特輯 技術解說

전자교환 시스팀의 소프트웨어 품질 및 신뢰도 보증

李 圭 旭
韓國電子通信研究所 品質保證室

Ⅰ. 서 론

소프트웨어가 점차 다양화, 대형화, 고도화되어 이의 개발비용이 증가하는 반면 하드웨어는 경박단소화 추세로 제비용이 감소되고 있어 시스팀 개발에서 소프트웨어가 차지하고 있는 비중이 날로 증대되고 있다. 전자교환 시스팀의 하드웨어 품질 보증은 작성된 개발지침서 및 설계표준서에 따른 품질 평가 항목 설정 및 이에 따른 평가, 표준화 점검, 시험 (기능, 성능, 신뢰성 등)등을 통하여 이루어지고 있으며, 신뢰도 보증은 환경시험에 의해 입증된 부품 고장율 공식에 따라 부품 신뢰도를 계산하고 시스팀 구조 및 cell path에 따라 신뢰도 계산 block diagram을 작성한 후 이에 따라 전체 하드웨어 신뢰도를 계산, 목표 신뢰도와 비교하여 필요시 취약부품 및 구조를 개선하는 단계로 이루어진다.

한편, 소프트웨어의 품질보증은 각 개발단계에서 필요한 품질보증 활동을 규정하고 이를 수행, 평가, 관리하므로 이루어지며 신뢰도 보증은 시험에 의해 발견 및 fix된 에러를 정량화하고 신뢰도를 예측할 수 있는 적정모델을 개발한 후 이에 의한 신뢰도 계산치를 목표 신뢰도와 비교하여 소요 시험기간을 결정하는 방법으로 이루어지고 있으나 소프트웨어 개념의 난해성 및 관리체계의 다원화로 체계적인 방법정립이 난해하다. 하드웨어 품질보증이 잘 되어있지 않아 보드의 고장이 자주 발생한다면 spare 보드를 보다 많이 확보하여 고장시 마다 보드를 교체하면 어느정도 해결이 가능하지만 소프트웨어 품질보증이 잘 되지 않은 채 시스팀이 release되어 소프트웨어 에러가 발생하면 이를 수정할 때까지 시스팀이 down되는 결과를 초래하므로 소프트웨어 품질보증이 하드웨어 품질보증 보다 중요하다고 생각된다.

요즈음, 소프트웨어의 생산 및 유지보수 비용이 증대되어 이의 고장 현상을 발견하고 발견한 고장을 이용하여 신뢰도를 예측하는 활동이 활발히 진행되고 있다. 대규모 시스팀의 소프트웨어를 효과적으로 관리하고 평가하기 위해서는 개발 최종 제품 뿐만아니라 개발 전과정에서 소프트웨어를 정량화하여 평가하는 활동이 수행되어야 하며 개발자 자신은 물론 개발 관리자가 주어진 규정에 따라 개발과정을 끊임없이 검사, 조정, 지시, 확인해야 한다. 본문은 TDX-10, 5ESS, AXE-10의 전자교환 시스팀에서 수행되는 소프트웨어 품질 및 신뢰도 보증활동을 종합, 검토하여 시스팀 개발시 수행을 요하는 항목 위주로 작성되었으며 전자교환 시스팀의 개발 및 운용중 필요한 품질 및 신뢰도 보증활동의 지침이 되었으면 좋겠다.

Ⅱ. 소프트웨어 품질보증

1. 소프트웨어 품질보증 범위

소프트웨어 품질보증의 대상과 단계는 소프트웨어의 계획, 개발, 시험, 운용관리 등 개발의 전 순기동안에 수행되는 품질보증 활동을 말하며, 여기서 소프트웨어는 교환기능을 실현하기 위한 호 처리관련 소프트웨어, 운용 소프트웨어, 유지보수 소프트웨어, 운영체계, 데이타 베이스 관리체계, 컴파일러, 디버거 등 각종 프로그램을 포함한다. 먼저, 전자교환 시스팀의 소프트웨어 순기별 품질보증 활동은 그림1과 같다.

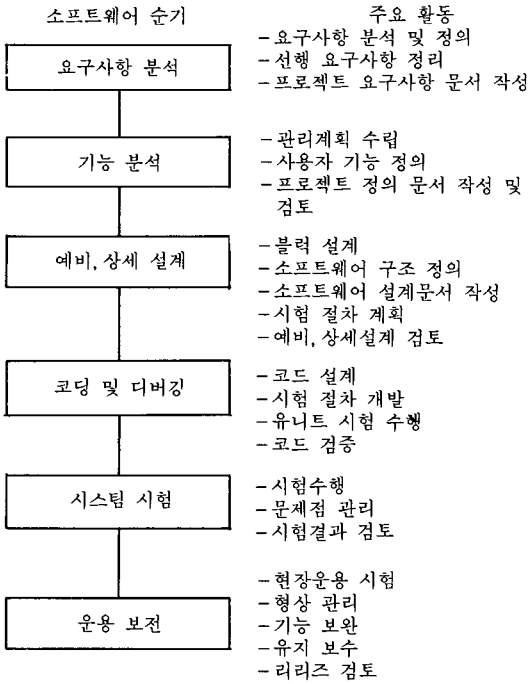

그림 1. 개발 순기별 소프트웨어 품질보증 활동

2. 품질 평가

일반적으로 소프트웨어 품질평가를 위해서는 종합적이고 체계적인 품질특성의 기준이 요구되며 이 기준을 정립하기 위해서는 소프트웨어 품질을 이해, 검토, 평가해야 한다. 특히, 소프트웨어는 하드웨어와 달리 보는 관점에 따라 품질이 여러 관점으로 파악될 수 있으므로 정확한 품질평가를 위해서는 요구되는 품질특성을 명확히 정의해야 한다. 먼저, 사용자 관점에서 개발 전과정중 개발 및 평가의 지표가 되는 품질목표 항목을 살펴보면 표1과 같다.

1) 품질목표 항목

(1) 정확성

요구사항, 개발지침서(시스팀, 소프트웨어, 하드웨어 설계지침서 및 시험지침서), 각종 메뉴얼에 기술된 내용이 상호 모순없이 일관성을 유지하고 있는지, 일관성을 벗어날 경우 타 내용과 마찰이 발생하는지 확인하고 소프트웨어와 관련한 모든 설계와 기능등을 확인하여 소프트웨어 오류에 대한 질적 양적 평가를 수행한다.

관련 기능의 수행을 위한 소프트웨어의 정확도, 운용중의 상태변화, 감시, 복구, 추적, 데이타 수정, 입출력 현황, 기술문서의 정확도를 점검하며 여기서 정확도는 개발지침, 사용자 요구사항, 사용자 기능, 운용메시지, 소프트웨어 개발방법론 및 설계방법론, 프로그래밍 표준화 지침, OS 메뉴얼, 디버거 메뉴얼 및 SDL 환경 메뉴얼을 기준으로 한다.

