. 구조화된 프로그램에 대해서는 도 13과 같이 항상 동일한 복잡도 0를 가진다.

3. 프로그램의 분기점에 의한 제어흐름의 방향에 관계없이 항상 일정 한 복잡도를 가진다([도 14] 참조).

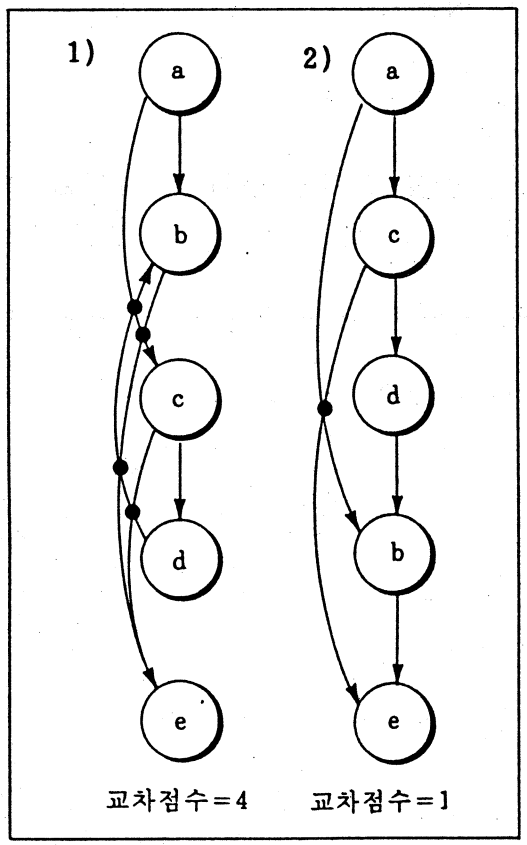
6. 복잡도의 개선 방안

프로그램의 복잡도에 영향을 미치는 순서는 고차점 수, eC(essential



Complexity:맥케이브가 제안한 구조도), IF 문, Jump의 수, DO 문, GO TO 문 순이다.

따라서 [도 15]의 1)을 2)로 변환시켜 교차점수를 줄여 줌으로써 프로그램의 복잡도를 개선시킬 수 있다 (단, 여기서 프로그램 논리를 변경시키지 않고 정점의 위치를 재배열한다).



〔도 15〕 교차점수를 줄이는 방법의 예

참 고 문 헌

1. T.J.McCabe, "A Complexity Measure", IEEE Trans. Software Eng., Vol. SE-2, No.4, pp. 308-320, December. 1976.

2. M.R.Woodward, M.A.Hennell and D.Hedley, "A Measure of Control Flow Complexity in Program Text", IEEE Trans. Software Eng.,Vol. SE-4, No.1, pp. 45-50, January, 1979.

3. M.R.Paige, "A Metric for Software Text Planning", in Proc. COMPSAC 80, pp. 499-504, 1980.

4. V.R.Basili,"Product Metrics", Tutorial on Models and Metrics for Software Management and Engeering,IEEE Computer Society Press, p. 214-217, July, 1980.

5. A.L.Baker and S.N.Zweben, "A Comparison of Measures of Control Flow Complexity", IEEE Trans, Software Eng., Vol. SE-6,No. 6,p. 506-512, November, 1980.

6. W.Harrison and K.Magel, "A Complexity Measure Based on Nesting Level", ACM SIGPLAN Notices,Vol. 25, No.3,p. 63-74, March 1981.

7. W.Harrison, K.Magel, R.Kluczny and A.Delock "Applying Software Complexity Metrics to Program Maintenance",IEEE Computer Society, Vol.15, No.9, pp.65-79, September. 1982.

8. IMSL, "Problem-Solving Software System, "MATH/SFUN Library Ref. Manual, IMSL Inc, 1985.

9. C.Jones, "Programming Productivity", McGraw-Hill Series in Software Eng., and Technology, New York, pp. 70-71, 1986.

10. S.Yau and J.P.Tsai, "A Survey of Software Design Techniques", IEEE Trans. Software Eng., Vol. SE-2, No.6, pp. 718-719, 1986.

11. Kuo-Chung Tai, "A Program Complexity Metric Based on Data Flow Information in Control Graphs", 7th International Conference on

Software Eng.,1984.

12. S.Henry, D.Kafura, "Software Structure Metrics Based on Information Flow", IEEE Trans. SE-7, No.5, pp. 510-519, 1981.
13. M.Halstead, "Element of Software Science", Elsevier Notth-Halland, New-York, 1977.
14. 이 경환, "소프트웨어 공학", 희중당, 1987.
15. 안 금순, "Scope Metrics를 이용한 복잡도 측정에 관한 연구", 중앙대 석사학위논문, 1986.
16. 한 규정, "소프트웨어 품질 보증을 위한 혼합적 복잡도 측정에 관한 연구", 중앙대 석사학위논문, 1987.
17. 한 규정, 이 경환, "혼합적 방법에 의한 소프트웨어 복잡도 측정", 한국정보과학회 논문지, 제16권 제2호, 1989.
18. 양 해술, T.Araki, "Refinement on Complexity and Structuredness of Program's Control Flow", 일본정보처리학회, 소프트웨어 공학 56-5, 1987.
19. 양 해술, Y.Tsujino, N.Tokura, "Software Complexity Measure Based on Data Flow Information",일본 전자정보통신학회, COMP 88-77, 1989.
20. 양 해술, "소프트웨어 복잡도의 측정 방법", 컴퓨터 정보사, 컴퓨터 월드, 1988. 8.
21. 양 해술, Y.Tsujino, N.Tokura, "Software Complexity Measure Based on Data Flow Information", 일본 전자정보통신학회 논문지 D-I, Vol, J72-D-I, No.11, pp. 789-796,1989.
22. 양 해술, "소프트웨어 공학의 현상과 동향",하이테크정보출판부, 1989.

23. 양 해술, "Y.Tsusjino, N.Tokura, the Complexity Measure of Program Based on the Inter-Module Dependency", 일본정보처리학회,소프트웨어 공학 62-5, 1990.

Ⅲ. 소프트웨어 구조도의 측정·평가 방법

1. 프로그램의 구조도의 측정방법

1.1 McCabe의 eC 척도

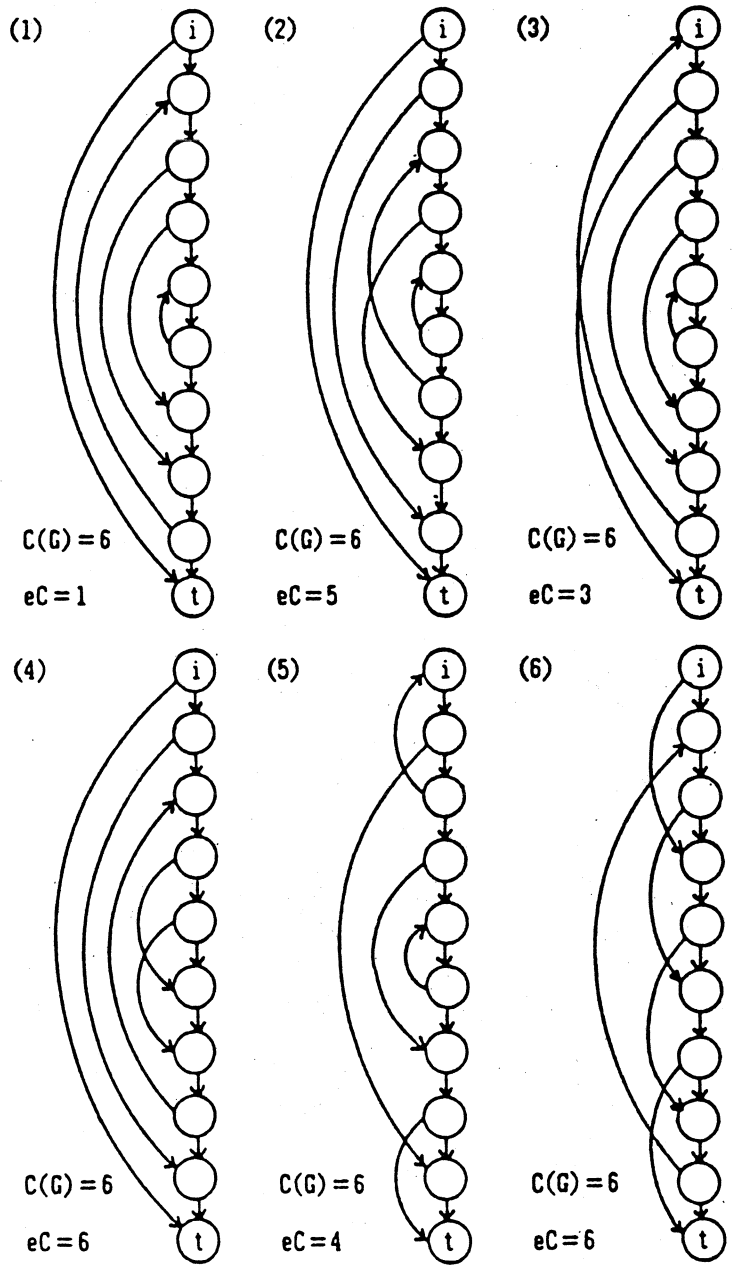
McCabe는 구조화 프로그램을 이용해서 프로그램 내의 구조화 정도를 반영해주는 척도 eC를 제안하였다. 이 척도는 프로그램 구조화 성분(Struc- tured Component;SEQUENCE, IF THEN ELSE, DO UNTIL or DO WHILE) 들을 단일 노드로 줄일수 없을 때까지 감축한 그래프로 부터 구한 Cyclomatic 수를 eC(essential Complexity)라 하고 다음과 같이 정의한 다. 이것에 의하여 프로그램이 얼마나 구조화 되었는가를 판단할 수 있다.

eC=C(G)-m(구조화 부분 그래프의 수)

따라서 (5)의 그래프에서 eC는 4가되며, 완전히 구조화된 프로그램에 대해서는 eC=1이 된다.

(1) eC 척도의 문제점

척도 eC 는 프로그램의 제어흐름 그래프 내에 있는 구조화 성분들을 하나의 정점으로 축소함으로써 본래 같은 수 의 분기점을 가지는 그래프라 하더라도 서로 다른 복잡도를 나타내기 때 문에 분기점의 배열에 따른 문제점이 해결 될 뿐만 아니라 프로그램의 구조화 정도를 파악할 수 있다.



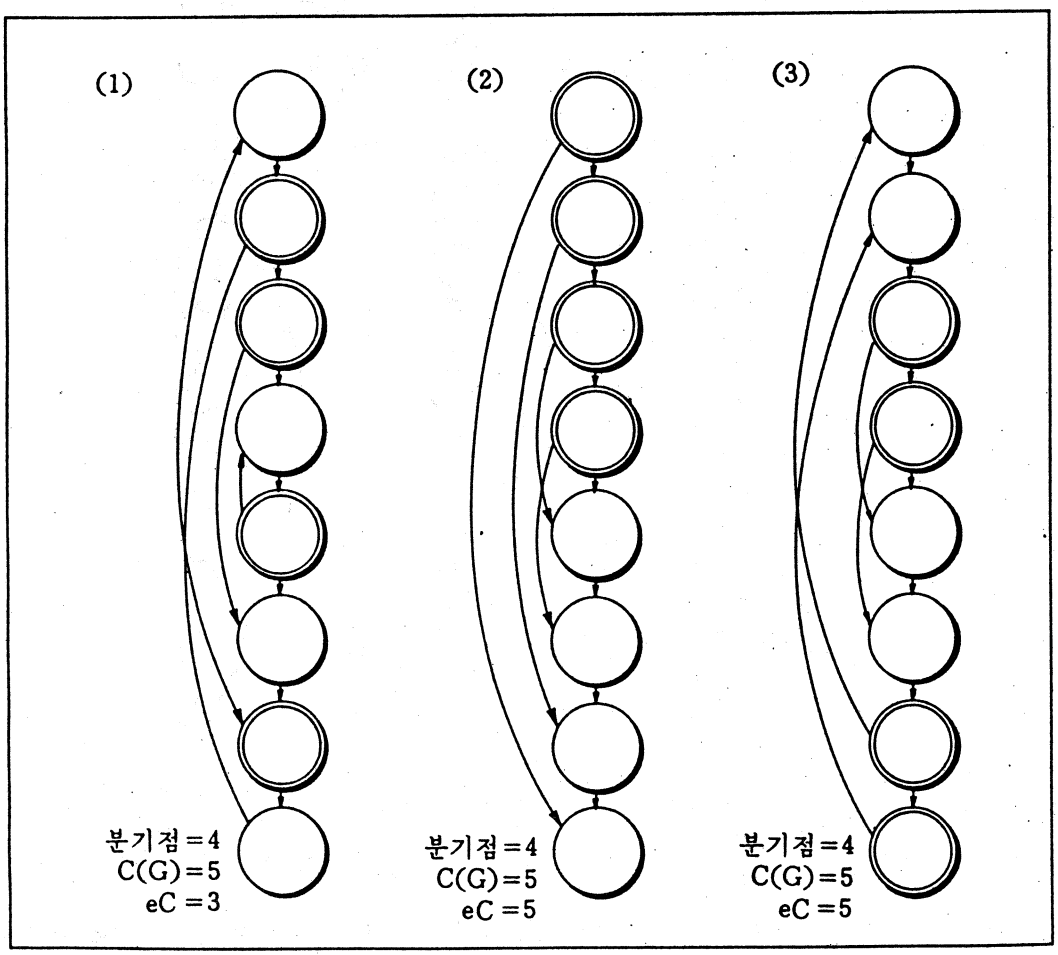
그러나 구조도 척도 eC의 문제점으로서 eC의 측정에 있어 그래프 의 축소는 다중화된 부분 그래프들 중에 가장 안쪽에 있는 부분그래프 (MIS;Most Inner Subgraph)에서 바깥쪽에 있는 부분그래프 (NSC;NonStructured Component)의 위치에 따라 eC의 값이 달라지게 된다.

