

Software Project from A Testing Perspective

Author: Lakshmi.K.Kumar

(번역: techbard, techbard@TERAMAIL.com)

Introduction

이 페이퍼는 QA/테스팅 조직의 관점에서 어떤 소프트웨어 프로젝트/프로덕트 프로그램을 특징짓는 속성들을 논의한다. 3 가지 중요한 프로파일이 논의되는데, 이 3 가지 프로파일은 라이프싸이클 내에서의 3 가지 단계에서 프로젝트/프로덕트 프로그램을 특징짓는다.

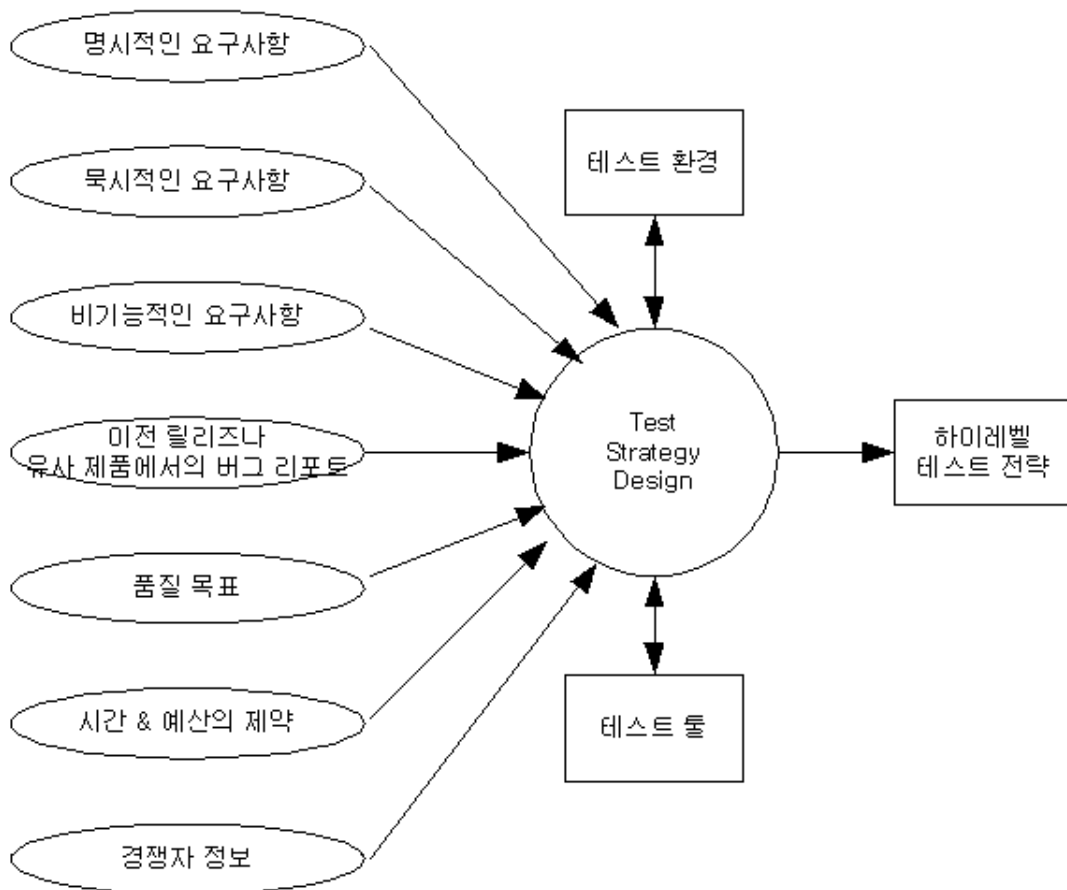
- 사전 계획 테스트 프로파일 - 프로젝트 계획 단계에서
- 테스트 실행 프로파일 - 프로젝트 실행단계 동안
- 릴리즈 테스트 프로파일 - 프로젝트 종료/프로덕트 릴리즈 시점에서

Planned Test Profile of a Software Project

사전 계획 테스트 프로파일은 프로젝트 계획 /범위설정(scoping) 단계에서 다소 높은 수준의 기술-관리적인 결정의 결과로 만들어진다. 프로젝트 계획 단계에서 만들어져야 하는 중요한 결정들은 다음과 같다.