(2) 신뢰성

시스팀이 주어진 환경하에서 주어진 시간동안 사용자 요구사항 및 품질보증 계획서에 명시된 신뢰도

표 1. 품질목표 항목

품질범위	품질목표 항목	정 의
제품운용	정 확 성	소프트웨어가 사용자가 요구하는 규격 또는 목표를 만족하는 성질
	신 뢰 성	소프트웨어가 오류없이 정확하고 일관된 결과를 제공하며 운용되는 성질
	효 율 성	소프트웨어가 최소의 처리시간과 기억장치를 사용하면서 요구된 기능을 수행하는 성질
	무 결 성	시스팀의 안전을 위하여 인가되지 않은 엑세스로부터 소프트웨어나 자료를 보호하는 성질
	사 용 성	학습, 운용, 입력 준비, 결과분석 등의 소프트웨어 사용이 용이한 성질
제품개발	유지보수성	운용 과정에서 발생한 결함의 원인을 찾고 그것을 교정할 수 있는 성질
	시 험 성	소프트웨어가 의도하는 기능을 제대로 수행하는지 입증하는 성질
	융 통 성	정상기능을 수행하는 소프트웨어에 새로운 요구사항을 반영시 프로그램을 변경 또는 보완하기에 알맞는 성질
제품전이	이 식 성	소프트웨어를 새로운 하드웨어 또는 시스팀 환경으로 옮겨 수행하는 수정작업이 쉬운 성질
	재 사 용 성	소프트웨어의 일부 또는 전부가 다른 응용분야의 소프트웨어 개발에 다시 사용할 수 있는 성질
	상호운용성	소프트웨어가 다른 시스팀과 연결되어 정보를 교환하기에 좋은 성질

및 가용도를 만족하는 결과를 시스템이 제공할 수 있는지 모든 소프트웨어 성능요소를 확인한다. 이외에 트래픽 처리용량, 호처리 성능기준, 서비스 기준, 호 지연 기준, 호 손실율에 관계된 교환기 프로그램 성능도 측정한다.

(3) 무결성

프로그램 및 주요 데이타에 대해 지속적으로 타이밍 또는 데이타 리던던시를 이용하여 외상여부를 검출하고 오류 분석, 오류 발견 및 복구, 외부 출력 수행으로 시스템의 오동작과 성능저하를 방지한다. 이를 위하여 전자교환 시스템에서는 타교환기의 응용 프로그램 데이타를 공유할 수 있도록 이들 프로그램이 사용하는 데이타를 프로세서의 성격에 따라 해당 데이타를 종합적으로 유지하고 관리하는 DBMS (data base management system)를 이용하고 있다. 데이타 베이스의 대상 데이타로는 가입자 및 중계선, 스위치, 번호 번역, 시스템 형상, 경보 및 장애, 호처리, 유지보수, 과금, 통계 관련 자료가 있으며 DBMS를 이용하여 데이타의 독립성 보장, 중복 데이타 감소, 데이타의 보완 유지, 데이타의 무결성 유지, 데이타의 동시 제어기능 등이 가능하다.

(4) 사용성

사용자가 이용하는 각종 기술문서, 운용자 메뉴얼 등의 사용의 용이성 및 보고의 편리성을 점검한다. 특히, 교환기 프로그램이 다양하고 방대하며 시스템 개발자와 운용자가 서로 상이한 동시에 요구사항 정의, 설계, 시험, 유지보수 등 여러 개발단계를 거치는 동안 담당자가 바뀔 수 있으므로 각종 기술문서는 개발자와 운용자 및 기타 관련자에게 사용이 편리하고 관리가 용이하도록 사용성의 점검을 철저히 할수록 좋다.

(5) 유지 보수성

시스템 운용중 통화로 장비, 입출력 장비, 경보장치, 주변장치 등에서 on-line 및 on-demand로 장애의 원인을 검출하고 이를 분류, 분석, 복구, 확인하며 장애격리에 따른 경보처리, 장애발생 및 처리결과 출력, 데이타 관리 및 보고기능을 위하여 프로그램이 어떻게 운용되는지 평가한다.

(6) 시험성

시스템이 기능수행 확인을 위하여 프로그램의 구조적 독립성, 단순성, 모듈화를 이루고 있는 정도와 시험기능의 구비 정도를 판단한다. 특정 특성 및 내용이 추가되어 수정 보완이 이루어질 경우 필요 이상의 시험을 수행하지 않고 품질을 보증할 수 있도록 해야하며 수정으로 인하여 다른 부분에 큰 영향이 미치지 않도록 시험 항목들이 구조화를 이루고 있어야 한다. 또한, 요구 사항에서 제시한 모든 항목들은 시험에 의해 그 기능이 검증될 수 있어야 하며 가능한 검증 내용은 명확하고 정량적으로 기술되어야 한다. 요구사항 정의단계에서 동작규격, 블럭설계, 실현 단계에 이르는 동안 필요한 정보의 추적이 용이해야 하며 interworking 관련 정보를 손실하지 말아야 한다. 또한 기능 구현의 기본 단위인 소프트웨어 블럭은 독립성을 고려해서 블럭간 주고받는 메시지를 가능한 줄이도록 하고 메시지 전송 수신시 타 블럭의 경유를 지양하도록 해야하며 블럭은 시험과 관리가 용이하도록 크기를 적당하게 해야한다.

(7) 재 사용성

교환기 프로그램은 구조상 서로 유사한 기능을 갖는 부분이 많이 있어 약간의 프로그램 수정으로 서로 재사용이 가능하다. 일반 가입자의 기본 호 처리 기능, 특수호 처리기능, 원격 가입자의 호처리 기능, 가입자 및 중계선의 busy/free 검색기능 등은 서로 밀접한 상관관계를 갖고 있어 프로그램의 기본 구조를 서로 이용할 수 있고 해당 부분에 대하여 전달성, 독립성, 모듈화가 이루어지면 보다 쉽게 프로그램의 재사용도 가능하다.

2) 품질기준 설정

품질기준은 개발하고자 하는 소프트웨어의 목표를 보다 기술적인 측면에서 구체화 시킨 품질 요소로서 개발제품에 대한 품질평가 척도의 최상위 레벨인 품질 요인중 적용 가능한 것들의 집합으로 이루어진다. 품질기준은 시스템 개발 전 순기에 걸쳐 개발 및 평가의 지표가 되므로 기능적 성능, 기술적 요구사항, 시스템 특성, 비용, 서로 상충되는 품질목표 등을 고려하여 실현 가능하도록 설정해야 하는데 먼저 수행될 기능, 요구되는 시스템의 성능, 기술적 요구사항 중 적용될 특성을 기준으로 단계별로 주어진 품질기준 리스트를 작성하고 품질 요인간 상충관계, 보완관계 및 중요도를 고려하여 다음과 같이 설정된다.