1) NSC가 MOS의 내부에 있을 경우

[도 1]의 (1)과 같이 MOS의 내부에 잇는 부분 그래프들은 하나의 장점으로 감축됨으로써 프로그램의 구조도를 잘 나타내준다.

2) NSC가 MIS에 있을 경우

[도 1]의 (2)와 같이 NSC로 인하여 그래프가 더이상 감축될 수 없기 때문에 C(G)=eC가 된다. 따라서 이 프로그램은 거의 구조화된 형태임에도



〔도 1〕 C(G)=6인 프로그램의 제어흐름 그래프

불구하고 비구조화된 그래프로 평가하게 된다. 또, [도 1]의 (3)은 거의 비구조화된 그래프 인데도 동일한 eC값을 갖는다. 그러나 이 두 그래프는 프로그램의 이해도와 구조도 면에서 볼 때 현격한 차이가 있으므로 같은 구조율을 가지는 그래프는 평가될 수는 없다.

3) NSC가 MIS와 MOS사이에 있을 경우

중간에 있을 경우에는 NSC의 내부에 있는 부분 그래프는 감축됨으로써 어느정도의 구조도는 나타낼 수 잇지만, 바깥부분은 감축될수 없으므로 2)의 경우에 지적된 문제점을 수반한다.

따라서 이상과 같은 eC의 문제점을 보완할 필요가 있다.

1.2 Regular Expression에 의한 구조도 측정

제어흐름 그래프의 각 노드나 에지의 레이블을 이용하여 그래프에서 가능한 모든 실행흐름을 Regular Expression으로 나타내고 이를 통하여 복잡도 및 구조도를 측정하도록 한다.

(1) 오퍼랜드와 오퍼레이터의 정의

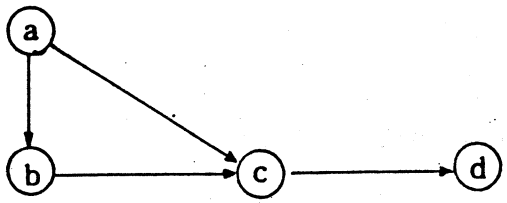
제어흐름 그래프를 Regular Expression으로 표현하기 위하여 다음과 같은 오퍼레이터와 심볼을 정의한다.

Regular Expression은 오퍼랜드와 오퍼레이터들의 조합으로 이루어지는데 오 퍼랜드들은 노드(에지)들의 열로 이루어 지며 오퍼레이터들은 노드들간의 관계를 나타낸다.

<표 1> Operator Symbol의 의미

operator symbol	의 미	예
+	선택노드	IF-THEN-ELSE,CASE
* closure	loop (반복노드)	Do, While-Do
+ transitive closure	loop (한번이상 실행되는 반복노드)	Do-until, Do-while
( )	operand와 operator의 구조적연결	(A+B), (AB)*
ϕ mull	labeled-node graph상에서 선택 노드(+)와 결합되는 operand 부족시 발생	A(B+ϕ) C

이들에 의하여 주어진 프로그램에서의 가능한 모든 실행 순서를 표시해야만 한다. [도 2]와 같은 Labeled Node 그래프의 Regular Expression은, a(b+ϕ)cd 이며, 이것의 의미는 a의 실행 후 bcd를, 또는 a의 실행 후 cd를 실행하라는 것이다.



[도 2] **Labeled node 그래프**

만일 그래프 상에 존재 가능한 모든 경로들 중 Regular Expression에 나타 나지 않은 경로가 있다면, 이것으로는 올바른 복잡도나 구조도를 측정할 수 없다.

(2) 복잡도(Cop)

제어흐름 그래프로부터 Regular Expression을 구했을 때, 그 Expression 안에 존재하는 오퍼레이터(+, *, +(small))의 수로 복잡도를 측정한다. 이 때 복잡도 Cop는

$$Cop = \sum_{op=1}^{n} Exp(op) + 1 = \text{오퍼레이터의 수} + 1$$

로 나타낼 수 있다.

만일 프로그램이 중복 샐행하는 경로가 존재할 때(nested structure, 비구 조화 요소를 갖는 구조)도 이 식으로 측정이 가능하다.

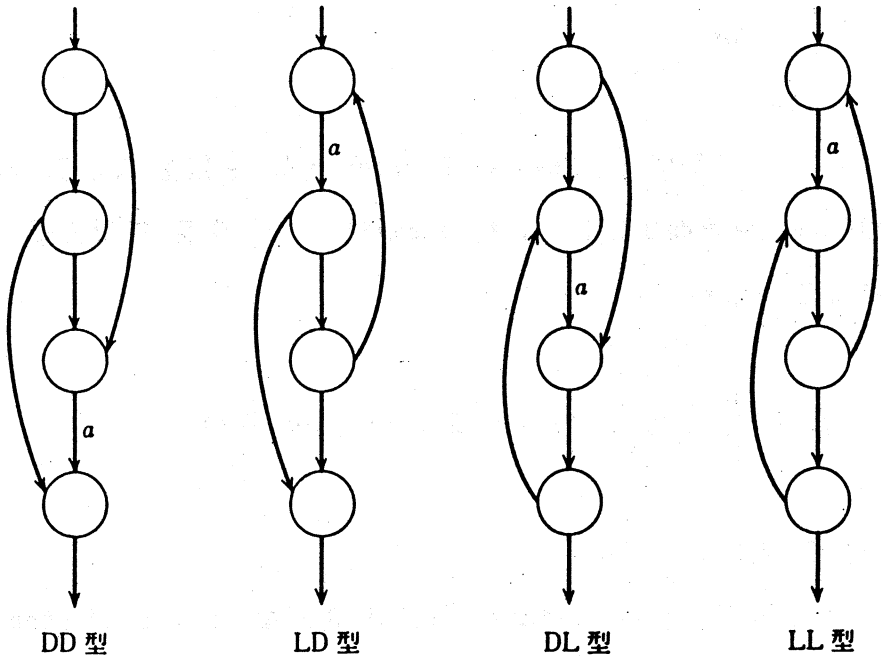
(3) 구조도(SP)

Regular Expression을 이용하여 얻은 Cop로부터 다음과 같은 구조도를 정의한다.

$$SP = \frac{Cop1}{Cop_1 + (mdn - 1) * k}$$

mdn의 Exptession에 가장 많이 사용된 노드(에지)의 수이며, K는 Knot 의 수를 나타낸다.

프로그램 복잡성과 깊은 연관이 있는 구조도는 다음의 성분이 그 정도를 좌우한다. 다음은 비 구조화 요소의 4가지 형태이다.



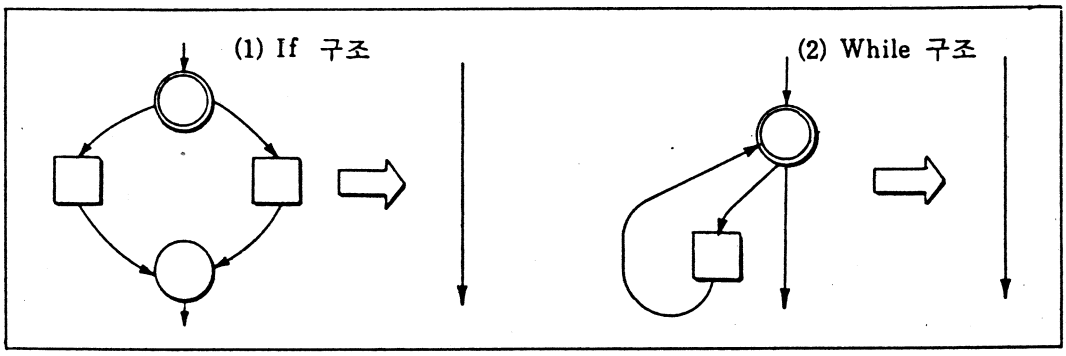
비구조화 성분의 기본형태

2. 프로그램의 비구조도

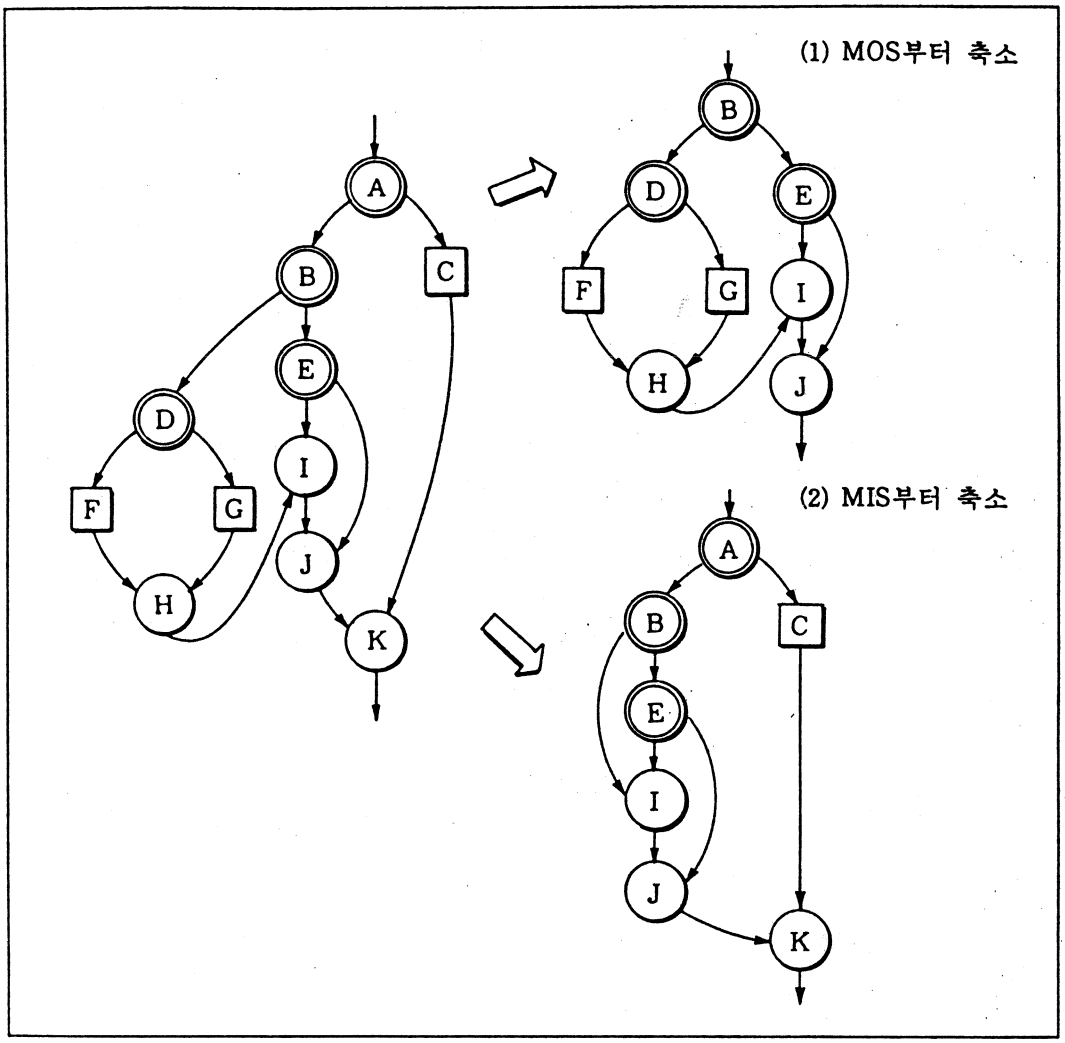
2.1 비구조화 성분의 구조 해석

구조화 프로그램은 Sequence, If-then-else, While 구조로 구성된다. 반면 비구조화 프로그램을 구성하는 기본적인 제어구조는 Williams가 제시 한 정의를 추가하여 사용한다.

그리고 여기에서 생각하는 제어흐름 그래프 G=(V, E)는 프로그램(비구조화 제어구조 및 대입문으로 구성)과 제어흐름으로부터 얻어질 수 있는 유향 그래프이다. 이하 제어분의 분기가 없는 순차적인 명령문의 집합 을



〔도 3〕 구조화 부분그래프의 축소

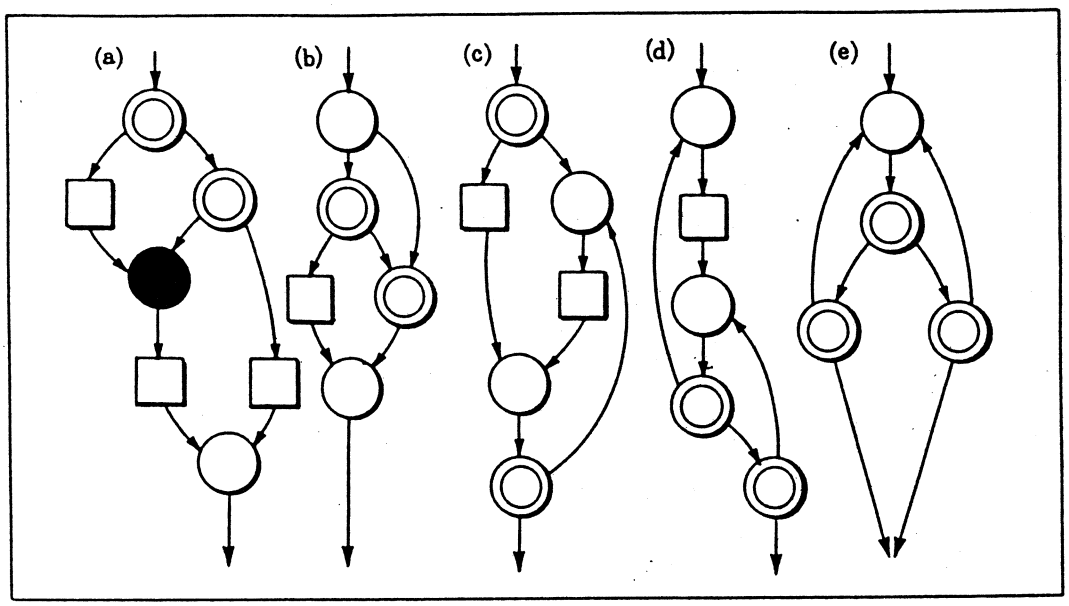


〔도 4〕 구조 부분그래프의 양쪽 방향에서의 축소

Block이라고 한다. 그리그 각 정점 U∈V는 프로그램의 Block에 대응 하고 있다. 또, 각 유향변(U, V)∈E는 정점 U로부터 정점 V에의 제어 흐름을 나타낸다. 즉, 비구조화 프로그램은 프로그램의 Flow-Diagram을 구조화시킨 Sequence, If-then-else, While을 [도 3]과 같이 하나의 Sequence로 축소에서 간략화시키며, 이러한 순서로 하나의 프로그램을 [도 4]와 같이 축소시켜 나간다.