- 테스트 예산의 할당 또는 테스트 비용/활동(effort) 추정
- 이렇게 할당되거나 추정된 예산의 사용 계획
- 질문에 답하기 - 얼마나 테스트해야 하나? 또는 언제 테스트를 중단해야 하나?
- 완료해야 할 테스트의 유형
- 테스트 디자인/개발 방법론
- 테스트 환경 & 툴
- 위의 조건들 때문에 발생하는 제약
- 위험 & 완화 계획

이러한 결정들은 아래 그림에서 보여지는 요소들에 의해서 영향을 받을 것이다.



Quality v/s Time/Effort/Cost Balance for Testing Activities

질문은 다음과 같다.

전체 프로젝트 비용/활동(effort)의 몇 퍼센트가 테스트에 할당되어야 하는가?

이에 대한 대답은 아래의 몇몇과 관련이 있다.

- 프로덕트/프로젝트의 속성 (미션 크리티컬한가 아닌가)
- 프로젝트/프로덕트에서 수립한 내부적인 품질 목표
- 프로덕트의 인증 요구사항
- 프로덕트/시장의 고객들이 만든 "품질 약속"
- 프로덕트를 사용하면서 제공되는 서비스에 대한 "서비스 레벨 협약"
- 품질 목표/품질 약속/SLA를 달성하지 못한 경우 발생할 수 있는 "페널티 조항"

테스트 예산을 결정하는 한 가지 방법은 "품질 목표"를 고정하고, 그 다음에 이 목표를 달성하기 위한 비용/활동/시간/자원을 추정하는 것이다.

또 다른 방법은 비용/활동/시간/자원의 제약(이것이 테스트 예산이다)을 고정시킨 다음, 이러한 제약을 가지고 달성할 수 있는 품질 목표를 추정하는 것이다. 이것은 첫번째 반복(first iteration)에서는 쉽지 않다. 이전 반복에서의 메트릭 데이터를 기반으로 한 연속적인 반복 내에서만 가능하다.

위의 요소들은 아래의 축약된 내용으로 좁혀질 수 있다.

- 품질 목표
- 인증/규격 만족의 필요
- 테스트 목표
- 테스트에서의 QA의 부담

품질 목표, 테스트 목표에 대한 근거는 고정되어야 할 것이다.

Quality goals

품질 목표는 수량적으로 측정가능한 요소(parameter)여야 한다. 예를 들면:

- 최대 허용 잔존 결점 밀도 - 모든 테스트를 완료하고 남겨진 KLOC 당 결점수
- 최대 실패율(maximum failure rate) - 시스템을 사용하는 단위 시간 당 최대 가능한 실패 수 (시간은 캘린더 날짜나 CPU 타임이 될 수 있다)
- MBTF - Mean Time Between Failure (단위 시간 당 실패의 수)
- 신뢰성 - 에러 없이 동작할 가능성
- 가용성 - 전체 운용 시간중 시스템이 동작하는 시간의 비율 (역자주: 일정 기간동안의 시스템 운영. 예를 들어 어떤 시스템이 일주일 100시간 가동 한다면 그것의 가용성은 $100/168=60\%$ ($168=24 \times 7$))

Certification/conformance requirements

이러한 것들은 표준에 의해 정의된다. 이것에 의해 프로덕트의 기능/기능성 측면이 구체화된다. 이것 역시 테스트 방법론 결정, 사전 인증 준비, 인증 대행 기관/테스트 하우스 & 비용의 결정에 영향을 준다.

Testing goals

질문은 다음과 같다.

얼마나 테스트해야 하나? 또는 언제 테스트를 중단해야 하나?

이것은 매우 중요한 결정이고, 실제 테스트 수행 이전에 제시되기 어려운 것이다. 이것의 근거는 아래의 하나 또는 그 이상이 될 수 있다.