- 안전도 : 시스템에 대한 엑세스 상황을 기록 통제하며, 인가되지 않은 엑세스로 부터 시스템을 보호하는 메카니즘을 제공하는 정도
- 정밀도 : 처리과정과 출력과정에서 오류없이 필요한 정보를 제공하는 정도
- 오류 관리도 : 소프트웨어 상의 결함 또는 사용자의 입력 오류 등 비정상적인 상황에서도 시스템이 계속 올바르게 수행될 수 있는 정도

• 공통도 : 타 시스팀과의 접속을 위해 표준화 또는 공통의 규약과 인터페이스 루틴을 채택하고 사용하는 정도

• 운용 용이도 : 소프트웨어를 손쉽게 적재, 초기화, 운용, 종료할 수 있으며 사용자에게 유용한 출력과 친밀한 접촉성을 제공하는 정도

• 완전도 : 소프트웨어가 사용자의 모든 요구사항을 충실히 포함하고 있는 정도

• 일치도 : 프로그램을 포함한 모든 개발제품이 균일성을 유지하며 상호 모순이 없는 정도

• 문서화 : 개발 규격이 규정된 양식에 따라 작성되고 내용의 명확 완전성, 프로그램이 이해하기 쉽게 작성된 정도

• 모듈화 : 시스팀을 구성하는 특정 요소의 변경이 다른 부분에 미치는 영향도가 최소가 되도록 각 부분들이 독립적인 기능을 갖도록 조직화된 정도

• 추적도 : 개발단계에서 생산되는 모든 개발제품이 이전 단계에서 설정된 개발규격을 충족하는 정도 및 서로 연결, 지원하는 정도

• 독립도 : 소프트웨어가 특정한 하드웨어나 소프트웨어 환경에 종속되지 않고 여러곳에 수행되기에 적절한 정도

• 처리 효율 : 소프트웨어가 최소의 실행시간으로 부여된 기능을 수행하는 정도

• 기억장치 효율 : 소프트웨어가 기억장치를 최소로 사용하면서 부여된 기능을 수행하는 정도

• 응답도 : 온라인 시스팀이나 일괄처리 시스팀이 사용자가 요구하는 출력을 지정된 시간내에 정확하게 제공하는 정도

• 확장도 : 새로운 기능을 추가하거나 계산, 처리 능력을 향상시키기 위한 변경작업이 쉬운 정도

• 보편도 : 소프트웨어가 사용하는 자료의 형, 표현, 구조 등이 표준화된 정도

• 단순도 : 소프트웨어 내부의 제어구조나 논리구조가 간단하고, 크기가 취급하기에 알맞은 정도

• 시험 적합도 : 소프트웨어가 완전한 기능수행을 입증하기 위한 시험계획 수립정도 및 시험하기에 적절한 정도

3. 품질평가 기법

소프트웨어 품질평가는 그림1과 같은 개발 전 순기에 수행되는 품질보증 활동을 대상으로 행해지므로 어느 한가지 기법보다는 각 개발단계의 특성에 맞는 평가기법의 사용이 요구된다. 평가기법은 크게 확인검증 (verification and validation)과 표준화 감사 (standardization audit)로 나눌 수 있다.

1) 확인 검증

확인검증은 어느 개발단계의 제품이 최초의 사용자 요구 또는 소프트웨어의 요구에 부합 여부를 입증하기 위한 확인 (verification)과 어느 단계의 개발제품이 이전단계에서 설정된 규격을 만족하는 지의 여부를 판단하는 검증 (validation)의 두 의미를 뜻한다. 확인검증은 검증평가(review and audit)와 시험평가 (test and evaluation)로 나누는데 검증평가는 시스팀 초기 단계에서 작성되는 요구사항, 개발지침서의 평가 위주로 수행되며, 시험평가는 프로그램을 대상으로 시험결과를 분석하여 시스팀 상태를 평가하므로 수행된다.

(1) 검증 평가

검증평가는 각 개발단계에서 수행된 개발 활동의 유효성 여부와 그 단계에서 산출된 개발 제품에 대한 달성수준을 평가하는 것이다. 이는 주로 개발초기에 요구사항 정의와 시스팀 설계과정에서 이용되며 검토 및 검사를 통하여 이루어진다. 검토의 기본 목표는 문제 해결 보다는 문제점 발견과 식별에 중점을 두고 있으며 주로 관련자가 회의를 통하여 의견을 제시하고 이를 검토후 자체적으로 문제점을 시정한다. 검사는 정형화된 절차에 의해 공식적으로 정정표와 평가기준을 사전에 확정한 후 가능한 상세하게 평가하며 수정에 대한 지침 또는 권고를 제시하고 시정조치 여부를 추적한다. 보다 자세한 검증평가는 2장 5절 검증에서 설명되었다.

(2) 시험 평가

시험평가는 실행이 가능한 소프트웨어 개발제품을 대상으로 기능을 확인 또는 검증하기 위해 시험절차를 입력하고 그 결과를 분석하는 활동으로 소프트웨어 개발주기 동안 단위 시험, 종합시험, 시스팀시험으로 구분된다.

• 단위시험 (unit test) : 상세 설계로 부터 프로그램 모듈에 할당된 기능이 올바르게 수행되는 지를 개발자가 자체적으로 검증하는 것으로 주요 과정은 프로그램의 자료 구조, 수행경로 등의 완전성을 입증한다.

• 종합시험 (integration test) : 모듈간의 인터페이스에 관련된 오류를 찾아내고 각각의 모듈이 하나의 시스팀으로 결합되어 올바르게 수행됨을 검증하기 위한 시험평가 활동으로 시스팀 설계 단계에

서 설정된 소프트웨어 구조 및 할당이 올바르게 되어있는지 입증한다.

• 시스팀 시험 : 개발단계의 마지막 검증단계로 소프트웨어 제품이 요구사항에 명시된 내용을 만족하고 있는지 확인하는 것으로 개발기관의 품질관리 부서가 설정된 품질목표를 달성하고 있는지 검증한다.

2) 표준화 감사

개발 전 순기에 걸쳐 적용되는 제반 표준들이 소프트웨어와 무리없이 적절히 운용되는지 평가하고, 각 단계에서의 개발과정과 개발제품이 적용된 표준에 부합 여부를 평가한다. 소프트웨어 개발에 사용된 표준, 관례 및 지침은 다음과 같다.

• 각 개발단계에서 적용하는 방법론, 기법, 도구의 정의 및 사용 요령

• 각 개발단계에서 생산되는 개발 제품의 형식, 상세도 수준

• 코딩 표준화 및 기술문서 표준

• 용어의 표준

표준화 감사의 주요 업무는 적용된 표준의 적합성, 개발과정 및 개발제품이 표준에 부합되는 정도를 파악하고 표준의 준수 여부를 검토하는 것이다.

4. 세부적인 품질 평가방법

세부적인 소프트웨어 평가 방법에는 개발 순기별로 평가항목을 작성한 체크리스트를 이용하여 평가하는 방법과 사용자 요구사항에 명시된 요구사항을 소프트웨어가 얼마나 만족하고 있는가를 확인하는 시험에 의한 방법으로 대별된다.

체크리스트에 의한 방법은 주로 소프트웨어 프로젝트 계획, 요구사항 분석 및 설계단계에서 체크리스트를 이용하여 수행되며 시험에 의한 방법은 시험계획, 시험 수행, 시험 결과서 처리 단계에서 수행된다.

1) 체크리스트에 의한 방법

(1) 소프트웨어 프로젝트 계획

이는 수립된 소프트웨어를 기초로 리소스, 비용, 일정 등을 평가하기 위한 계획으로 교환기 소프트웨어와 지원환경 소프트웨어, 데이타 베이스 등을 고려한 소프트웨어 기본구조를 정립하고 소프트웨어 개발 및 관리체계를 평가한다.