위와 같은 방법으로 축소시킬 경우 [도 5]와 같은 비구조화 성분으로 이루어진 부분그래프가 나타나게 된다. [도 5]와 같은 제어흐름 그래프에서○,◎,□은 정점으로서 ◎는 제어구조, □는 제어의 분기가 없는 대입문, ○는 입력 Edge가 2이상인 정점을 나타낸다.
그리고, 정점을 향한 Edge를 입력 화살표 정점에서 나오는 것을 출력화살표라고 한다. 특히, (a)에서는 선택 Bus와 별도의 선택 Bus가 비정상적인 교차점을 가지므로 ([도 5]의 (a)에서 교차점의 ●부분) 이것을 비정상 교차점이라고 한다. 그리고 ○,◎,□를 순서적으로 선택 Bus와 반복 Bus와 반복 Bus에서 이루는 정적인 논리구조를 가진 유일한 diagram으로 해석한다. [도 5]에 나타난 다섯가지의 비구조화 성분에 의한 제어구조의 특징을 살펴 보면,

(a)는 선택 Bus가 닫히기 전에 별도의 선택 Bus와 교차되고 있으며,
(b)는 반복이 복수개의 출구를 가지고 있으며,
(c)는 반복이 복수개의 입구를 가지고 있으며,
(d)는 반복이 중복(Overlap)되고 있으며,
(e)는 병렬 반복을 가지고 있다.



〔도 5〕 비구조화 성분으로 이루어진 부분그래프

따라서 비구조화 프로그램은 이상의 다섯가지 중에서 하나 이상의 성분을 포함한 것을 의미한다. 정적인 논리구조는 동적 Bus의 최단 논리 Bus로써 부여할 수 있는 논리구조이다.

○은 선택 Bus가 교차하는 반복의 입구 부분으로서 그 방향은 아래와 같이 결정한다. 먼저 ◎에서 나온 Bus는 우측이나 좌측으로 한 방향으로 결정하여 Jump하는 것으로 여기서는 우측이다. 다음에 ○의 판정은 , ◎의 우측 Bus위에서 한번 만나고 되돌아오기 때문에 다시 같은 선택 Bus의 좌측 Bus위에서 만나는 경우는 선택 Bus의 교차점이다. 또, 최초 ○와 만나기 때문에 돌아오기전에 다시 만난다면 반복의 입구부분이다.

위와 같은 순서로 반복 Bus와 선택 Bus의 이상을 나타내는 비정상 교차점을 검출할 수 있다. 여기서 복수개의 반복 Bus가 한개의 교차점에서 교차해서 동일한 반복의 입구로 분기되는 경우에는 이것들을 통합하여 취급한다.

비정상 고차점의 검출은 선택 Bus가 반복 Bus가 반복 Bus의 입구 부분으로 분기 하는 경우에 그 이전의 고차점은 선택 Bus의 고차에서는 없기 때문에 ◎를 □로 바꾼후에 그 Diagram에서 다음의 비정상 판정 알고리즘과 같이 를 순서적으로 하나씩 수행한다.

[비정상 고차점 판정 알고리즘]

/*입력 제어흐름 그래프에 대하여 비정상
고차점을 판정하기 위한 자료 산출 */

입력: 제어 흐름 그래프 G

출력: 계산된 제어 흐름 그래프

방법: 제어 흐름 그래프 G의 구조에 대하여 다음을 순서적으로 수행한다.

1. ○에 나타나는 순으로 번호를 붙인다.
2. 선택 Bus의 우측 Bus에 나타나는 고차점이 1에서 붙인 연속번호에 R을 첨부한 Lable을 부여한다.
3. 선택 Bus의 좌측 Bus에 나타나는 고차점에 1에서 붙인 연속번호에 L을 첨부한 Label을 붙인다.
4. 2개의 Label이 붙은 고차점을 Stack에 기술한다.
5. 고차점이 비정상 고차점인지 또는 정상 고차점인지의 판정은 2개의 Label을 Stack에 기술한 순서를 비교한다.

2개의 Label이 R과 L로 되어있고 그 한 Label의 전방이 일치하면 정상 고차점이 되고 일치하지 않으면 비정상 고차점이 된다.

비정상 고차점으로 판정된 경우에는 다음에 판정할 고차점의 Label에, 비정상 고차점으로 판정된 고차점과 같은 Label이 있으면 그 Label을 비

정상 고차점의 하나 더 앞의 Label과 교환하고난 후에 다음 고차점을 판정한다. 이것은 비정상 Label이 다음 판정을 시작하는데 유입되는 것을 막기 위해서이다.

2.2 비구조화 성분의 Label정의와 비구조도의 측정방법

여기에서는 제시한 비구조화 성분들에 대한 중요도를 고찰하고 비구조 화 성분 Label을 정의한다. 비구조화 성분의 중요도라는 것은 2.1절에서 살펴본 제어구조를 국소적으로 본 경우에 정적인 논리 Bus의 복잡성이 크다는 것을 의미한다. 예를 들면 [도 5]의 (e)는 4개의 반복 Bus의 복잡성이 있고 (d)는 반복중에 별의 반복 입구를 공유하고 있으므로 3개 의 반복 Bus가 얽혀있다. 따라서 (e)는 (d)보다 비구조화 성분의 중요도가 크다고 볼 수 있다. 그리고 (c)는 2개의 반복과 1개의 선택 Bus의 복잡성보다 중요도가 크다는 것을 알수 있다. (b)는 1개의 반복과 1개의 선택 Bus 의 복잡성보다 중요도가 크다는 것을 알수 있다. 또한, (a)는 2개의 선택 Bus의 얽힘이 있다. 반복 Bus와 선택 Bus의 복잡함을 비교하면 위에서 나타난 반복의 진행방향이 중요도를 크게하고 있으므로 그 중요도의 순 서는 e>d>c>b>a가 된다.

비구조화 제어구조에 대해서 비구조도를 나타내기 위한 요인 Vector V 를 위해 비구조화 성분 Label V1에서 V5를 정의한다.

(a) 선택 Bus의 비정상 고차점의 입력 화살표수-1. 즉, 선택 Bus가 비정상적으로 겹치는 수;5

(b) 반복 Bus에서 출력하여 나가는 출력 화살표수-1. 즉, 정규 출구 수 1개를 제거한 부정 출구의 수;V4

(c) 반복 Bus의 밖에서 안으로 분기되어 들어오는 입력 화살표를 가지 는 교차점의 수. 즉, 부정 입구의 수;V3

(d) 반복 Bus내에 포함되는 별도의 반복 입구 부분의 수;V2

(e) 입력 화살표에 3개 이상 가지는 반복 입구 부분의 수;V1

단, 바깥쪽의 반복에 완전히 포함 되는 반복이 있을 경우에는, 안쪽 반복에서도 Count하기 때문에 바깥쪽에서 이중으로 Count하지 않도록 한다.

위와 같은 요인에 대해서 비구조화를 유발하는 성분 Vector $\vec{V}$를 중요도가 큰 순서로 나열함으로써 5차원 Vector로 척도 $\vec{V}$를 정의한다.

$$\vec{V}=(V1, V2, V3, V4, V5)$$

다차원 Vector $\vec{V}$에서 좌측의 요소가 크면 클수록 비구조화를 유발하 는 중요한 성분이 많이 포함되어 있다는 것을 의미한다. 실제로 비구조화 프로그램의 경우 (a)나 (b)가 적당하더라도 하나 이상 포함하고 있기 때문에 Vi는 동일한 종류의 것을 합한 총계로 한다. [도 5]의 (a)-(b)에 대한 각각의 요인 Vector는 다음과 같다.

(a)=(0, 0, 0, 0, 1)　　(b)=(0, 0, 0, 1, 0)

(c)=(0, 0, 1, 0, 0)　　(d)=(0, 1, 0, 0, 0)

(e)=(1, 0, 0, 0, 0)

3. 끝 맺 음

현재는 제어의 흐름에 기반을 둔 구조도가 가장 많이 이용되고 있다. 그

이유는 프로그램 사이즈나 데이타 흐름에 기반을 둔 구조도 보다는 제어의 흐름에 의해 측정하는 구조도가 더 신뢰성이 높기 때문이다.

따라서 앞으로의 연구 진행 방향은 데이타 흐름정보에 의한 구조도의 정확도 개선 문제와 다양한 대상언어의 적용문제이다.

또, 세가지 요소(프로그램 사이즈, 제어의 흐름, 데이타의 흐름)를 전 부 고려한 복합적인 척도의 개발이 필요하다고 본다.

참 고 문 헌

1. T.J.McCabe, "Complexity Measure', IEEE Trans. Software ENG., Vol. SE-2, No. 4, pp. 308-320, December. 1976.
2. M.R.Woodward, M.A.Hennell and D.Hedley, "A Measure of Control Flow Complexity in Program Text", IEEE Trans. Software Eng., Vol. SE-4, No. 1, pp. 45-50, January. 1979.
3. M.R.Paige, "A Metric for Software Text Planning", in Proc. COMPSAC 80, pp. 499-504, 1980.
4. V.R.Basili, "Produst Metrics", Tutorial on Models and Metrics for Software Management and Engeering, IEEE Computer Society Press,p. 214-217, July, 1980.
5. A.L.Baker and S.N.Zweben, "A Comparison of Measures of Con- trol Flow Complexity", IEEEtrans,Software Eng., Vol. SE-6, No. 6, pp. 506-512, November, 1980.

6. W.Harrison and K.Magel,"A Complexity Measure Based on Nesting Level", ACM SIGPLAN Notices,Vol. 25, No.3, pp. 63-74, March. 1981.
7. W.Harrison, K.Magel, R.Klucznу and A.Delock "Applying Software Complexity Metrics to Program Maintenance",IEEE Computer Society, Vol. 15, No. 9, pp. 65-79, September. 1982.
8. C.Jones, "Programming Productivity", McGraw-Hill Series in Software Eng. and Technology, New York, pp. 70-71, 1986.
9. 양 해술, 노 희영, N. Tokura, "Backer 알고리즘에 기반을 둔 비구조적 성향에 대한 구조화 연구', 한국정보과학회 논문지, 제15권 제5호, 1988.
10. 양 해술, Y. Tsusjino, N. Tokura, "비구조화 성분율 고려한 프로그램의 비구조도",한국정보과학회 '89 봄학술발표논문집, 1989. 4.
11. 양 해술, "소프트웨어 공학의 현상과 전망",하이테크 정보출판부, 1989.

Ⅳ. 품질보증기술의 현상과 과제

소프트웨어 공학의 입장에서 테스트기술 분야는 프로그래밍 기술 과 함께 비교적 방법론이 확립되어 있는 분야로 상당한 수준에까지 실용화가 진행되고 있다.

또, 소프트웨어 품질보증 분야 역시 소프트웨어 공학의 다른 분야보다 도일찍 품질관리 방법을 도입, 이미 상당한 수준에 이르고 있다.

그러나 소프트웨어 품질관리 기능조사에 의하면 품질관리 실천상 기본 데이타의 하나인 " 시스템 가동후의 오류상황의 파악율은 50%" 를 넘지 못하고 있다.

또한, 품질관리의 도구(Tool)에 있어서도 " 테스트 회복 도구의 이용률은 80% " 에 지나지 않는다.

이러한 사실들은 테스트 및 품질보증 기술이 실무에서 제대로 적용되 지 못하고 있음을 보여준다.

따라서 여기에서는 개발현장의 각 과정에서 품질평가를 결정하는 기술들을 사례 중심으로 살펴본다.

1. 테스트의 기술

1.2 계층적 테스트 고찰방법

테스트 작업은 신뢰성과 생산성향상의 양면에서 개선할 필요가 있다. 전자는 상위공장에서 유입된 불량부분을 가능한한 초기 테스트 단계에서 정확하게 검출하는 기술이 요구되고, 후자는 소프트웨어 개발 공정의 40

-50%를 점유하는 테스트 작업의 효율을 컴퓨터의 지원아래 향상시키는 기술이 필요하다.

또한 더욱 중요한 테스트 전략으로 다음과 같은 사항을 들 수있다.

1) 각 테스트 공정에서 오류검출의 목표치를 설정한다.

2) 태스크 테스트(코딩,검토 등)의 중요성을 인식, 오류 검출의 목표 치를 최소한 50% 이상 (가능하다면 70%-80%)으로 설정하고, 이 를 테스크 테스트(Task Test)에서 검출해야 한다. 이것이 달성되기 전에는 실제 컴퓨터를 이용한 테스트를 해서는 안된다.