- 시간/활동/비용 예산이 모두 소모될 때까지 테스트한다. - 예산이 바닥날 때 테스트를 중지하고, 발견된 결점이나 수행된 테스트에 대한 고려는 하지 않는다.
- 사전에 정의된 테스트 케이스를 수행한다. - 이것이 가장 자주 발생하는 상황으로, 선택된 숫자나 테스트 집합은 대개 논리적인 근거를 가지고 있지 않다. 이 영역은 대단히 개선될 수 있는 영역이다. 이것은 "테스트 효율성"에서 상세하게 다뤄진다.
- 평균 실패율(failure rate)이 출발점 이하로 떨어질 때까지 테스트한다. - 미션 크리티컬 시스템의 경우에 이렇게 한다. 얼마나 많은 비용/시간/활동이 들어가더라도, 테스트는 "최대 허용 실패율" 품질 목표를 달성할 때까지 계속되어야 한다.
- 커버리지의 레벨을 달성한다. - 테스트 커버리지는 서로 다른 종류의 테스트의 상황에서 서로 다른 의미와 측정 단위를 가진다. (Ex: % KLOC, % function points, % numbered requirements) - 커버리지는 수량적으로 측정된 & 테스트가 커버리지 통과기준을 만족할 때까지 계속되어야 한다. 테스트 커버리지는 부록에서 더 자세히 다뤄진다.
- 사전 정의된 잔존 결점 밀도를 달성한다. - 사전에 추정해서, 전체 예상 결점 수 & 사전 정의된 비율의 결점이 발견될 때까지 테스트를 한다. (& 사전 정의된 결점이 시스템에 남아있지 않도록 하기 위해서 제거된다)

Dependency of product quality on testing (QA load on testing)

고려할 또 다른 요소는 테스트에서의 QA의 부하이다. 소프트웨어 개발 라이프사이클에서 결점의 제거가 발생하는 또 다른 메커니즘이 있다. 예를 들면, 요구사항 리뷰, 디자인 리뷰, 코드 인스펙션, 툴 지원 디자인 분석 & 확인, 시뮬레이션에 의한 검증 등이다.

이러한 메커니즘의 비효율성이나 부재는 각 단계의 버그 봉쇄(phase containment)를 약화시켜서, 테스트에서 QA의 부담을 강화시킨다.

QA 부담을 결정하는 한 가지 방법은 아래에 설명된다. 프로젝트의 "품질 목표"를 "잔존 결점 밀도 (KLOC 당 결점수)"로 표현하자.

"예상 결점 밀도" & "이 결점에 기여한 기원 (단계)"를 추정하는 기법이 있다. (Ex: COQUALMO 모델)

테스팅 시작 시점(대개는 시스템 테스팅)의 결점 밀도는 이전 단계의 메트릭 데이터(요구사항 리뷰, 디자인 리뷰, 코드 인스펙션 & 유닛 테스트)에 의해서 결정될 수 있다.

$$\begin{aligned} \text{Defect density at the start of system test} &= \\ \frac{(\text{Total number of defects estimated/ expected} - \text{total number of defects already} \\ &\quad \text{detected/ corrected in review \& inspection})}{(\text{Size of the software in KLOC})} \\ \text{QA load on testing} &= \\ \frac{\text{Defect_density at the start of system test} - \text{desired residual defect density} \\ &\quad \text{at product release}}{\text{Estimated initial defect density}} \end{aligned}$$

Distrubution of Testing Budget

전체 테스팅 예산이 확정된 이후에는, 다양한 테스팅의 요소에 이 예산의 배분하는 것을 결정해야 한다. 시스템의 모든 파트/요소/기능에 동일한 정도의 테스팅을 할당할 필요가 없다. 테스팅 예산의 배분 근거는 아래의 하나 또는 그 이상이 될 수 있다.

- 모듈/컴포넌트의 중요성 과/또는 심각성
- 기능성/기능/비즈니스 프로세스의 중요성 과/또는 심각성
- 시스템의 운용 프로파일
- 버그의 경향이 있던(buggy) 모듈/기능성에 대한 과거 테스트 데이터 (이전 버전의)
- 현재 릴리즈의 소프트웨어 개발 라이프사이클(SDLC)상 이전 단계의 품질 메트릭 (단계 버그 봉쇄 지표 - phase containment indicators)
- 모듈/서비스/비즈니스 프로세스의 FMEA(Failure Mode and Effects Analysis) 결과 (완료되었다면)

위의 요소에 대한 분석은 테스트 중인 시스템에 "품질에 치명적인" 측면으로 분석되어야 한다. 다양한 모듈/컴포넌트/서비스/기능/기능성/비즈니스 프로세스의 "중요성/심각성"이 결정된 이후에는, 이러한 각각의 "측정 가능한" 품질 & 테스트 목표가 달성되어야 한다.