수행되어야 할 주요 사항들은 소프트웨어 개발목표 설정, 소프트웨어 작업 분할구조 결정, 이에 따른 일정계획 수립, 필요한 인력 및 개발비용 산정, 프로젝트 수행에 요구되는 조직 및 배치 설정이며 이러한 프로젝트 계획, 교육홍보, 계획관리, 조직관리, 개발관리에 대한 세부항목과 지침에 대한 수행여부의 적정도를 평가한다.

(2) 소프트웨어 요구사항 분석

요구사항은 개발하는 프로그램의 기본으로 완전성, 정확성, 시험성이 다음과 같이 체크된다.

• 정확성 : 요구사항의 내용은 정확하고 일관성이 있으며 완전한가?

• 판독성 : 요구사항을 이용하는 모든 관계자에게 이해가 용이하며 정의된 용어와 기술된 내용이 상세한가?

• 중복성 : 내용기술의 반복이 지양되고 필요시 해당부분의 참조로서 이해수단을 제공하며 불필요한 반복횟수가 없는가?

• 기타 : 개발하는 소프트웨어의 성능, 신뢰성 등의 기능적 특성과 개발완료시 패키징 방법, 설계방법, 실현시 고려사항 등이 언급 되었는가?

(3) 설계

소프트웨어 설계는 소프트웨어 요구사항을 기초로 소프트웨어의 기능 및 성능을 가장 적절하게 실현시킬 수 있는 알고리즘과 그 알고리즘에 의해서 처리될 자료 구조, 프로그램 구조 및 절차에 대한 특성을 찾고 이를 구체화하는 작업으로 설계후에는 설계가 요구사항에 대해 정확히 기술 되었는지 확인하는 설계검토가 필요하다. 이는 설계가 요구사항의 완전한 set를 나타내고 있는지, 설계가 완전하고 일관성이 있으며 애매 모호함이 없는지를 주로 평가하는 것으로 데이타, 로직 구조, 모델 구현, 입출력, 데이타 요구사항 등을 주로 자동화된 도구를 이용하여 수행한다. 설계단계에서 수행되는 주요 체크리스트는 다음과 같다.

① 일반적인 측면

- 기능 설명서에서 요구하는 기능을 만족시키기에 적합한 블럭 구조로 형성 되었는가?
- 기능적 분할이 실현 단위(화일, 실행모듈, 프로세스 등)위에 잘 할당 되었는가?
- 실현단위들 사이의 인터페이스 레벨은 비슷한가?
- 설계 지침에 따라서 설계작업이 이루어 졌는가?
- 원활한 설계작업을 위해서 필요한 문서들이 충분히 준비되어 있는가?
- 준비된 문서들이 전적으로 설계하기에 적당하

며 서술내용이 기능적으로 올바른가?
- 설계작업의 원활한 지원을 위하여 지원도구들은 충분한가?
- 기능시험 절차서에서 요구하고 있는 내용을 충분히 이해하고 있는가?

② 기술적인 측면
- 각종 설계지침의 적용에 충분한 자신이 있는가?
- 교환기 기본기능 및 설계에 참여하고자 하는 부분의 (호처리, 유지보수, 운용관리, ISDN, packet 등) 기능과 구조에 대한 지식이 준비되었는가?
- 기존에 설계되어 기술된 문서를 충분히 검토하였는가?
- 시스팀 메시지 종류에는 어떠한 것들이 있는가?
- 코멘트 설명서(COD) 및 출력 설명서(POD)가 운용자 지침서의 내역을 잘 반영하고 있는가?
- 설계하고자 하는 블럭의 기능은 다루기에 쉬운가?

③ 도구 및 메뉴얼
- 각종 준비된 메뉴얼이 준비되었고 준비된 메뉴얼을 활용하기에 충분한가?
- DBMS 사용자 메뉴얼을 숙지하고 있는가?
- 입출력 메시지 인터페이스 메뉴얼은 준비 되었으며 그 사용법은 충분히 숙지하고 있는가?
- 프로그램 표준화 적용과 검증방법 및 도구의 사용법은 어떠한가?
- 설계도구의 사용법에 익숙한가?
- 도큐먼트 자동생성 도구의 사용법은 숙지 하였는가?
- 각종 DB 검색방법 및 도구에 대한 지식은 어떠한가?

④ 프로그램 구조
- SDL 적용지침에 따라서 블럭구조가 설계 되었는가?
- SDL 다이어그램은 프로세스 또는 state oriented된 것인가?
- State 변수와 그 값들은 전체적으로 유일한가?
- State 변수의 양을 숙지하고 그 내용을 state 변수별로 간단하게 설명할 수 있는가?
- 비정상적인 흐름을 제어할 패스를 갖고 있는가?

⑤ 데이타 구조
- DB에 등록될 데이타는 적당한가?
- 데이타 구조는 비정상적인 경우에도 올바르게 대처할 수 있는 구조로 설계 되었는가?
- DB relation의 접근권리는 합당한가?
- 데이타 구조는 너무 복잡하지 않은가?
- 공통으로 사용 가능한 데이타는 없는가?

⑥ 읽기 구조
- 블럭 제어 흐름이 너무 자세하게 기술되지 않았는가?
- 블럭 설계를 위한 SDL 다이어그램이 survey하기 쉽도록 레벨을 갖고 그려졌는가?
- 정의되어 사용되는 단어들이 사용하기에 용이하도록 일관성이 부여되어 있는가?
- State 변수들의 이름은 그 자체로도 기능설명이 가능하도록 작명 되었는가?
- 기술문서들이 이해하기 쉽고 읽기에 쉽게 작성 되었는가?

⑦ 알고리즘
- 기존의 어떤 알고리즘으로 부터 유추되어야 할 알고리즘이 필요한가?
- 존재하고 있는 알고리즘이 수정 및 변경 되어야 하는가?
- 새로운 알고리즘이 요구되는가?

(4) 코드 검사

코드검사는 블럭을 구현한 소스 프로그램이 프로그래밍 표준화 지침에 의거하여 정확히 코딩 되었는지의 여부를 조사, 분석하여 조기에 에러나 문제점을 발견, 시정하므로 품질을 향상시키기 위함이다.

코드검사에서는 프로그래밍 언어로 실제 작성된 코드가 앞단계에서 제시한 설계기준을 정확히 실현하고 있는지 검증하기 위하여 소프트웨어 설계문서 및 설계지침서와 일치 여부를 확인하고 설계 알고리즘이 코드에서 정확히 구현되고 있는가 확인하며 이를 위해 소프트웨어설계 검토회의 또는 코드검사 전담팀을 구성한다. 전담팀은 중계자, 설계자, 실현자, 시험자로 구성되는데 중계자는 유능한 programer로 (프로그램 작성자일 필요는 없음) 자료준비, 계획, 검사모임 주관, 발견된 에러의 기록, 에러 개정여부의 조치를, 설계자는 프로그램 설계에 대한 임무를, 실현자는 소프트웨어 설계 변환을 통한 코딩작업을, 시험자는 test case 작성 및 시험을 수행한다. 코드검사는 에러 교정 보다는 에러 발견에 중점을 두고 실시되며 이의 체크리스트는 표2와 같다.