3) 계층적인 테스트가 실시되어야 하는데 이를 위한 일반적인 테스트 구비조건은 다음과 같다.

o 외부사양 테스트와 내부사양 테스트의 병행 ... [도 1]에 표시된 바와 같이 내.외부 사양 테스트는 각각 일장 일단이 있어 어느 한쪽 만으로는 충분치 않으므로 양자를 복합적으로 병행해야 한다.

o 테스트 기준을 명확하게 설정하고 타당한 테스트 항목의 설정기법 을 채용 ... 특히 외부사양 테스트의 경우에는 사양의 엄밀한 해석 과 형식화 방법이 주요 핵심이 된다.

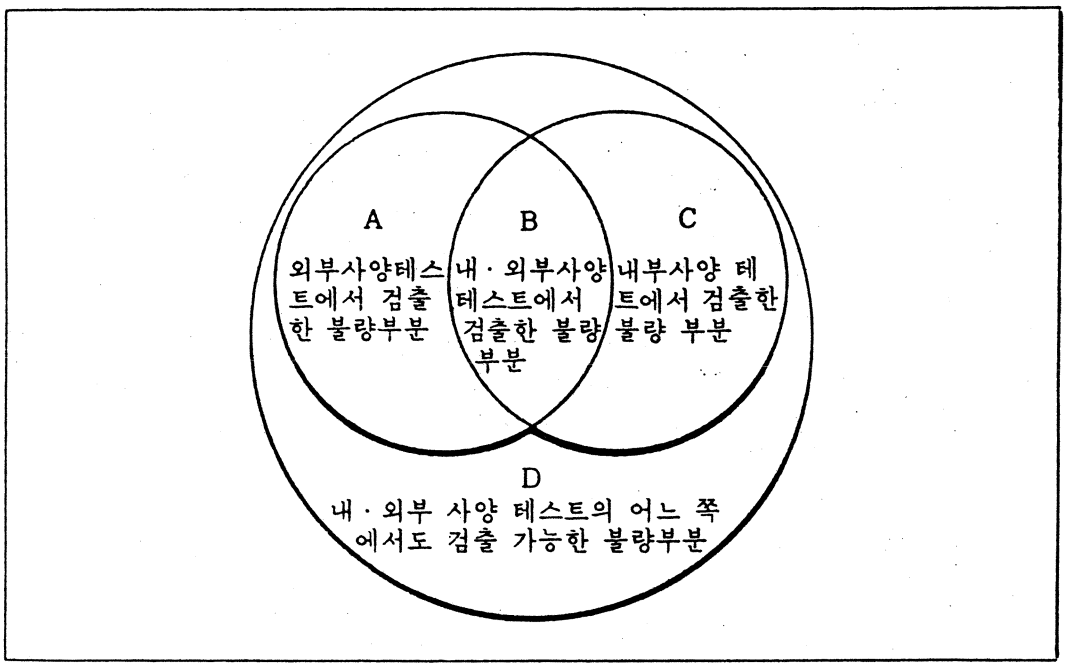
또 내.외부사양에서 나타나는 많은 테스트 항목의 조합사이에서 실용 품질보증과 동시에 채용 가능한 테스트 항목을 추출하기 위한 기준 설정 이 필요하다.

o 테스트 평가척도 및 기술의 채용

o 테스트 작업의 간략화의 실현

4) 시스템 테스트의 실시

[도 1]과 같이 내.외부사양 어느 것으로도 검출이 불가능한 경우도 있다. 이것은 사용자 요구가 외부 사양에서 정확하게 실현되지 않는 경우이기 때문에 사용자 환경과 운용 방법이 유사한 상태에서 시스템 테스트가 필요하다.



〔도 1〕 외부사양 테스트와 내부사양 테스트의 관계

1.2 테스트 항목의 작성기법

1) 테스트 항목의 작성기준

테스트 항목의 작성기준은 가능한한 객관성이 있어야 한다. 즉, 개인간의 차이를 축소시키는 기준이 필요하게 된다.

내.외부 사양의 대표적인 기준과 기법은 <표 1>과 같다. 내부 사양에서는 제어구조에 기초를 둔 기준들이 적용되고 있으나 다른 기준과 조합 해 병행하는 것이 바람직하다.

<표 1> 테스트 항목의 작성기준

구 분	구조모델	데이타 항목 작성 기준 예	
외부사항	공간모델	사양기법 예	테스트 항목작성 기준의 예
		원인결과	. 원인결과 그래프 기법 . 실험계획법에 의한 기법
	시간모델	.결정표 .상태 전이도	. AGENT 기법
내부사항	제 어 구조모델	테스트 평가 척도의 적용 .C0 : 전체의 실행문을 한번이상 실행 .C1 : 전체의 분기를 한번이상 실행 .M0 : 전체의 모듈을 실행 .M1 : 모듈의 관계를 실행	
	데이 타 구조모델	.D0 : 전체의 데이타 요소를 설정.참조 .D1 : 데이타 계통관계의 조합과 전체의 데이타 요소를 설정.참조	

또한, 외부사양의 경우에는 현재로선 문장에 의한 기술이 보통이고 명확한 기준설정이 곤란하지만 어떤 과정의 사양을 모델화해서 그림이나 표 또는 그래프 형식으로 기술할 수 있다.

그리고 3)절에서 소개하는 테스트 항목 작성기법을 사용, 객관적으로 정도가 높은 테스트 항목을 작성할 수 있다.

2) 실천적 테스트 항목 작성기준

과거의 실적 테스트를 정리. 분석하고, 양적. 질적인 기준을 사전에 설정하는 것이 중요하다. 참고적으로 <표 2>, <표 3>에 기준 설정의 예 를 표시 하였다.

<표 2> 검사 항목의 양적 기준 예

순 번	샘플프로그램	프 로 그 램	기 준
1	A	. 정렬 프로그램	300이하
2		. 처리 프로그램	400이하
3		. Batch 프로그램	600이하
4		. 이행 프로그램	600이하
5	B	. 온라인 컨트롤 프로그램	200이하
6		. 처리 프로그램	400이하
7		. Batch 프로그램	500이하

단위 :스텝/건

<표 3> 검사 항목의 질적 기준 예

순 번	분 류	컨트롤프로그램	처리프로그램	Batch프로그램
1	.기본.정상항목	50	60	80
2	.이상.장애항목	20	10	10
3	.한계.경계항목	10	10	5
4	.주변.조건/인터페이스	20	20	5

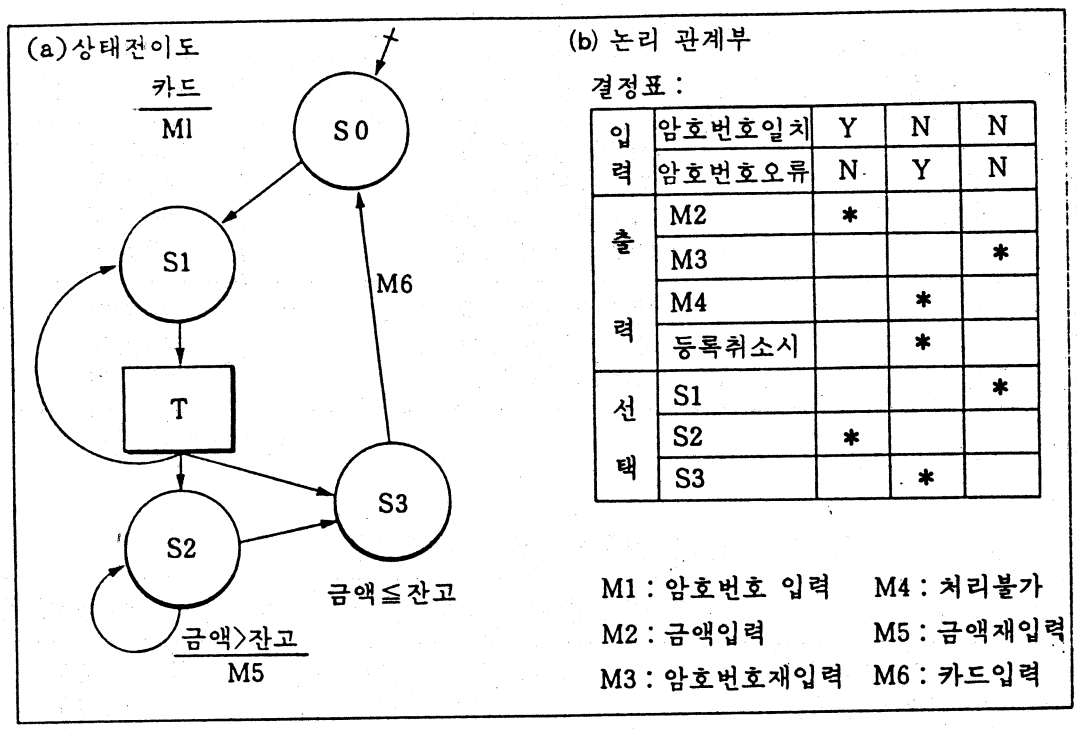
3) 외부사양 테스트의 항목 작성 지원방법(AGENT)

여기에서는 <표 1>의 외부사양 테스트를 계통적으로 수행하기 위한 테스트 항목 작성기법과 그 지원도구를 서술한다.

우선 외부사양에 기초를 엄밀하게 해석하고 이의 표현을 도식화한 기능도식을 작성한다([도 2] 참조).

다음에 도식정보를 테스트 항목 작성 지원도구인 AGENT(Automated GENeration system of Test cases)에 입력, 최적의 테스트 항목을 작성한다.

또 기능도식의 작성과정과 AGENT 내에서의 자동 체크에 따라 외부사 양의 애매모호한 점과 불완전한 점이 검출된다.



입력	암호번호일치	Y	N	N
	암호번호오류	N	Y	N
출력	M2	*		
	M3			*
	M4		*	
	등록취소시		*	
선택	S1			*
	S2	*		
	S3		*	

〔도 2〕 기능 도식의 예 (현금 자동지급기의 기능사양의 예)

이 기법의 장점은 종래 개인의 능력과 경험에 완전히 의존하던 테스트 의 항목작성이 상기 작업방법에 의해 경험이 적음에도 불구하고 적절한 테스트 항목을 작성할 수 있다는 것이다.

1.3 충분한 테스트 평가

테스트의 충분성을 측정하는 척도로서는 제어구조에 착안한 C0, C1 척도가 유명하고 도구도 보급되어 있다.

따라서 <표 4>에 표시된 구비 조건을 파악, 도구를 선택하고 작성하는 것이 바람직하다.

<표 4> 테스트 충분성 척도의 구비조건

항 목	구 비 조 건
기 능	1. C0, C1의 레벨이 취득가능 2. 정보의 축적 기능이 있음 3. 프로그램의 수정중에도 가능
성 능	1. 처리시간 및 메모리 증가 억제
조작성.운용	1. 조작 및 운용이 복잡해서는 안됨

1.4 테스트 지원시스템의 개요

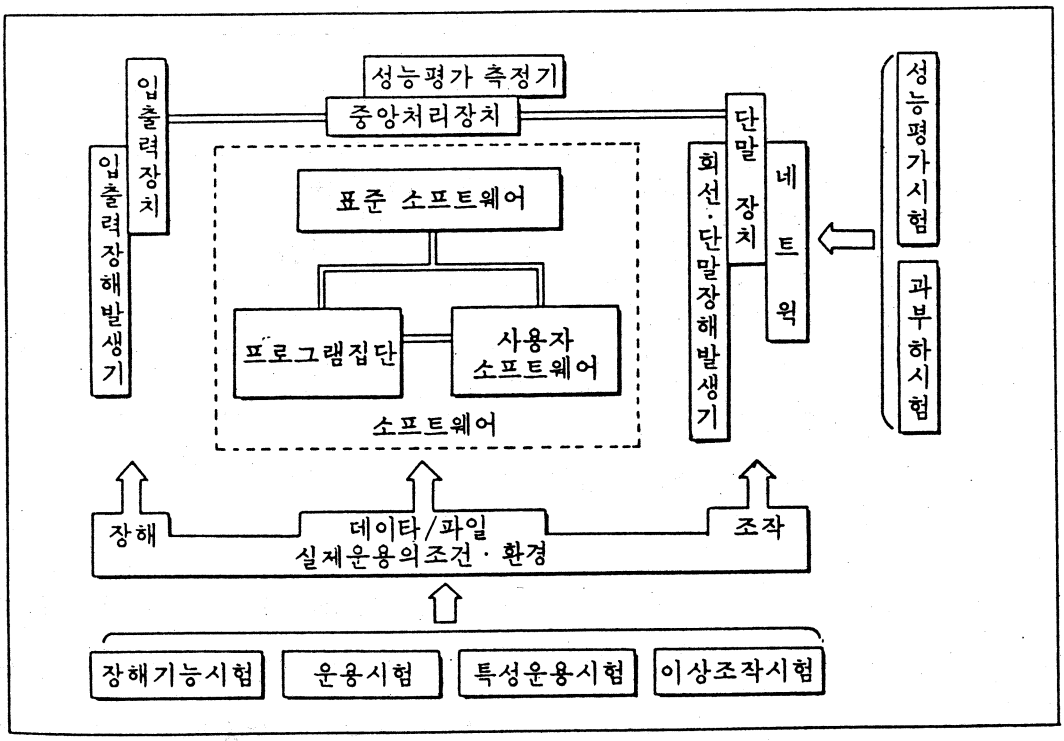
테스트 지원시스템은 테스트 항목 작성지원, 테스트 실행지원, 테스트 평가지원등의 각각의 서브시스템으로 구성된다. 실행지원은 테스트 데이타 생성기(Test Data Generator), 회화형 테스트 실행지원, 테스트 결과 판정지원 등의 도구로 구성되고 테스트 효율향상의 관점에서 매우 중요 한 분야이다.

특히 테스트 결과의 자동조합은 프로그램 수정시 레규레이터(Regulator) 불량을 방지하기 의한 반복 테스트에 필요하다.

1.5 시스템 테스트

시스템 테스트는 [도 1]의 D에서 나타난 영역, 즉 사용자 요구가 외부 사양에 올바르게 복구되어 있는가를 시스템적으로 점검하는 일이 주 목적이다.