다음 단계는 이러한 각각의 하위 목표를 달성하기 위한 요구사항(HW, SW, 훈련, 툴, 인원 & 활동)을 결정하는 것이다. 이것은 이러한 하위 목표 각각을 달성하기 위한 "비용"을 차례로 결정한다. 전체 테스트 예산은 반복적으로 & 공정하게 하부 활동들로 배분될 것이다.

테스트 예산의 배분은 최대의 ROI (return on investment)를 얻기 위해 노력해야 하는 아주 중요한 최적화 활동이다. SW/HW/툴 구매의 결정이 이루어지는 동안, 프로젝트의 범위를 넘어서서, 장기간의 ROI가 가능한 것인지도 고려해야 한다.

Different Types of tests that will be done

수행완료할 필요가 있는 테스트의 유형은 요구사항 & 품질 목표에 근거를 둔다. 예를 들어, 미션 크리티컬 시스템에서는 화이트 박스 테스트에 대한 강제가 권장될 수도 있다. 요구사항은 최대 잔존 결함 밀도 또는 %코드커버리지를 보장할 수도 있다. 또 웹 기반의 시스템은 성능 테스트가 중요할 수도 있다.

COTS(Commercial Off the Shelf) 기반 시스템에 대한 유닛 테스트는 내부 보조모듈에 의해 제한 받을 수 있다. OSS 기반의 COTS에서는 테스트 & 비운용적인 테스트에 근거한 운용적인 프로파일과의 공정한 혼합이 필요할 수도 있다.

이러한 결정은 프로젝트 요구사항에 매우 종속적이다. 따라서, 작업 팀의 재량으로 남겨진다.

Test Design/Development Methodology

이것은 아주 심각한 영향을 줄 수 있는 매우 중요한 결정이다. 여기서 훌륭한 결정이란 반복적인 시간을 줄이는 돌파구를 의미한다! (A good decision here might mean breakthrough cycle time reduction!)

이것의 본질은 "앞부분에서 자동화" (회귀 테스트만을 위한 것이 아닌)이다. 공식적인 요구사항/디자인 문서 또는 코드에서 "자동화된 테스트 생성"이 중요하다. (Ex: UML 모델 기반 테스트 디자인, SDL 모델에서의 자동화된 테스트 생성, 메시지 시퀀스 차트에서의 자동화된 테스트 생성, 유닛 테스트에 근거해 생성된 툴) 이 단계는 ROI 계산과 프로토타이핑 개념 검증이 포함될 수 있다. 복잡한 시스템의 경우 초기 투자 비용에도 불구하고, 이러한 접근법으로 놀라운 이득을 얻을 수 있다.

Test Environment(s) & Configuration (HW, SW)

이 활동은 다음의 일부를 포함한다.

- 요구사항 & 시스템 아키텍처에 근거한, 테스트 베드/테스트 환경의 확인
- 스텝/드라이버 시뮬레이터 등의 확인
- 환경 (하나 이상 존재한다면)의 평가
- 이러한 환경을 사용해서 테스트 할 수 있는 또는 할 수 없는 것의 확인
- 인위적으로 만든 환경이 실제 환경을 충분히 세밀하게 흉내내는지에 대한 검증
- 프로토타입 개념 검증 (옵션)

Test Tools (Type, capabilities & feature list)

테스팅의 유형, 테스트 개발 방법론 & 테스트 환경, 테스트 툴에 근거해서 선택될 필요가 있다.

여기서는 3 가지 결정 레벨로 결정되어야 한다.