표 2. 코드검사 체크리스트

코드검사 체크리스트
1. 본 체크리스트는 몇번째 체크리스트인가?___번째
2. 코드검사의 라인 수___line, 수정된 라인 수___line, 계___line
3. 사용된 language:_____
4. 코드검사를 위한 준비 소요시간___시간
5. 코드검사에 소요된 시간___시간
6. 전체 코드검사에 소요된 person-hours: _____ preson-hours
7. 에러수정에 소요된 인원 및 시간:___인 ___시간
8. 에러 타입
logic 에러 _____개 syntax 에러 _____개
내부 interface 에러___개 외부 interface 에러___개
시스템 resource 에러_개 코딩 기준을 범한 에러_개
comment 에러 _____개 이전 단계에서의 에러_개
totals _____개
9. 코드검사율(시간당 조사된 라인수)_____lines/hour

2) 시험에 의한 방법

소프트웨어 시험은 버그를 찾아내는 것 외에 현재의 조건과 예측조건 사이의 차이를 확인하고 시험이 완료시 소프트웨어 결함에 따라 미리 예측되는 위험을 판단하고 시험중 발견된 결함을 분류, 종합하기 위한 것이다. 먼저 시험 대상을 선정하고 특정 기법을 이용하여 시험을 수행하게 되는데 주로 설계사양, 데이타 구조, 입력에 대한 출력의 관계를 체크한다.

시험은 그 평가관점에 따라 기능시험과 구조시험으로, 평가기법에 따라 정적시험과 동적시험으로 나눈다. 기능시험은 소프트웨어가 요구사항에 정의된 기능을 제대로 수행하는지 시험하게 되고, 구조시험에서는 소프트웨어 구조가 정확하고 타당한지를 시험하게 된다. 한편, 동적시험은 소프트웨어를 실제로 수행시키면서 그 기능 및 구조를 평가하고 정적시험은 수행을 동반하지 않는데 실제 소프트웨어를 시험평가할 때는 시험을 독립적으로 수행하지 않고 기능/구조 시험과 정적/동적 시험의 각 쌍에서 하나씩 선택하여 적절히 결합하여 수행하게 된다.

(1) 기능시험과 구조시험

이는 소프트웨어가 사용자 요구기능을 제대로 수행하는지 시험하는 것으로 주로 코딩이 완료된 후 프로그램의 수행기능을 시험하며, 시스팀의 수행기능과 입출력 데이타를 제공하고 그 처리결과를 분석하는 방법으로 진행된다. 기능시험에서는 프로그램 내부구조의 적절성 등을 고려하지 않으며 외부적으로 드러나는 기능의 정확성을 주로 시험하게 되므로 이를 블랙박스 시험이라고도 한다. 기능시험을 통해서는 주로 부정확한 기능, 누락된 기능, 인터페이스 오류, 성능상 오류, 초기화 또는 종료시 발생되는 오류, 자료구조에서 발생되는 오류 등을 발견하며 소프트웨어 요구사항에 명시되지 않은 기능에 대해서는 시험이 곤란하다. 기능 시험은 소프트웨어 개발단계중 단위시험, 통합시험, 시스팀 시험 등에 모두 적용되며 통합시험 및 시스팀 시험에서 큰 역할을 한다.

구조시험은 프로그램의 내부 수행경로와 자료구조의 적절성을 시험하는 것으로 소프트웨어 내부동작을 알고 있을 경우 내부 성분들이 적절히 수행되는가를 시험한다. 프로그램 논리나 코드구조를 시험하기 위해서는 프로그램안의 문장, 조건문, 경로의 특정 집합을 시험할 수 있는 시험사례를 입력시키고 그 수행결과가 올바른 경로를 통과하는지 확인하게 된다. 프로그램 구조를 완전하게 시험하기 위해서는 모든 경로를 수행할 수 있는 시험사례를 선택하여야 하나 프로그램에 따라 무한히 많은 경로를 포함하는 경우가 발생하므로 실제로 구조시험을 할 때에는 경제적인 제한사항을 고려하여 시험의 정확도를 높일 수 있는 범위를 결정하여 적용하게 된다. 이런 구조시험은 주로 단위시험에서 모듈 내부의 설계 및 구현 내용의 적절성을 시험하는데 적용되며 통합시험이나 시스팀 시험단계에서는 소극적으로 적용한다.

(2) 정적시험과 동적시험

정적시험은 프로그램을 직접 수행하지 않고 정적인 상태에서 프로그램의 정확성, 구조의 건전성을 시험하는 것이다. 원시 프로그램에 대한 정적시험에서는 코딩표준의 준수 여부 검출, 의심스러운 코딩 검출, 변수의 선언 및 엑세스상의 변칙, 부 프로그램 호출시 매개변수의 불일치 등과 같은 프로그램 구조상의 오류를 주로 시험한다.

한편, 동적시험은 시험사례를 입력하여 프로그램을 수행시키면서 프로그램의 동적인 수행상태나 결과를 분석하는 것으로 프로그래밍 이후 단계에서 주로 적용된다. 정적시험에서는 코딩 세그먼트에서 벗어난 데이터 흐름이나 제어 흐름상의 변칙 등과 같이 동적시험으로는 알 수 없는 문제점들을 발견할 수 있으며 프로그램내의 가능한 수행경로와 해당 경로를 수행하기 위한 경로조건을 알 수 있다. 반면에 동적시험은 모듈뿐 아니라 전체 시스팀을 통합한 뒤 그 구조 및 기능에 대한 시험 수행시 적용된다.

5. 검증

소프트웨어 검증은 소프트웨어 개발순기 각 단계에서 산출되는 소프트웨어 제품이 바로 전 단계에서 정의한 요구조건의 만족 여부를 검증하는 활동으로 소프트웨어 제품들이 요구사항, 설계지침, 표준화와 일치 여부를 분석, 검토, 검사, 체크, 확인하는 것이다.

1) 요구사항 검토

소프트웨어 설계전에 소프트웨어 표준화 및 설계지침 작성, 설계지침 검토회의, 이의 수정 및 배포 활동을 수행한다.

- 설계지침 : 작성 소프트웨어 설계를 위하여 교환기 구조 및 특성, 인터페이스, 사용자 요구사항 등을 고려한 소프트웨어 상위문서로 소프트웨어 설계지침, 프로그래밍 표준화 및 각종 사용자 메뉴얼을 작성한다.
- 검토회의 : 작성된 문서가 소프트웨어 요구사항으로서 교환기 구조, 소프트웨어 특성등이 잘 고려되었는지 확인하고 타 사용자 요구사항의 조건과 모순이 없는지 검토회의를 통하여 수정 및 보완 사항을 통보한다.
- 수정 및 배포 : 검토회의 결과에 따른 수정 작업을 수행하고 설계지침서, 표준화 지침서, 사용자 메뉴얼을 소프트웨어 설계자에게 배포하며 필요시 교육을 실시한다.

2) 설계 검토

요구사항에서 정의된 사용자 기능은 시스팀 관점에서 function으로 재 정의되며, 각 function은 동작흐름과 상호작용을 정의하는데 이때 실제 구현의 기본 단위인 블럭으로 구체화시키기 위한 소프트웨어 설계문서가 작성된다.