이를 위해 실제로 사용자 환경에 가까운 상태를 창출할 필요가 있다. [도 3]은 시스템 테스트의 개념을 나타낸 것으로, 각종 시스템 구성을



〔도 3〕 시스템 테스트의 개념

만들기 위한 하드웨어 설비, 소프트웨어 등과 지원 도구에 의한 애플리케이션 환경이 중요한 요인이 된다.

실제 데이타 파일류의 준비와 운용에 있어서는 시스템 엔지니어, 때로 는 사용자의 협력을 얻어 테스트를 실행해야 한다.

이를 위해서는 많은 터미널과 네트웍 또는 하드웨어 장애를 모형화한 시뮬레이터(Simulator)의 개발.보존이 중요한 책임이다.

1.6 테스트 기술의 통합(완성)과 이후의 과제

수백 K스텝에 이르는 온라인 프로그램 개발에서 상기 계통적 테스트를 적용할 경우, 테스트 완료후 발견된 단위규모당 불량건수는 종래의 방식 과

비교해 30-40% 감소된다는 사례보고가 있다.

따라서 시스템 기술에서는 현재 적용가능한 기법과 도구를 정확히 구사하면 상당히 높은 수준의 신뢰성을 얻을 수 있다.

한편, 앞으로의 과제로는 데이타 구성에 기초한 실용적인 테스트 도구 의 개발, 요구 정의 기술과 결합된 외부사양 테스트의 항목 작성기술, 더 나아가서 지식 데이타 베이스를 이용한 테스트 노하우(Knowhow)의 활 용 등이 대두될 수 있다.

2. 품질 보증 기술

품질보증기술은 소프트웨어 라이프 사이클 전반에 걸친 기술로 여기에서는 품질평가 기술과 이를 기반으로 한 테스트 공정의 품질평가 사례를 중심으로 기술한다.

품질보증 기술로서는 최근 주목받고 있는 계량화 기술의 동향과 평가 기술들을 중심으로 실질적인 품질관리 사례를 소개한다.

2.1 계량화 기술

1) 개요

소프트웨어의 개발 비용, 납기 등 계량화가 지연되어 온 소프트웨어 품질에 대해서도 최근들어 계량화 움직임이 활발해지고 있다.

▲ 첫째는, 소프트웨어 매트릭스(Metrics) 라고 불리는 것으로 생산성 과 품질 등 소프트웨어 전반에 걸친 계량화의 시도이다.

▲ 둘째는,소프트웨어에서도 품질 전개를 적용하고 있는 사례가 많아

지고 있다. 이것은 사용자의 원래 요구를 파악하여 제품의 기획품질을 설정, 이것을 각 기능부품 및 모듈(Module)의 품질(부품의 품질)과 공정 의 요소에 도달할 때까지 계층적으로 품질을 전개해가는 수법이다.

위와 같은 두가지 계량화 움직임은 배경이 서로 다르지만 요구 품질을 구체적으로 측정가능한 레벨로 전개해 평가한다는 점이 서로 비슷하다.

또한 품질을 받아들이는 방향에는 사용자의 관점과 개발자의 관점이 다르므로 우선 전체적인 레벨에서 품질의 특성을 정의할 필요가 있다.

<표 5> 사용자 관점에서 본 품질특성의 비교

Boehm	McCall	SQMAT	SQUALAS(IBM)
	Correctness	.정 확 성	.가 능 성
Reliability	Reliability	.신 뢰 성	.신 뢰 성
Human -Engineering	Usability	.사용 용이성	.호 환 성 .조작성 .습득용이성
Efficiency	Efficiency	.효 율	.성 능
Maintainability .modifiability .understandability .testability	Maintainability Testability	.보 수 성	.보 수 성 .확 장 성
Portability	Reuseability Portability Flexibility	.융 통 성	.가 변 성 .도입 용이성
	Interoperability	.접 속 성	
		.비 밀 성	.비 밀 성

현재 공표되어 있는 매트릭스 중에 품질특성에 착안해 제안된 것이 <표 5>이다.

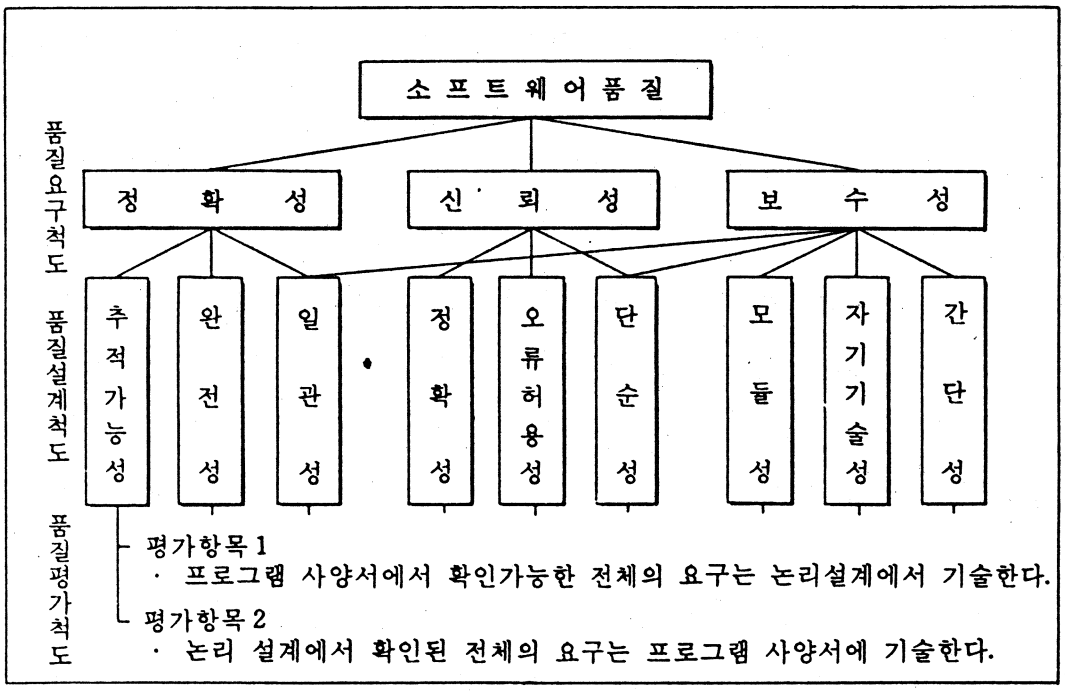
한편 품질을 비교.평가하기 위해서는 용어의 정의와 품질척도의 표준화가 모색되어야 한다. 현재 ISO(국제표준화협회) 및 일본 품질관리협회에서 소프트웨어 품질특성의 표준화 작업이 시도되고 있다.

2) 품질 계량화 사례(SQMAT)

품직척도는 일반적으로 다단계 구조로 이루어져 있다. 개발자쪽에서 많이 사용되는 품질척도 SQMAT(Software Quality Measurement and Assurance Technology)를 소개한다.

이것은 [도 4]에 나타난 바와 같이 3단계의 척도로 구분되어 있고 각각은 나름대로 명확한 의미를 갖고 있다.

즉, 제 1단계의 품질요구 척도는 사용자 관점에서 선정하는 척도이다. 제 2단계의 품질 설계척도는 소프트웨어 개발자의 관점에서 선정되는 척도로 개발단계에서 평가된다.



[도 4] 품질 척도의 구조

또, 제 3단계의 품질평가 척도는 품질설계가 제대로 실현되었는지를 구체적으로 측정하기 위해 각 공정별로 설정하는 척도이다.

2.2 신뢰성 모델과 평가 사례

1) 신뢰성 모델

지금까지 많은 신뢰성 모델이 발표되어 있지만 주로 A.S. Goel이 제안한 다음과 같은 4가지의 패턴으로 분류된다.

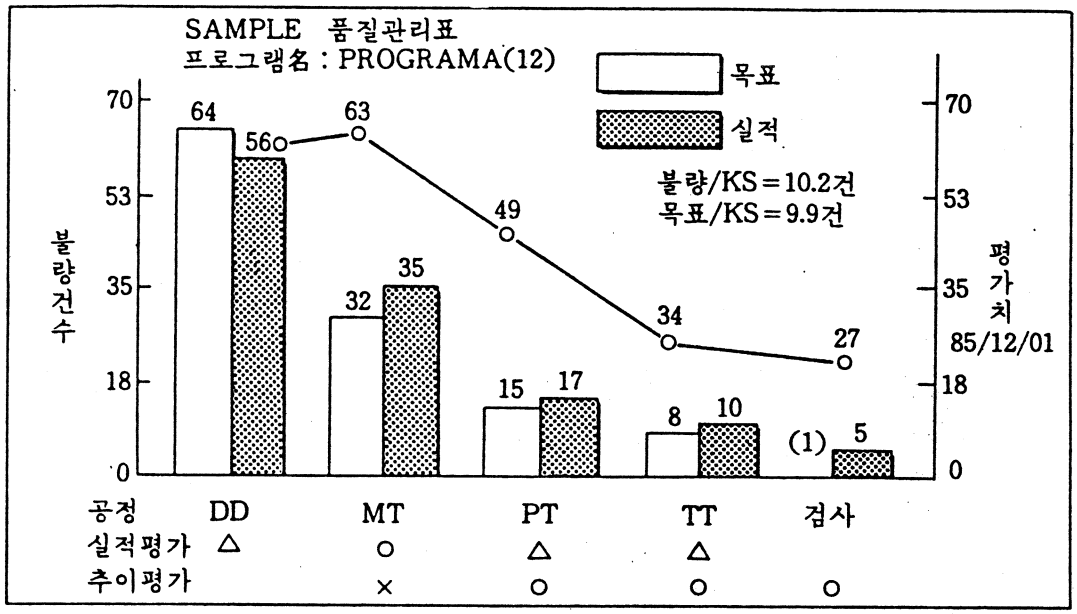
- 고장 간격 모델(Time Between Failures Models)
- 고장률 모델(Failure Count Models)
- Fault Seeding Models
- 입력 도메인 모델 (Input Domain Based Models)

2) 신뢰성 평가 사례(SQE)

여기에서는 소프트웨어 품질 중에서 설계에서 검사까지의 개발공정을 중심으로 가동 상황의 재입력(Feed Back)을 포함하는 소프트웨어 품질 평가 시스템(Software Quality Estimation System)을 소개한다. SQE에서 품질의 추론, 목표치의 관리방식은 신뢰성 설계단계에서 설정 한 검출 불량 건수의 목표치와 실제치를 비교하고, 예측치를 평가하는 동 시에 각 테스트의 공정에서 검출 불량 실적으로부터 품질의 추리를 평가 하는 방식이다.

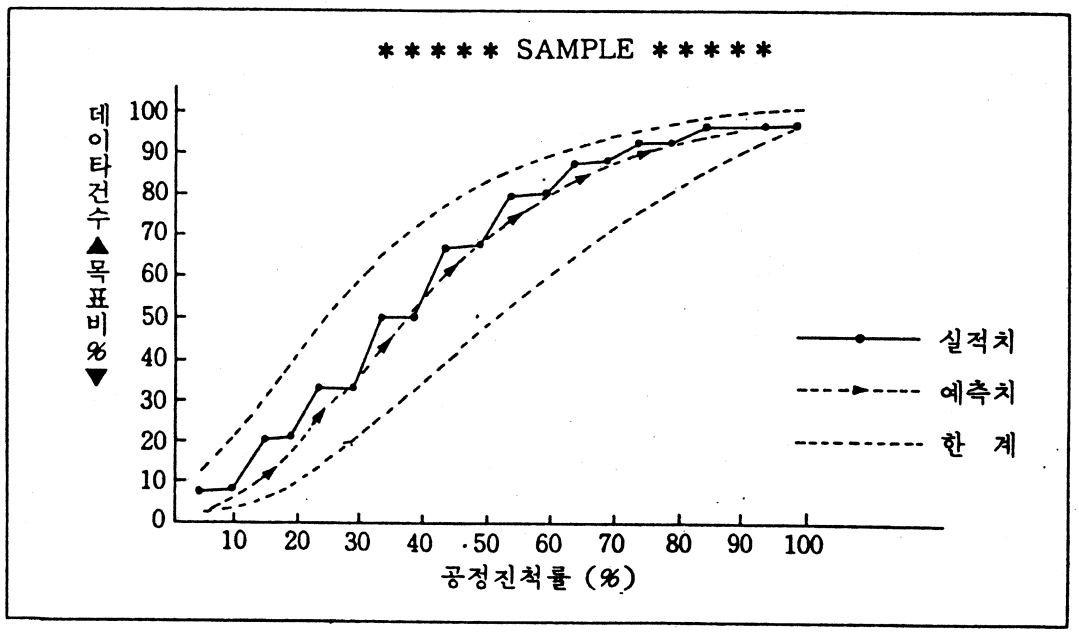
품질의 추리는 평가대상 공정까지의 각 테스트 공정의 검출 불량 건수에 계수를 부과하여 평가치를 나타내고 전단계의 평가치와 비교,평가하는 방법이다.

즉, 전단계의 공정에서 불량 건수를 많이 검출하고 후단계의 공정에서 검출건수가 떨어지면 평가치는 낮아지게 되며 값이 떨어진 만큼 좋은 평가 가



(주) ( )안은 검사 발견 불량건수

〔도 5〕 QC 그래프의 예



〔도 6〕 FRCST의 예

되는 것이다. 이러한 내용을 [도 5]와 같이 QC 그래프로 표시한다.

누적불량건수 예측의 추정은 [도 6]과 같이 출력되고 그 예를 <표 6> 에 표시한다.