- 테스트 툴의 유형: 이것은 프로젝트가 기반하고 있는 기술과 테스트의 유형과 관련이 있다.
 - ROI 평가: 툴 & 훈련 비용 vs 품질 & 생산성 개선이 평가된 다음, ROI 가 계산된다. (툴의 장기간의 사용도 고려되어야 한다)
 - 테스트 툴의 결정: 이것은 서로 다른 벤더의 유사한 툴의 비교 평가와 연구가 포함된다. 이러한 유형의 툴 중 테스트 도중에, 테스트 프로세스 이전에 사용할 수 있는 것들은 아래와 같다.
-
- 코드 인스펙션 툴
 - 유닛 테스트 툴
 - 기능 테스트 툴
 - 시스템 테스트 툴
 - 프로토콜 테스트 툴
 - 텔레콤 테스트 툴 (시뮬레이터/에뮬레이터/모니터)
 - 부하/스트레스 테스트 툴
 - 데이터베이스 테스트 툴
 - 웹 테스트 툴
 - 자바 테스트 툴

툴 선택 단계는 다수 벤더의 유사 툴에 대한 평가가 있어야 하며, ROI 분석 & 확인된 목적에 대한 "툴 도입 적합성" 또한 평가되어야 한다.

Execution Type Profile of a S/W Project

테스트 프로파일 실행은 테스트 효율성, 테스트 자동화, 테스트 관리 프로세스, 결점 추적, 메트릭 수집 등의 속성을 수행하는 것이다.

Test Efficiency

Efficiency of test logic

이것은 각 테스트와 선택된 테스트 집합의 효율성의 측정이다. 이것은 테스트 실행 효율성(Ex: 자동화) 또는 테스트 자원의 효율적인 사용(인프라스트럭처 & 사람)을 포함하지 않는다.

철저한 테스트는 훌륭한 솔루션이 아닌것으로 이해된다. 비용/활동/시간에 대한 제약은 철저하지 않은 테스트를 강제할 수 있다. 도전과제는 "올바르거나 최적인 테스트 부분조합"을 선정해서, 최대한의 결점을 찾아내는 것이다.

테스트 로직의 효율성은 아래의 몇몇 방식으로 측정할 수 있다.

- 테스트 케이스 당 평균 코드 커버리지 - (이것의 반대는 테스트 케이스의 집중도와 날카로움이다)
- 테스트 케이스 당 평균 상태 커버리지 (상태 기반 시스템의 경우)
- 테스트 완전성 메트릭 - 테스트 케이스의 결함(fault) 노출 능력 (mutation tests measure this)
- 해당 코드 커버리지가 요구되는 최소 테스트의 수
- 결함 노출이 요구되는 최소 테스트의 수

철저한 테스트는 아니지만 효과적인 테스트로 직교 배열 같은 (Tagucho methods) 기법이 사용될 수 있다. 이것은 시스템의 전체 입력 공간을 정의함으로써 가장 최적의 부분집합(주어진 크기로)을 선택하는 기법이다.

DMADV(Define, Measure, Analyze, Design, Verify) 기법은 테스트 케이스 디자인을 개선하는데 사용될 수 있다. 운용적인 프로파일 테스트 & 비운용적인 프로파일 테스트를 공정하게 혼합해서 하나의 기법을 독립적으로 사용할 때보다 더 많은 결점을 발견할 수 있다.

비운용적인 프로파일 테스트의 선정은 아래의 입력들 일부에 의해서 가이드될 것이다.

- 이전 릴리즈에서 수집된 메트릭의 분석 » 버그 있는 모듈/기능/비즈니스 프로세스의 유형 분류 » 이것에 대한 테스트 강화
- 현재 릴리즈의 앞단계 (디자인/코딩)에서 수집된 메트릭의 분석 » 이전 단계의 버그 봉쇄능력이 떨어져서 발현된 버그 있는 모듈/기능/비즈니스 프로세스의 확인 » 이것에 대한 테스트 강화
- 시스템에 대한 FMEA (failure mode effect analysis)이 완료된 경우 » 모듈/기능/비즈니스 프로세스에 대한 RPN (risk priority number) » 높은 RPN 에 대한 테스트 강화

Efficiency of test execution

테스트 실행 효율성은 자동화, 툴의 사용, 테스트 준비 시간의 단축, 테스트 준비 시간을 최소화하기 위한 테스트의 재 배치, 테스트 자원의 효율적인 활용(사람 & 인프라), 더 나은 테스트 프로세스(쉽고 유저 친화적인 테스트 관리, 결과 취합, 결점 관리 등)에 의해서 개선될 수 있다.