설계검토는 소프트웨어 설계사항이 사용자 요구사항, 설계지침, 표준화 지침 등을 만족하는지 확인하며 기술된 각종 기술문서들이 규정된 목차와 내용을 명시하고 있는지 확인하는 것이다. 설계 검토회의에서는 소프트웨어 설계 책임자 또는 담당자가 작성한 설계문서를 검토하며 소프트웨어 설계를 위해 채택한 기술, 가능한 에러, 각종 지침서 위반에 대한 검토내용을 지적하고 이를 문서화하여 설계에 반영토록 한다. 따라서 검토회의에서는 소프트웨어 설계와 관련한 정의, 제어흐름, 상호작용을 서로 토의하여 설계상 잘못되었거나 미비된 점을 보완하고 조기에 에러를 검출하며 블럭간 상호작용을 조정한다.

3) 감사

이는 설계에서 정의한대로 소프트웨어가 구현되었는지 여부를 확인하고 설계, 실현, 시험 단계에서 기술해야할 모든 관련 기술문서의 작성여부, 소프트웨어 형상관리에 대한 제반절차의 제시여부, 요구사항 정의 및 이의 실현방안에 대한 확인, 공식 및 비공식시험에 대한 확인 활동을 말한다. 감사는 소프트웨어 개발의 각 단계에서 수립된 절차에 따라 수행되며 이들 절차는 요구사항, 프로그램 설계 지침서, 코드 검사, 시험 절차서 및 시험 수행을 기반으로 이루워지며 이들 결과는 해당 부서로 통보되며 필요시 시정조치를 요구하여 이들을 확인한다. 검증에서 사용되는 체크리스트는 표 3과 같다.

표 3. 검증 체크리스트

검증 체크리스트

1. 검증단계 : 구조 ____규격 ____디자인 ____테스트 ____기타_____
2. 검증횟수_____번째
3. 전체 document 페이지 수 _____ 페이지
4. 검증하는 document 페이지 수 _____페이지
5. 검증자 수 및 검증자의 전체경력____명 ____years
6. 검증을 위해 소요된 준비 시간 _____ person-hours
7. 검증에 소요된 전체 시간_____ person-hours
8. 검증결과표
   minor 에러 수_____개 major 에러수_____개
   계 _____개

6. 문제점 보고 및 시정조치

소프트웨어 개발 또는 운용과정에서 발생한 문제점을 효율적으로 관리하고, 체계적인 절차를 세워 문제점을 보고하며 이를 시정, 조치, 확인하는 것이 필요하다.

1) 문제점 보고

소프트웨어 문제점이 발생한 경우 보고자는 문제점의 현상, 문제점이 시스팀에 미치는 영향, 문제점 발생 대상(하드웨어, 소프트웨어, 기술문서)과 그 위치, 현상태를 프로그램 변경위원회로 보고하는데 그 보고서는 표 4와 같다.

2) 시정 조치

개발 담당자는 소프트웨어 변경을 위한 적용대상 시스팀, 처리일정, 개발 담당자명, 변경 요구사항, 해결방안, 변경에 따른 관련부서 통보사항, 변경대상 기술문서 등을 기술한 변경요구서(change request)를 제출한다. 개발 책임자가 변경요구서와 최초 수정요구서를 프로그램 변경위원회로 제출하면 프로그램 변

표 4. 문제점 보고서

문제점 보고서

1. 문제 발견자 및 발견한 시간 : 발견자 _____ 발견시간 _____
2. 문제 수정자 및 수정한 시간 : 수정자 _____ 수정시간 _____
3. 문제를 수정하는데 소요된 person-hours _____ person-hours
4. 문제점 내용

_____

_____

5. 문제를 수정하기 위한 절차

_____

_____

경위원회는 변경관리 시스팀에 의해 변경요구에 대한 번호 (change request number)를 부여하고 변경관리 시스팀은 최초 수정요구서와 변경요구에 대한 각종 이력을 관리하며 소프트웨어 변경 대상이 되는 공식 소스화일을 관리한다.

3) 확인

변경된 소스화일은 generic issue를 위해 소프트웨어 product로 만들어지며 개발자는 변경이 끝난 소스화일에 대해 기능확인 시험을 수행한다. 개발자 자체 시험이 끝난 소스화일은 최초 수정요구서와 변경요구서의 변경에 따른 공식시험을 시스팀 시험 담당자에게 요청한다. 시험전담팀에 의한 공식시험을 통해 변경에 대한 확인이 이루워지면 시험전담팀은 이 결과를 프로그램 변경위원회에 통보하고 프로그램 변경위원회는 해당 소스화일에 대해 공식승인을 하며 최초 수정보고서와 변경요구서를 종료시킨다.

Ⅲ. 신뢰도 보증

그동안 교환기 소프트웨어의 신뢰도는 소프트웨어의 복잡성, 관련 이론의 미정립, 관리체계의 다원화, 데이타 수집의 난해 및 그 적용 tool의 미확립으로 체계화되어 계산되지 못하고 있었다. 시스팀 개발 단계별로 정확한 소프트웨어 데이타가 수집되어야 신뢰도 계산이 용이하며 특히 실제 운용중 발생한 고장 데이타가 정의된 고장 기준대로 측정되지 않았다면 정확한 신뢰도 계산은 기대할 수 없을 것이다.

소프트웨어 프로그램의 에러 분포는 매우 복잡하여 프로그램내 어느 위치에서 에러가 발생할지 알 수 없으며, 프로그램 수행시 어떤 입력 state가 입력되어 어떤 에러 코드가 수행될지 알 수 없어 우리는 한 고장이 발생한 후 다음고장이 언제 발생할지 알 수 없다. 즉, 소프트웨어 고장 발생은 random하여 프로그램 수행 중 언제 고장이 발생할 지 알 수 없고 단지 주어진 기간동안 다음 고장이 발생할 평균 시간과 고장의 발생 확률만을 예측하여 이를 기반으로 신뢰도 모델을 정립하고 신뢰도가 계산된다. 최근에는 여러 신뢰도 모형이 개발되어 소프트웨어 신뢰도를 예측하고 예측된 신뢰도를 실측치와 비교, 분석하는 활동을 활발히 수행하고 있다. 특히, 최근 소프트웨어 신뢰도 연구는 다음 사항을 주요 과제로 다루고 있다.

- 소프트웨어의 고장발생 추이는 어떠한가?
- 발생된 고장 데이터를 기반으로 신뢰도를 예측할때 어떤 모델이 가장 적절한가?
- 시스팀 리리즈를 위한 목표 신뢰도를 어떻게 정할 것인가?
- 시스팀 리리즈를 위해서 요구되는 시험기간은 몇일(달)인가?