<표 6> FRCST 검색 보고서의 출력 예

순 번	프로그램	규 모 (KS)	전 체 항목수	실 시 항목수	검 색 실시율	불량건수	불량률	추 정 불량건수
1	SAMPLE	12	428	39	9%	4	3~24	13~103

2.3 품질 평가기술의 과제

계량화 모델에 대해서는 적용 사례를 많이 수집, 결과의 검증을 통해 효율이 좋은 척도의 선정과 평가지원도구의 개발을 추진하는 것이 바람직하다.

또한, 사용자 관점에서의 품질척도의 표준화 추진이 필요하다고 본다.

3. 끝맺음

지금까지 우리는 테스트 및 품질평가를 주제로 한 품질보증 기술에 대 해 첨단기술의 하나인 실용 레벨의 기술을 중심으로 살펴보았다.

지금까지의 테스트 및 품질보증 기술을 잘 이용한다면 상당한 수준에 도달할 수 있다고 본다.

한편, 앞으로 이 분야의 발전은 요구정의 기술의 명확한 사양에 기반을 둔 테스트 항목 작성 기술과 상위 설계단계의 기술발전에 따라 비약적으 로 발전하리라고 본다.

참 고 문 헌

1. 奈良隆正他, " 소프트웨어의 品質保證活動", ENGINEERS, pp. 6~11, 1983. 10.
2. 野木兼六他, " 소프트웨어 테스트 項目作成支援 시스템 ", 日立評論, Vol. 66, No. 3, pp. 29~32, 1984.
3. 石井康雄, " 소프트웨어의 檢査と品質保證", 日科技連出版社, 1986.
4. 東基衛他, " 소프트웨어의 品質計測/保證技術"(SQMAT), 品質, Vol. 16, No. 1, pp. 79~84, 1986.
5. Goel, A.I, "Software Reliability Models : Assumptions, Limitations and Applicability," IEEE Trans. on Sofware Engineering, Vol. SE-11, pp. 1411~1423, 1985.
6. 古賀惠子他, " 소프트웨어 品質評價 시스템의 開發", ENGINEERS, pp. 18~21, 1986年 8月.
7. Curtis, Bill, "Measurement and Experimentation in Software Development", IEEE, 1981.
8. 石井康雄, " 소프트웨어의 製造, 日科技連出版社
9. 梁 海述, "컴퓨터 프로그램의 품질보증 기준과 실제", 한국정보산업협회, 「정보 산업」, 1988. 11.
10. 梁 海述, "테스트 및 품질보증 기술의 현상과 과제", 컴퓨터 정보사, 「컴퓨터 월드」, 1988. 6.

V. 소프트웨어 품질평가 TOOL(ESQUT) 소개

소프트웨어는 요구 정의로부터 개발종료에 이르기까지 제반 사항을 구현하는 것도 중요하지만 개발 완료 이후의 유지보수도 빼놓을 수 없는 핵심 요소이다.

일반적으로 개발초기에서 보수까지를 포함하는 소프트웨어 라이프 사이클(Life Cycle)에 소요되는 비용은 전체 개발 비용의 70%에 이르고 있다.

이에 따라 보수비용이 적게드는 고품질의 소프트웨어 개발방안으로 개발과정에서 원시 프로그램(Source Program)의 품질을 수치적으로 측정하는 것이 소프트웨어 개발의 필수 조건으로 제시되었다.

이러한 관점에서 원시 프로그램의 품질을 측정하는 방법으로는 다음과 같이 4가지 방법이 가능하다.

1) 구체적인 보수성 품질의 도식화가 가능하다.
2) 기계적이며 자동적인 계측이 가능하다.
3) 객관적인 평가가 가능하다.
4) 보수작업으로 원시 프로그램의 수정, 갱신 부분을 파악할 수 있다.

따라서 여기에서는 이상과 같은 4가지 목표를 기본으로 하여 소프트웨어 라이프 사이클에 소요되는 비용을 줄이기 위해 C 언어로 작성된 원시 프로그램의 개선과 수정을 평가할 수 있는 중요한 요인 항목의 선정과 각 항목에서의 측정 방법을 일본 TOSHIBA에서 개발한 ESQUT 시스템에 대하여 소개하고자 한다.

1. 소프트웨어 품질평가의 요인항목

보수 비용이 낮은 소프트웨어의 모델을 4가지로 분류하여 각 모델에서 품질평가의 요인항목을 설정하고 측정방법을 소개한다.

1.1 모듈기능 사이트 모델(Site Model)의 요인항목

이 모델에서 모듈(Module) 하나가 가지는 기능의 수를 측정하여 정량 화한다. 원시 프로그램의 각 모듈이 가지는 기능은 적을수록 좋고, 한 모듈당 하나의 기능를 가지는 것이 기본이다.

모듈당 기능이 적으면 수정시 다른 모듈에 미치는 영향이 적으며 추출 을 간단히 할 수 있기 때문에 원시 프로그램의 수정과 갱신이 용이하다. 다음은 C 언어로 작성된 모듈에 대하여 각 모델의 요인항목의 정량화 방식을 설명한다.

기능 사이트 모델에는 출구수, 블럭수, 커멘드(Command)의 카테고리 수, 함수의 통합수, 가장 바깥 레벨의 블럭수, 2번째 블럭수 등 모두 6개 의 요인 항목을 설정한다.

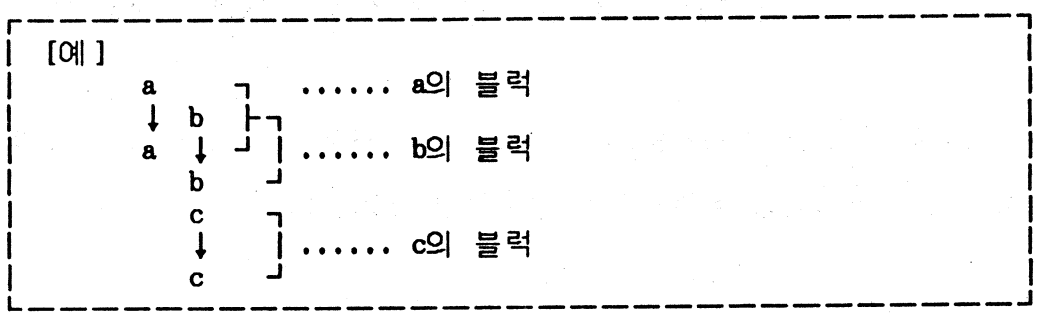
1) 출구의 수

Return문의 수와 Exit문의 수의 가중 평균으로 출구수를 계산한다.

2) 블럭의 수

블럭수는 내부변수의 유효범위로 구분된 블럭수를 계산한다. [예] a→a, b→b, c→c의 3개의 블럭을 표시하면 다음과 같다.

a와 b블럭은 서로 그 유효범위가 중복되기 때문에 동일한 블럭이 된다. 그러나 c블럭은 중복되어 있지 않기 때문에 다른 블럭으로 본다. 따라서



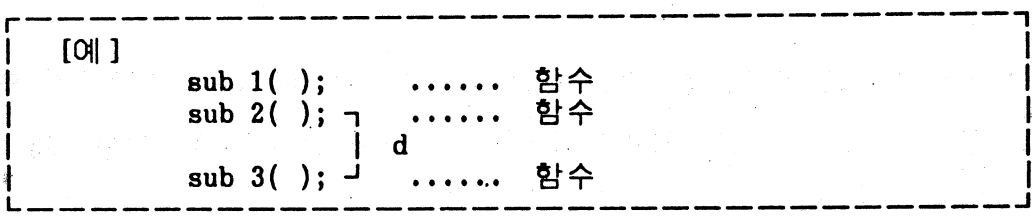
a, b블럭 1개, c블럭 1개로서 블럭수는 2이다.

3) 커맨드의 수

라이브러리(Library) 함수, 시스템 호출(System Call)의 수를 계산한다.

4) 함수의 통합수

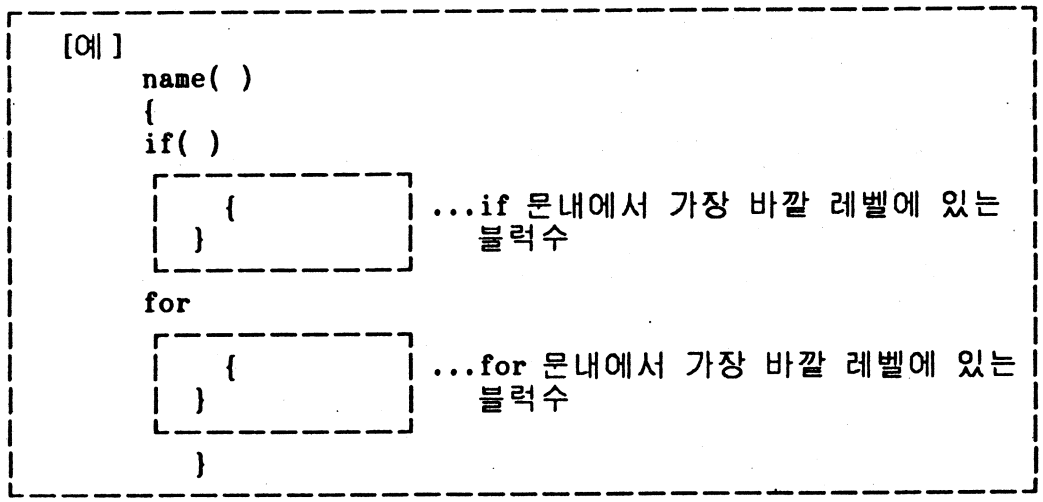
일정 간격을 둔 함수군의 수를 구한다.



위와 같이 함수내에 일정 이상의 거리 d가 만난 경우, 이 Sub 3을 다시 함수군으로 간주하고 그 수를 계산(Count) 한다.

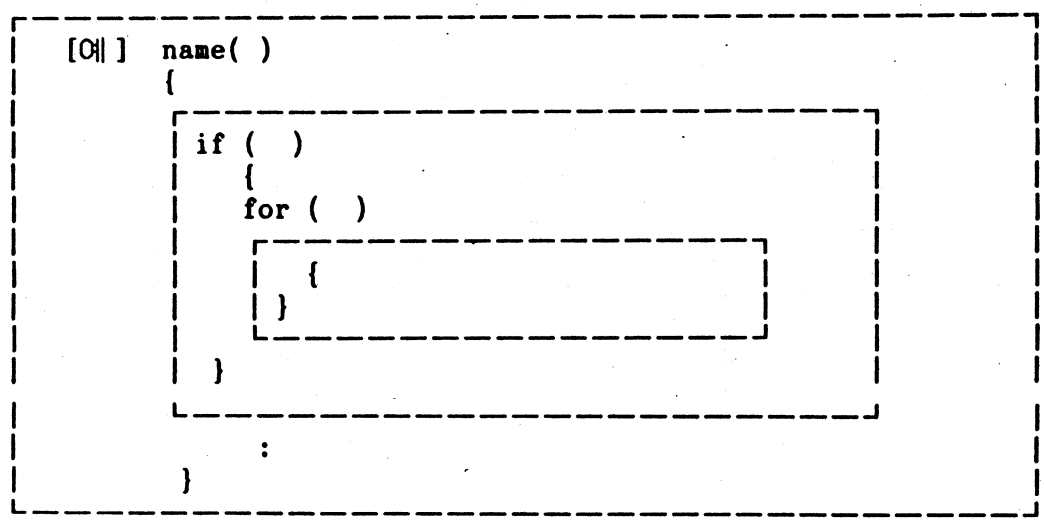
5) 가장 바깥 레벨의 블럭수

If, Else등의 조건문과 For, While등 반복문으로 작성된 블럭 ("{"과"}"이 생략되어 있는 경우도 카운트한다.) 가운데 함수의 가장 바깥에 있는 블럭의 수를 계산한다.



6) 2번째의 블럭수

If, Else등의 조건문, For, While등의 반복문으로 작성된 블럭("{"과"}"이 생략된 경우도 계산한다.) 가운데 관계내에서 2번째 레벨에 있는 블럭수를 계산하다.



1.2 읽기쉬운 모델의 요인 항목

원시 프로그램을 읽기 쉽도록 정량화 하기 위한 것으로 읽기 쉬운 모델은 작성된 원시 프로그램을 정량적으로 표현하여 프로그램을 쉽게 이해

할 수 있도록 한다는 생각에 기초를 두었다.

이러한 것은 보수시 영향을 주는 범위가 적절히 추출되므로 원시 프로그램의 수정, 갱신이 용이하다.

읽기 쉬운 모델의 요인항목에는 커맨드율, 벡터의 유무, 변수의 유효범위 평균치, 스텝수, 출구수, 블럭 평균길이, 내부변수 밀도, 전역(Globa 변수의 밀도, Go To문의 수 등 모두 9개의 요인항목을 설정하였다.

1) 커맨드 율

커맨드율은 모듈 충문자수에 대한 커맨드 문자수와 실행 명령문에 대한 커맨드 사용회수의 비율로 구한다. 커맨드율 R은 다음 식으로 계산한다.

```
R = (Cw/Mw) * (Ct/St)
                단, Cw : 커맨드 문자의 수
                    Mw : 모듈 충문자의 수
                    Ct : 커맨드의 사용회수
                    St : 실행 명령문의 수
```

2) 변수의 유효범위 평균값

내부 변수 및 함수의 인수 유효범위의 평균값을 구한다. 유효범위는 정의된 장소를 제외한 (최후에 나타나는 행번호) - (최초에 나타나는 행번호) + 1의 식으로 표시한다.

$$\text{평균} = \frac{\sum_{n}(\text{유효범위} * Wr)}{n}$$

단, n : 변수의 종류.
Wr : 유효범위의 중량.

3) 벡터(vector)의 유무

벡터는 함수의 정의 직전 또는 직후에 기술한 커맨드의 수이다.

```
[예]
   /*Vector-1*/      ..... 함수의 정의의 직전
   name 1( )         ..... 함수 정의
     {
    :
   }
   name 2( )         ..... 함수 정의
     /*Vector-2*/    ..... 함수 정의의 직후
     {
   }
```

4) 스텝 수

함수 선두의 "{"에서 마지막의 "}" 까지의 스텝 수를 말한다.