TQSS methods 는 테스트 실행의 효율성을 개선하는데 사용될 수 있다.

Test Management (process) Tools

테스트 관리툴의 사용이나 미사용은 테스트 프로젝트의 기간과 크기에 달려 있으며, 조직이 이미 테스트 관리 시스템을 가지고 있는지 그렇지 않은지에 달려있다.

소규모의 단기간의 프로젝트에서는 이미 시스템을 가지고 있고 사용을 주장하는 경우 이외에는 필요하지 않을 수도 있다. 하지만, 테스트 프로젝트의 모든 산출물들은 조직적인 방법으로 유지되어야 한다. 정형화된 템플릿, 가이드라인, 체크 리스트, 프로젝트 전반에 걸쳐 수집된 형식과 절차들에서 추출된 데이터 (metrics)는 매우 유용하다.

대규모의 장기간의 프로젝트에서는 상업적인 테스트 관리툴이 추천된다. Ex: Mercury International 의 Test Director 또는 Rational 의 Test Manager. 이것 역시 조직의 자산인 도구가 될 수 있다.

테스트 관리 툴은 다음의 기능을 제공한다.

- 테스트 팀 정의
- 테스트 작업 분배
- 테스트 계획
- 테스트 수트/계획/스크립트를 담는 제어가능한 저장소의 구성환경
- 자동 실행을 위한 테스트 스케줄링
- 테스트 결과 취합

- 테스트 리포트 생성
- 결점 추적
- 멀티 사이트 작업 (*역자주: 다수의 작업자가 지리적으로 떨어져 있어도 협업이 가능한 기능)

Test Automation

테스트 자동화이던 아니던, 회귀 테스트만을 위한 것이던, 앞부분에서 자동화이던 간에 ROI에 근거한 결정이 있어야 한다. 테스트 자동화의 결정이 내려진 이후에는 가능한 옵션중에서 선택하는 것이 중요하다. 이러한 것들에는 다음이 있을 수 있다.

- 셸 스크립트의 사용
- TCL의 사용 (* 역자주: TCL은 티클 언어를 말함)
- C/C++의 사용
- 테스트 하니스의 사용 (test harnesses)
- TTCN의 사용
- Jscript의 사용
- 레코드-리플레이 툴
- Rational, Parasoft, Mercury International의 상용툴의 사용

Test Suspension & Resumption Criterion

만일 테스트 중인 시스템이 매우 버그가 많아서 (buggy) 테스트를 더 진행할 수 없다면, 테스트 활동은 중단될 필요가 있다. 각각의 모듈/기능성/릴리즈/테스트 활동에 대해서 특정한 테스트 중단 & 재시작 통과기준이 이 단계에서 정의되어야 한다.

Metrics that need to be Gathered

테스팅의 다양한 단계와 유형이 진행되면서 수집되고 추정되어야 하는 다양한 메트릭이 여기서 정의된다. 이것들은 프로덕트 메트릭 & 프로세스 메트릭이 포함된다.

프로세스 메트릭은 실행 테스트 프로파일에 좀 더 다듬을 수 있는 피드백을 준다. 프로덕트 메트릭은 테스트 예산 분배, 테스트 케이스 디자인, 관심 영역 등을 재조정하는 피드백을 준다.

프로세스 메트릭 (예제 리스트)

- 테스트 스케줄 초과
- 테스트 활동 초과
- 이전 단계의 테스트에서의 단계 결점 봉쇄 (phase containment)
- 엔지니어당, 일별당 디자인된 테스트 케이스의 수 (수동/자동) (생산성)
- 엔지니어당, 일별당 수행된 테스트 케이스의 수 (수동/자동) (생산성)
- 테스트 베트의 점유 기간 (생산성)
- 버그 수정 후 되돌아 오는데 까지 걸리는 시간 (Bug fix turn over time)

프로덕트 메트릭 (예제 리스트)

- 발견된 전체 결점 수 & 코드당 심각도/모듈/기능/요구사항/비즈니스 프로세스에 의한 분류
- 성능 메트릭 (ex: 일정 트랜잭션에 대한 반응 시간)
- 부하/스트레스 테스트 메트릭 (ex: 특정 유형 또는 혼합 유형의 최대 처리 가능한 동시 트랜잭션 수)
- 최종 소프트웨어의 실패율 추정치
- 프로덕트에 잔존할 것으로 예상되는 결점 수
- 소프트웨어의 신뢰성 성숙도 추정