1. 신뢰도 정의

소프트웨어 신뢰도는 주어진 환경하에서 주어진 시간동안 소프트웨어가 고장없이 요구사항대로 운용될 확률을 말한다. 여기서 언급된 고장, 시간, 환경 용어에 대하여 살펴보면

- 고장 : 고장은 규정된 규격의 요구사항대로 운용되지 못함을 의미하며, 여기서 요구사항은 주어진 입력 states에 응답을 요하는 기능 sets를 말한다. 고장은 크게 fault와 failure로 구분되는데 failure는 main 프로그램의 수행 결과 결함을 의미하여 fault는 특정조건 하에서 프로그램 수행시 결함사항으로 failure를 유발할 수 있는 bug를 의미한다.
- 시간 : 시간은 CPU time과 calendar time으로 나누는데 CPU time은 프로그램을 수행시 프로세서가 실제 소요한 시간이며 calender time은 시스팀 시험 등을 일, 월값으로 나타낸 것이다. Calender time은 프로그램 수행의 변화 usage를 고려치 않으므로 소프트웨어 신뢰도 계산시 CPU time이 calendar time보다 더 정확함을 알 수 있다.
- 환경 : 환경은 운용 profile로 표현하는데 운용 profile은 각 입력 states (command, option, argument 등)에 대한 프로그램 수행 확률로 나타낸다.

2. 기본 가정

소프트웨어 고장은 매우 다양하여 모든 경우를 반영한다면 일정한 모형화도 불가능하고 따라서 신뢰도 계산도 할 수 없게 된다. 따라서 다음과 같은 가정하에 고장 현상이 발견되도 신뢰도 모델이 정립된다.

- 소프트웨어 고장간 시간은 상호 독립적이므로 한 고장이 다른 고장의 발생에 영향을 주지 않는다.
- 개발중 발견된 고장은 즉시 수정되며 리리즈후에 발견된 고장은 다음 버젼의 리리즈 전에 수정된다.
- 고장 수리 동안은 새로운 고장은 삽입되지 않는다.
- 고장율은 남은 고장수에 비례하고 시험 시간의 함수이다.

3. 신뢰도 모델

그동안 소프트웨어 신뢰도 계산을 위해 직접, 간접적인 많은 이론 및 모델이 개발되어 왔다. 직접적인 예를 보면, 1967년 Hudson은 고장 발생을 birth로, 고장 수정을 death로 보고 고장 발견율이 남은 에러에 비례한다고 보았고, 1973년 Schick and Wolverton은 고장 발생율이 남은 에러 및 시간의 곱에 비례한다고 보았으며, 1975년 Schneidewind는 시간당 에러 발견은 평균이 exponential 분포를 하는 nonhomogeneous Poisson process에 따른다고 보고, 1975년 Musa는 소프트웨어 고장 측정은 calendar time보다 CPU time으로 해야 정확성을 갖을 수 있다고 주장했고, 1979년 Goal and Okumoto는 소프트웨어 디버깅을 Markov process로 보아 각 state를 정하고(예 : 첫 state는 디버깅 전, 둘째 state는 디버깅 후, …, 마지막 state는 고장이 없는 상태), state간에는 적절한 확률 천이가 있고 고장 발견은 exponential로 감소하는 nonhomogeneous Poisson process에 따른다고 보았다. 한편, 간접적인 방법으로 소프트웨어 구조를 분석하여 신뢰도를 예측하는 방법이 있는데 이는 소프트웨어의 복잡도를 기반으로 하는 방법이다.

1) 간접적인 신뢰도 모델

복잡도에 의한 신뢰도 예측은 소프트웨어의 위상구조, 프로그램의 어휘를 바탕으로 하며 이는 프로그램의 flow chart 또는 코딩이 완료시 복잡도를 측정하므로 이루워진다. 복잡도의 측정기준은 보는 관점에 따라 여러가지가 있는데 보통

- 프로그램을 이해할 수 있는 정도
- 프로그램을 다른 사람에게 설명할 수 있는 정도
- 프로그램을 변화시킬 수 있는 정도
- 프로그램의 오류를 수정하고 유지할 수 있는 정도
- 프로그램을 작성하는데 소요되는 노력의 정도

를 이용하여 측정하며 소프트웨어의 복잡도는 주로 프로그램의 크기, 제어흐름 및 데이타의 구조에 기반을 둔다. 이중 가장 간편하고 폭넓게 이용할 수 있는 방법으로 전자교환기 같은 대형 프로그램에서 주로 이용되는 프로그램 크기는 다음 항목에 의해 측정된다.

- 코멘트, 입출력 format, 실행문, 컴파일러 제어 등을 포함한 모든 소스 라인수
- 서브루틴, 항수 등의 프로그램 모듈 수
- 프로그램 모듈의 평균 길이
- 코멘트문을 제외한 모든 소스라인 수
- 할당, 제어, 입출력, 산술문 등의 실행문 수

2) 직접적인 신뢰도 모델

직접적인 모델은 현재의 신뢰도 값을 기반으로 리리즈를 위한 목표 신뢰도를 정하고 얼마나 더 시험해야 시스팀을 리리즈 할 수 있는지 결정하는 모델로서 시험기간에 발생한 고장자료를 바탕으로 몇가지 가정을 세우고 적정 신뢰도 모델을 개발 또는 선정하여 신뢰도를 예측하게 된다. 이들 모델은 고장간 시간 모델, 고장 수 모델, 고장 삽입 모델 및 입력영역 중심 모델로 대분할 수 있다.

(1) 고장간 시간 모델

이 모델은 소프트웨어 고장시간 간격이 지수 분포를 따르고, 고장 및 고장간 시간은 서로 독립이라는 가정하에 고장시간 발생을 예측하는 모델이다. 즉, 연속되는 고장간 시간은 고장이 제거됨에 따라 증가하여 에러가 검출, 제거됨에 따라 남은 고장이 감소되고 신뢰도가 향상되는 형태의 모델이다.

(2) 고장수 형 모델

이 모델은 시험기간 중에 발생 및 제거된 고장 수가 증가할 수록 향후 검출될 고장수가 줄어들 것이라는 이론하에 Poisson 분포를 기반으로 고장 발생 패턴에 따라 여러 모델중 하나를 선정하여 신뢰도를 예측한다. 특히, 교환기의 경우 고장 데이타는 주로 calender time으로 측정되므로 정확한 신뢰도를 구하기 위해 CPU time으로 변환을 위한 여러 방안이 연구되고 있다.

(3) 고장 삽입 모델

소프트웨어 내의 고의로 고장을 삽입후 시험을 실

시하여 삽입한 고장의 발견수와 전체 고장 발견수의 비율을 이용하여 신뢰도를 계산하는 방법이다. 이 모델은 프로그램에 삽입된 고장과 원래 프로그램에 존재하는 고장의 발견 가능성이 동일하다고 보고 일정 시간동안 발견된 고장이 삽입된 고장인지 원래 에러인지를 결정하여 비율을 결정한 후 프로그램내 고유 에러수를 예측하게 된다.

(4) 입력영역 중심 모델

소프트웨어를 하나의 함수로보고 입력 자료중 고장 입력이 출력 치역으로 사상되는 비율을 이용하여 신뢰도를 측정하는 방법이다. 모든 입력자료를 시험할 수 없어 선택기법에 의해 선택자료를 이용하므로 신뢰도 정확성은 선택기법에 따라 크게 다를수 있으며 신뢰도는 단순화된 test case를 실제로 수행중 관측된 고장수를 기초로 계산된다.