5) 출구 수

Return 문의 수와 Exit 문의 수의 가중 평균으로 출구수를 구한다.

6) 블럭의 평균 길이

If 문과 Switch 문의 블럭내 문자의 평균 길이이다.

```
[예] If 문의 경우
     if ( )
       {
      ┌──────────┐
      │    A     │
      └──────────┘
     }
     else
       }
      ┌──────────┐
      │    B     │
      └──────────┘
     }
```

평균길이=|A|+|B|/2...|A|>=n 또는 |B|>= n인 경우

0...나머지 경우

단, |A| : 블럭 A의 총문자수, n : 패러미터(parameter)

```
[예] Switch 문의 경우
          switch ( )
            {
          case o :
          case x :
          case ▲ :
          }
```

평균길이 = Na / Na ... Na / Nc >= n의 경우

o ... Na / Nc < n의 경우

단, Na : Switch문 내의 총 문자수

Nc : Case문의 수

n : 패러미터

7) 내부 변수의 밀도

내부 변수의 밀도는 스텝 수에 대한 내부 변수의 수로 한다.

```
내부 변수의 밀도 = 내부 변수의 수 / 스텝수
```

8) 전역 변수의 밀도

스텝 수에 대한 전역 변수의 수로 한다.

```
전역 변수의 밀도 = 전역 변수의 수 / 스텝수
```

9) GO TO문의수

GO TO문의 수를 계산한다.

다. 복잡한 모델의 요인항목

프로그램 로직(Logic)의 복잡성을 정량화하는 것으로 복잡한 구조

모델에서는 다수의 모듈로부터 구성된다. 이 모듈 간의 관계에 착안하여 단순 명쾌한 결합 모듈군이 요구된다는 생각에 기초를 둔다.

단순하고 명쾌한 결합이면, 보수시 영향을 주는 범위가 적어지므로 원시 프로그램 수정 및 갱신이 용이하다.

복잡한 모델에는 실행문수, 할스테드(Halstead)의 Vocabulary site, 프로그램 길이 프로그램 볼륨, 맥케이브(McCabe)의 Cyclomatic 수, 정보 비트등 모두 6개의 요인 항목이 있다.

1) 실행문의 수

프로그램의 ;의 수를 계산하여 구한다.

2)할스테드의 Vocabulary Site

Vocabulary Site는 오퍼레이터 수와 오퍼랜드 수를 합한다.

```
Vocabulary Site = 오퍼레이터 수(n1) + 오퍼랜드 수(n2)
```

여기서 오퍼레이터는 오퍼랜드 이외의 기호로서 그 수를 오퍼레이터 수라고 말한다.

오퍼랜드는 '-' 또는 영문자로 시작하는 -과 영.숫자의 열로서, 그 수를 오퍼랜드 수라고 한다.

3) 할스테드의 프로그램 길이

프로그램 길이는 오퍼레이터 총출현 회수와 오퍼랜드 총출현 회수의 합이다.

```
프로그램길이=오퍼레이터 총출현회수(N1)+오퍼랜드 총출현회수(N2)
```

4) 할스테드의 프로그램 볼륨

프로그램의 길이와 Vocabulary Site와의 관계를 대수적으로 나타낸다.

프로그램 볼륨 V = (N1 + N2) log2 (n1 =n2)

5) 맥케이브의 Cyclomatic 수

제어흐름 그래프 상에서 구한 Cyclomatic 수 이다.

$$C(G) = e - n + 2p$$

단, e: 에지의 수
n: 노드의 수
p: 연결성분의 수

6) 정보량 비트(bit)

정보량 비트 K는 다음 식으로 구한다.

X = Xi (Xi는 i라고 하는 오퍼랜드 또는 오퍼레이터의 출현회수)

$$K = X \log_2(X) - \sum_i (Xi \log_2(Xi))$$

라. 구조 모델의 요인항목

구조 모델에서 모듈 결합의 복잡도를 정형화하기 위해 모듈간의 결합도에 착안하여 단순명쾌한 모듈의 집단이 바람직하다.

따라서 보수시에 영향을 미치는 범위를 적절하게 추출할 수 있기 때문에 원시 프로그램의 수정과 갱신을 용이하게 할 수 있다.

설정한 구조 모델의 사용도 합계, 모듈당 출구수, 다른 모듈의 호출도합, 호출수의 표준 편차 등 4개의 요인항목을 설정한다.

1) 전역 데이타의 사용도 합계

전역 데이타의 사용회수와 구조 중별로 곱하고 합하여 구한다.

사용회수에는 그 사용 방법으로서 참조와 갱신별로 사용회수를 생각한다. 구조 종별로는 단독, 배열, 구조체의 배열 및 각각의 포인트(point) 사용 유무가 있다. 또 평가를 위해서는 전역 데이타의 사용도합을 최소한으로 지양한다.

구체적으로 전역 데이타의 갱신은 가능한 적게,또 단독 혹은 구조체의 배열과 다중 포인터(Pointer)를 사용하는 전역 데이타는 가능한 사용하 지 않도록 하는 것이 개선의 효과가 있다.

2) 모듈당 출구수

소프트웨어를 구성하고 있는 전체 모듈의 출구수(Return문, Exit문)의 총합을 모듈 총수로 나눈 평균값이다.

Return 문과 Exit 문에 계산용 패러미터에 의한 가중치를 준 가중평균이라고 볼 수 있다.

값이 작을수록 좋으므로 출구수를 많이 가진 모듈일 경우 복잡성이 증가하기 때문에 모듈 분할등에 의해 모듈 한개당 출구수를 적게하는 것이 효과가 있다.

3) 다른 모듈의 호출도 합

평균 함수 호출에서 소프트웨어를 구성하고 있는 전체 모듈의 상호 호출관계의 복잡성을 정량화 한다. 전체 모듈에서의 함수호출 총합을 모듈 총수로 나눈 평균 값이다.

4) 호출수의 표준 편차

함수 호출수의 표준 편차에 대한 것으로 소프트웨어를 구성하고 있는

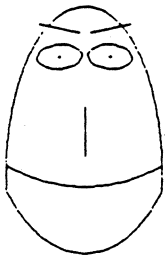
전체 모듈중 여러 곳의 모듈을 마치 중앙 터미널(Central Terminal)과 같이 제어(Case 문)를 집중시키는 경우 중요한 모듈에 주의력을 집중할 수 있다는 점에 착안, 구조를 정량화하는 방법이다.

이것은 평균값과 각 모듈 상호의 함수 호출수를 사용하여 표준편차를 구한다. 측정값이 클수록 좋은 평가이기 때문에 제어 전용 모듈 (조건 분기 만을 행하고, 상호 분기 간에 모듈을 호출한다)을 작성하여 명료한 구조가 되도록 하는 것이 좋다.

1. FACE에 의한 종합평가 - FACE -

< 표시형식 >　종합평가의 결과를 FACE의 Illustration으로 표시하여,전체MODEL을 조합하여 평가의 좋고 나쁨을 얼굴의 표정으로 평가합니다.

표　제	종 합 평 가		
MODEL	기능 SIZE, 읽기쉬운 정도, 구조, 복잡성		
SOFTWARE 명	ice_soft　　v1.0		
MODULE 명	종　　합		



좋은 얼굴이 어느 정도인가?			
기능SIZE	눈꼬리가 치켜 올라갔다	읽기쉬운정도	입이 웃고 있다
구　조	입이 크다	복잡정도	눈섭,눈이 가깝다

최종 해석일	87/01/07/11:32:43
표시 형식	FACE
종　별	종합평가

선택해 주십시요!■　0.표시종료　1.표시변경

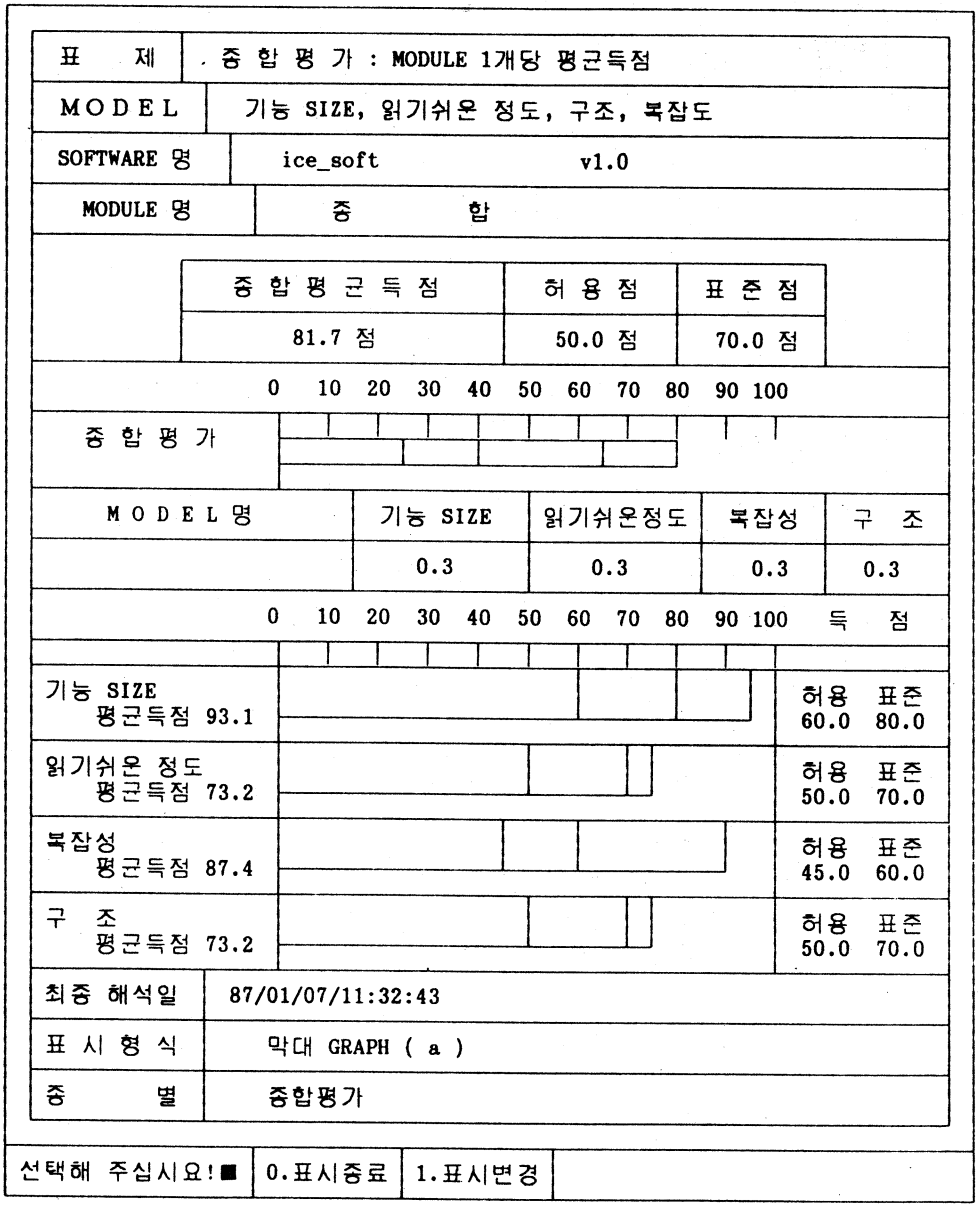
『FACE의 좋은 정도』

기 능　S I Z E :눈꼬리가 치켜 올라 갔다.
구　　조　:입이 크다.
읽기 쉬운 정도 :입이 미소를 띠우고 있다.
복 잡 한　정 도:눈섭이 눈 가까이 있다.

2. MODEL의 내용에 의한 종합평가 - 막대 GRAPH -

평가한 SOFTWARE의 종합득점에 각MODEL의 종합득점을 점수와 막대GRAPH로 표시하여, 허용점과 표준점을 나타내고 보수성 품질평가 결과를 명확히 합니다.

< 표시형식 > 종합평가로서 SOFTWARE의 종합득점을 막대 GRAPH로 표시하고 선택된 MODEL의 순서로서 각 MODEL마다의 막대GRAPH로 표시합니다. 또한 MODEL의 평균득점,허용점,표준점을 표시합니다.