Release Test Profile of a SW Project

Analysis of Test Results - Testing Project Post-Mortem

주어진 시간 동안 수집된 개별 테스트의 결과 취합 후, 이러한 테스트 결과들은 분석되어야 한다. 이러한 분석은 통계적으로 엄격해야 하며, 장기간의 관측에서 경향이나 패턴을 드러내야 한다. 릴리즈와 릴리즈 사이에 수집되어야 하는 것들은 다음과 같다.

- PASS/FAIL 보다 더 종합적인 리포트 생성
- 어떤 분류에 따라 발견된 결점/에러의 분류
- 심각도
- 요구사항과의 일치성
- 기능성/기능에의 영향
- SW 모듈 또는 컴포넌트의 포함 여부
- 비즈니스 프로세스에 끼치는 영향성
- SDLC 단계의 준수

- 다양한 테스트의 단계에서 각 단계가 얼마나 결점 봉쇄를 했는지의 양적인 판단 - 수치 / 앞 단계에서 발견된 결점, 에러의 %
- 한 개 릴리즈, 다수 릴리즈 사이의 테스트 결과 경향 분석
- SDLC 프로세스, 툴 사용 등과의 경향 연결

Determination of Achieved Test Goals & Quality Goals

릴리즈에서 잔존 결점 밀도, 실패율, MTBF, RPN, 가용성 등의 양적인 판단은 테스트 효과성에 결정적인 영향을 준다. 이 데이터는 경쟁자에 대한 비교 & 벤치마크에 도움이 된다.

Earned Value Analysis

수행된 테스트의 실제 비용과 추정 치에서의 벗어남은 프로젝트 릴리즈에 공식적으로 영향을 미쳐야 한다. 이 데이터는 유사한 프로젝트의 시작이나 다음 릴리즈에 참고자료로 사용되어야 한다.

이것은 미래의 테스트 프로젝트 비용 계획 & 제어에 필수적이다.

Conclusion

따라서, 소프트웨어 프로젝트에서 이러한 3 가지 테스트 프로파일을 정의하는 것은 소프트웨어 테스트 관리의 성숙도를 높일 수 있게 해 준다.

부록 - Test Coverage Definitions

공식적이고 양적인 테스트 커버리지의 측정의 정의는 테스트의 유형에 종속적이다. 공식적인 방법 (수학적인)이나 툴이 이것과 잘 맞아야 테스트 커버리지를 측정할 수 있다.

테스트 커버리지 메트릭의 예제가 아래에 있다.

화이트 박스 테스트:

- Block coverage

- Branch coverage
- Value-range coverage
- Call & return
- 실행되고 동작한 코드의 %
- exception coverage

블랙 박스 테스트:

- 기능 점수 커버리지
- 기능 커버리지
- 메시지 커버리지
- Unexpected events coverage
- 에러 코드 커버리지
- 이벤트/시나리오의 교차/충돌 커버리지
- 타이머 커버리지

OO 테스트:

- 클래스 커버리지
- 인스턴스 커버리지
- 메소드 커버리지
- 상속성 커버리지

프로토콜 테스트:

- 메시지 커버리지
- PDU value range coverage
- 상태 커버리지
- 에러 핸들링 커버리지
- 메시지 교차/이벤트 충돌 커버리지
- 타이머 커버리지

SDL 기반 테스트:

- 방문한 상태의 수
- 왕래한 변이 경로의 수

About the Author

The author has been with WIPRO for 6 years now. For the last two years she has been working in the interops division. Lakshmi is an Electronics H/W & F/W engineer. In WIPRO, she has worked with AGCS & Nortel accounts as a s/w developer before moving to Interops in June 2001.

Since 2001, she has been involved in the LIBRA project & TTCN related activities. In 2001-2002, Lakshmi was involved in a DMADV project, which involved a detailed study of "Reliability Engineering". Some of the thoughts expressed in this paper originate from this study.

In her current role, she is responsible for the "Nokia - Symbian Practice group in Bangalore" & the "TTCN center of excellence" of Interops.

{끝}