3) 신뢰도 모델 적용

앞에서 언급된 신뢰도 모델을 실제 적용시에는 많은 어려움이 예상되므로 시험결과에 의한 신뢰도 모델을 개발하기 전 분명한 가정을 언급해야 하고 모델을 개발하여 적용할 때도 모델이 갖고있는 제한성을 잘 파악해야 한다. 소프트웨어는 개개의 개발환경에 의존하므로 일률적으로 적용할 모델은 없으며 더욱이 같은 환경하에서 개발된 소프트웨어 제품도 여러번 시험과정을 거치면 처음 세운 가정이 나중에는 불합리한 경우가 생기게 된다. 전자교환 시스팀의 소프트웨어 시험을 유니트 시험, 통합시험, 기능시험, 시스팀 시험 단계로 나눌때 각 시험 단계에서 적용할 가능한 모델을 살펴보면 다음과 같다.

(1) 유니트 시험(unit test)

유니트 시험은 소프트웨어 설계의 가장 작은 단위인 모듈에 대한 시험으로 이 단계에서 생성되는 test case는 실제 운용환경과는 다른 모듈 입력 영역으로부터 생성되는 것이 대부분이다. 또한 test case가 random하게 취해지지 않고 미리 정해진 결정적인 형태로 취해지므로 이 단계에서는 고장삽입 모델을 사용하는 것이 좋다.

(2) 통합시험(integration test)

통합시험은 유니트 시험이 끝난 모듈들을 설계된 구조로 통합하면서 각 모듈간 관계되는 인터페이스를 시험하는 단계로서 이 단계에서 생성되는 test case들은 실제 운용환경과 유사한 시험환경을 조성하여 random하게 취해질 수도 있고 합리적인 시험전략에 의해 미리 정해진 형태를 갖을 수도 있다. Test case가 random하게 취해질 경우엔 상기 모든 모델이 적용 가능한데 만일 입력 영역 중심 모델을 사용하면 실제 사용형태를 잘 반영할 수 있는 test case가 좋으며 결함삽입 모델도 삽입된 결함이나 고유의 결함들이 발견될 확률이 같다는 가정에 위배되지 않으므로 사용 가능하다. 고장간 모델이나 고장수 모델도 시험이 랜덤하게 수행된다면 모두 적용 가능하고 test case들이 random하지 않을 경우엔 고장수 모델이 최적의 모델이 될 것이다.

(3) 기능 시험(function test)

기능시험은 통합시험에서 검출된 결함이 수정된 후 패키지화된 이후 적용하는데 여기서는 소프트웨어 기능들이 사용자의 적절한 기대치에 부합하는가를 확인하게 되는데 적절한 기대치는 소프트웨어 요구사항 명세서에 정의되어 있다. 검출된 고장이 즉시 수정되기 어려우므로 고장간 시간 모델은 적용하기 어렵고 고장수 모델이나 입력영역 중심 모델이 적용 가능하다.

(4) 시스팀 시험(system test)

시스팀 시험은 소프트웨어, 하드웨어 및 관련 정보를 합친 최종 산물을 점검하는 시험으로 기능시험과 유사하게 고장수 모델이나 입력영역 중심 모델이 가능하다.

(5) 운용 단계

이 단계에서는 소프트웨어 기능이나 경로들이 사용자들에 의해 이용되어지는 형태가 비슷하게 반복되므로 입력이 random하지 않고 검출되는 결함이 즉시 수정되기 쉽지 않으므로 고장수 모델이 가장 적합하다.

Ⅳ. 결 언

최근 소프트웨어 품질의 정확한 측정을 위해 비용 평가모델, 생산성 측정 모델, 품질 척도 모델, 신뢰도 모델 등이 연구 개발되고 있는데 본문에서는 전자교환 시스팀의 소프트웨어 품질을 정량적인 방법에 의해 평가할 수 있는 방법을 위주로 기술하였다. 성공적인 소프트웨어 품질목표 달성은 규정된 체크리스트에 의해 각 체크 포인트마다 정량적으로 개발단계를 조사하고 조사 데이타를 이용하여 정확한 모델을 개발하며 신뢰도를 예측하는 활동을 지속하므로 가능하다.

전자교환 시스팀의 소프트웨어 규모가 다양화됨에

따라 이를 효과적으로 관리하고 평가하기 위해서는 개발 최종 product는 물론 개발 전 단계를 수시로 검토 조정하고 보다 양질의 소프트웨어를 생산하기 위한 활동을 지속해야 하는데 본문에서 언급된 소프트웨어 평가방법중 가능한 항목부터 차례로 적용하는 것이 바람직하다.

또한, 아무리 좋은 모델과 이론이 개발되고 품질향상을 위한 제도가 훌륭하더라도 소프트웨어의 통계적 데이타가 확보되지 못했거나 수집된 데이타가 실제 상황과 다르다면 연구된 모델과 이론의 진가를 발휘할 수 없을 것이다.

參 考 文 献

[1] TDX 개발단, TDX-10 품질보증계획서(안), 1988년 11월

[2] TDX 개발단, TDX-10 개발 일반 지침서, 1990년 10월

[3] TDX 개발단, TDX-10 소프트웨어 설계 지침서, 1990년 10월

[4] 송인근, 소프트웨어 품질평가 방안, 전자통신연구소, 1990년 1월

[5] 이재형, 이영곤, 한상학, 소프트웨어 시험평가에 관한 고찰, 한국통신 품질보증, 제2권 3호, 1989년 11월

[6] 이규욱, 한기철, 전자교환기의 소프트웨어 신뢰도 예측, ETRI Journal, 12권 3호, 1990년 10월

[7] 이규욱, 5ESS 소프트웨어 품질보증 활동, 한국통신 품질보증, 제3권 3호, 1990년 10월

[8] Kyu Ouk Lee, *Reliability Evaluation and Prediction of a Time Division Switching System,* Northern Illinois University, May 1990.

[9] John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability Measurement, Prediction, Application,* AT&T Bell Labs., 1987.

[10] AT&T Bell Labs., 5ESS Switch Software Quality Assurance Plan, June 1988.

[11] Roger B. Ackerman, Roberta J. Coleman, Elias Leger, John C. MacDorman, *Process Quality Management & Improvement Guidelines,* AT&T, Dec. 1987.

[12] D.A. Christenson, S.T. Huang, *Code Inspection Management Using Statistical Control Limits,* AT&T Bell Labs.,1988.

[13] D.A. Christenson, *An In-Process Fault Density Metrics Model, AT&T* Bell Labs., 1988.

[14] Bell Communications Research, *Reliability and Quality Switching Systems Generic Requirements,* Bellcore, Oct. 1990.

[15] Bell Communications Research, *Reliability Prediction Procedure for Electronic Equipment,* Bellcore, Sep. 1990.

[16] Bell Communications Research, *Manufacturing Program Analysis for Quality and Reliability,* Bellcore, Aug. 1987.

筆 者 紹 介

李 圭 旭

1952年 10月 11日生

1980年 2月 한국 항공대학 전자공학과(학사)

1990年 3月 미국 노던 일리노이 대학 산업공학과(석사)

1979年 12月～현재 한국전자통신연구소 품질보증실 선임연구원