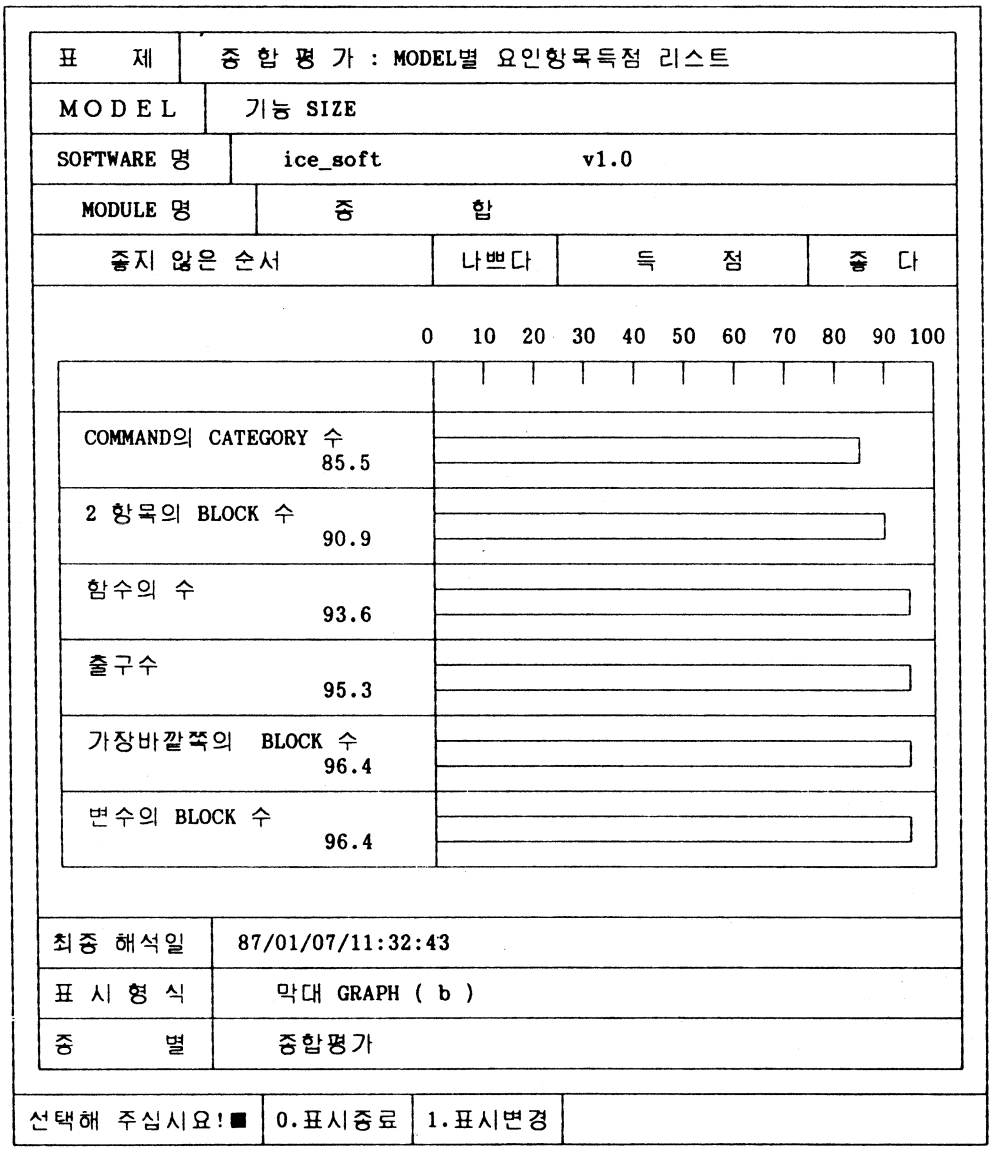


● 막대 GRAPH에 의한 평가방식 ●

3. 지정 MODEL의 요인항목에 의한 종합평가 - 막대 GRAPH -

1 개의 MODEL에 착안하여 그 MODEL중의 어느 요인항목을 평가하는 것이 좋았는지,또는 나빴는 지를 나타냅니다. 그리고 좋지않았던 요인항목을 막대GRAPH로 표시합니다.

< 표시형식 > 지정된 MODEL에 대해 요인항목의 평가를 좋지 않은 순으로 요인항목명과 막대GRAPH로 표시합니다.



4. 지정 MODEL의 1 요인항목에 의한 MODULE의 종합평가 - 막대 GRAPH -

1 개의 MODEL에 착안하여 그 MODEL이 포함한 요인항목중 한 개의 목표를 정하여,그의 요인항목에 MODULE의 종합평가를 실시하여 막대GRAPH로 표시합니다.

< 표시형식 >

표 제	종 합 평 가 : 요인항목별 MODULE의 득점 리스트			
MODEL	기능 SIZE			
SOFTWARE 명	ice_soft v1.0			
MODULE 명	종 합			
○ 요 인 항 목	출 구 수			
좋지 않은 순서	나쁘다	득 점	좋 다	

0 10 20 30 40 50 60 70 80 90 100

MODULE 명	
getcc 84.0	
what_fil 92.0	
point_fn 92.0	
nck_neas 92.0	
main 94.0	
get_text 94.0	
output_r 100.0	

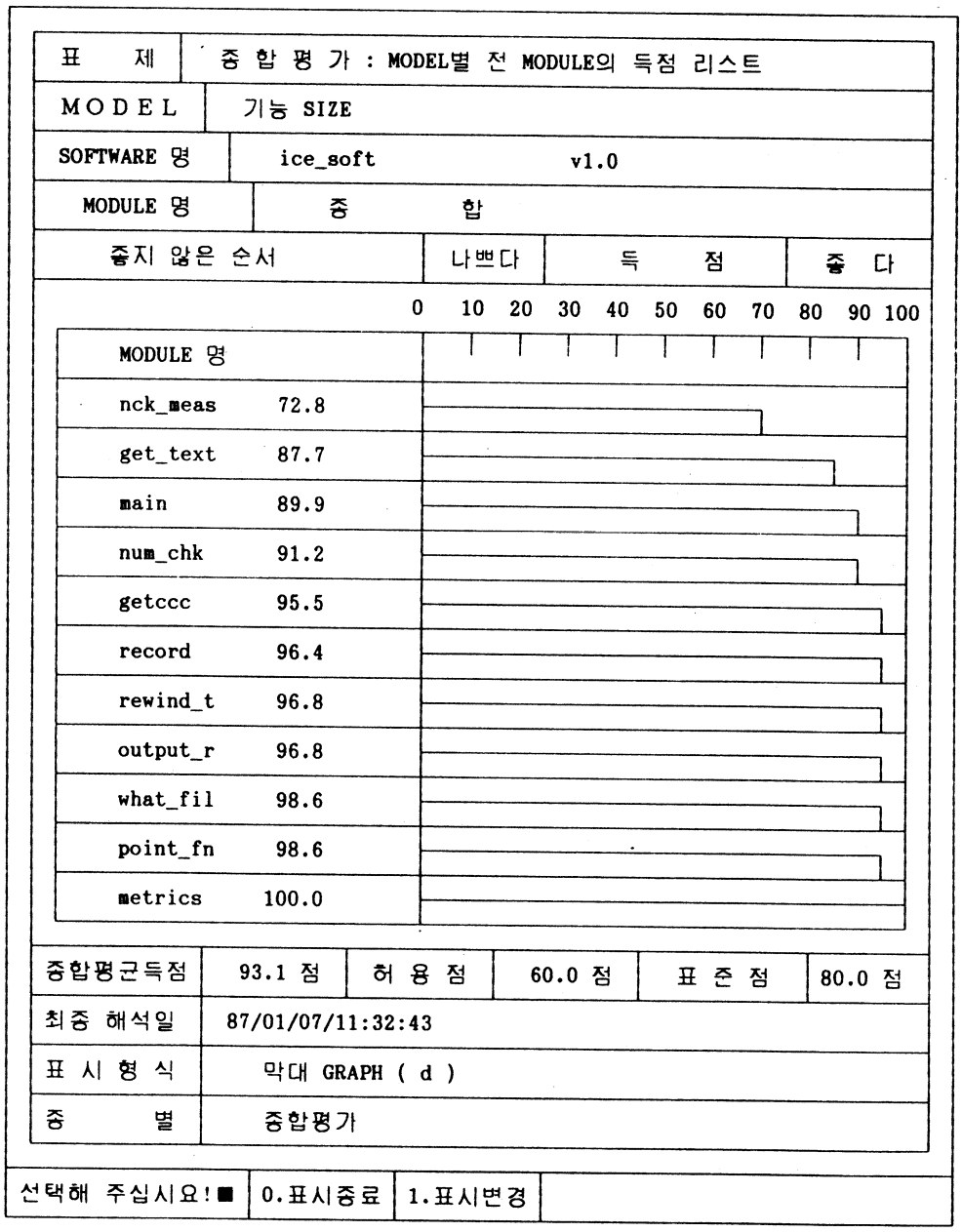
종합평균득점	95.3 점	허 용 점	50.0 점	표 준 점	70.0 점
최종 해석일	87/01/07/11:32:43				
표 시 형 식	막대 GRAPH (c)				
종 별	종합평가				

선택해 주십시요!■	0.표시종료	1.표시변경	2.앞 PAGE	3.다음 PAGE

5. 한개의 MODEL에 의한 MODULE의 종합평가
- 막대 GRAPH -

1 개의 MODEL에 착안하여 SOFTWARE를 구성하는 전 MODULE하나하나에 대하여 MODEL의 종합평가를 실시하여 막대GRAPH로 표시합니다.

< 표시형식 >



6. 지정 MODEL의 요인항목별 상세정보
- TABLE -

1 개의 MODEL에 착안하여 그 모델에 포함되는 요인항목의 측정치로 부터 득점까지의 상세수순에 의해 상세정보를 TABLE로 표현한다.

< 표시형식 >

표 제	종 합 평 가 : MODEL별 요인항목득점 리스트
MODEL	기능 SIZE
SOFTWARE 명	ice_soft v1.0
MODULE 명	종 합
○ 평 균	

요인항목명	측정치	요인득점	요인치	득 점
출구치	0.2	95.3	0.18	17.1
변수의 BLOCK수	1.4	96.4	0.18	17.3
COMMAND의 CATEGORY수	0.7	85.5	0.16	13.7
함수의 수	2.3	93.6	0.16	15.0
가장 바깥쪽의 BLOCK수	0.4	96.4	0.16	15.4
2 항목의 BLOCK수	0.5	90.9	0.16	14.5

MODEL 합 계	기 능 SIZE	93.1

최종 해석일	87/01/07/11:32:43
표 시 형 식	TABLE
종 별	MODULE별 평가

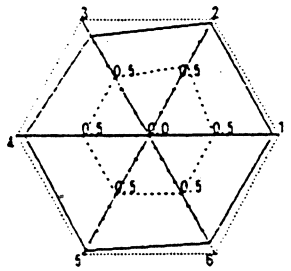
선택해 주십시요!■	0.표시종료	1.표시변경	

7. 지정 MODEL의 요인항목에 의한 평가경향
- RADAR CHART -

1 개의 MODEL에 착안하여 그 모델에 포함되는 요인항목의 평가경향을 RADAR CHART 로 분명히 합니다.

< 표시형식 >

표 제	종 합 평 가 : MODEL별 요인항목 경향
MODEL	기능 SIZE
SOFTWARE 명	ice_soft v1.0
MODULE 명	종 합



	— 평가치 ... 허용치
1	출구수
2	변수의 BLOCK수
3	COMMAND의 CATEGORY수
4	함수의 수
5	가장 바깥쪽의 BLOCK수
6	2 항목의 BLOCK수

최종 해석일	87/01/07/11:32:43
표 시 형 식	RADAR CHART
종 별	종합평가

선택해 주십시요!■	0.표시종료	1.표시변경	

8. 각 MODEL의 각 요인항목에 의한 품질향상의 개선전략
- COMMENT -

각 MODEL의 각 요인항목에 대하여 MODULE의 품질향상을 꾀하기 위하여 개선전략의 COMMENT를 표현합니다.

< 표시형식 >

표 제	종 합 평 가 : 프로그램의 개선전략
M O D E L	읽기 쉬운 정도
SOFTWARE 명	ice_soft v1.0
MODULE 명	종 합

노 트

COMMENT를 붙여서 엄밀한 설명을 붙이시오

COMMENT를 붙여서 엄밀한 설명을 붙이시오

변수선언, if case문의 에 COMMENT를 붙이시오

알기쉬운 방법을 붙이시오

최종 해석일	87/01/07/11:32:43
표 시 형 식	COMMENT
종 별	종합평가

선택해 주십시요!■	0.표시종료	1.표시변경	2.앞 PAGE	3.다음 PAGE

2. 끝 맺 음

일본 도시바에서 개발한 ESQUT에 적용한 소프트웨어 품질을 평가할 수 있는 주요한 요인항목을 설 정하고 측정 방법을 고찰하였다.

소프트웨어 모델을 첫째, 모듈기능 사이트 모델의 요인 항목은 6가지 둘째, 읽기쉬운 모델의 요인항목은 9가지 셋째, 복잡한 모델의 요인항목 은 6가지. 넷째, 구조모델의 요인항목은 4가지로 각각 분류하고 각 모델 별로 품질을 평가할 수 있는 요인항목과 조건및 측정방법을 제시하였다.

ESQUT의 앞으로의 연구 과제는 각 요인 항목의 타당성 검증에 있다고 하겠다.

참 고 문 헌

1. Miller, E.F, "Program Testing:Art Meets Theory",IEEE Computer, Vol. No.7 pp. 42-51, 1977.
2. T.J. McCabe, "A Complexity Measure", IEEE Trans. Soft. Eng., Vol. SE-2, No.4, pp. 308-320, 1976.
3. Halstead, N.H., "Elements of Software Science", New-York, Elsevier North Holland, 1977.
4. Masaak, Kuribauyashi, Makoto Takeuchi, Yoshiak, katayama, "Evaluation Method for software by Quality Control Table",Information Processing Society of Japan, 1986.
5. Makoto Sato, "ESQUT" Software Quality Evaluation System, Information Processing society of Japan, 1986

6. Yamada, S. and Osaki, S., "Applied Stochastic Models and Data Analysis", Vol. 1, pp. 65-77, 1985.

7. Notiko Hashimoto, a.Morita, R.Yumita, Y.Ishihara, K. Kobayashi, "Quantitative Evaluation Method of Project in Software Development," Information proccessing Society of Japan, 1986.

8. TOSHIBA, "소프트웨어 품질 평가 Tool ESQUT", IMAP 관리시스탬, 소프트웨어 품질평가, 1987.

9. 양 해술, Toshiro ARAKI, "Refinement on Complexity and Structuredness of Program's Control flow", Information Processing society of Japan, Vol. 56-5, 1987.

10. 양 해술, 백 청호, 이 창석, "소프트웨어 품질 평가를 위한 요인 항목에 관한 연구", 한국정보과학회, 88' 가을 학술발표논문집, 1988.

11. 양 해술, "컴퓨터 프로그램 품질보증의 기준과 실제", 한국정보산업연합회, 정보산업, 1988.

12. 양 해술, "소프트웨어 품질 평가항목",컴퓨터 정보사, 컴퓨터 월드, 1988.