Software testing

소프트웨어 공학 개론

T8
200611478 성두훈
200611494 원스타
200611518 조민경
200611458 김영승

유준범 교수님 CLASS A

1. Software testing 이란?

- **소프트웨어 테스팅**(software testing)은 개발된 컴퓨터 소프트웨어의 품질을 측정하기 위해 사용되는 과정이다.

- **IEEE 에서의 Software testing 정의**
  - 시스템이나 컴포넌트가 특정된 상황에서 실행되며, 그 결과가 채집되거나 기록되고, 시스템이나 컴포넌트의 특정 관점에서 평가(evaluation)가 이루어지는 일련의 활동, 또는 이를 지휘, 통제.
  - 하나 이상의 테스트 케이스의 집합, 또는 테스트 프로시져 집합.
  - 혹은 이 둘의 집합.

-디버깅과의 차이 말하기.

2. 하드웨어 결함의 이유

- **소프트웨어 결함은 다음의 과정을 통해 일어난다.**
  - 인간은 코드, 소프트웨어, 시스템, 또는 문서 안에 결함을 만들어내는 실수를 범할 수 있다. 결함 코드가 실행되면 시스템은 바라던 결과에 대해 실패할 수 있다.
  - 소프트웨어, 시스템, 문서 안의 결함은 실패로 이어질 수 있지만 모든 결함이 그러한 것은 아니다. 또한 결함이 없다 해도 환경이 바뀌면 실패할 수도 있다. 이러한 변화의 예는 새로운 하드웨어 플랫폼에서 실행되거나, 소스 데이터가 바뀌거나 다른 소프트웨어와 상호 작용하는 것을 들 수 있다.

3. Software testing 의 필요성

- 금전적 손실
- 시간 낭비

 - 프랑스의 Ariane 5호

- 비즈니스 이미지 손실

 - 기업이나 사용자의 요구에 오류로 인해 충족을 못 했을 때

- 부상, 사망에 이르는 심각한 문제 발생

 - 자동차나 비행기 같은 몸체에 내장된 소프트웨어에 문제가 있을 때

4. Software testing 의 종류

1. 블랙박스 테스팅
2. 화이트박스 테스팅
3. 유닛 테스팅
4. 통합 테스팅
5. 기능 테스팅
6. 앤드-투-앤드 테스팅
7. 새너티 테스팅
8. 리그레션 테스팅
9. 인수 테스팅
10. 부하 테스팅
11. 스트레스 테스팅
12. 퍼포먼스 테스팅
13. 사용성 테스팅
14. 설치/삭제 테스팅
15. 회복 테스팅
16. 보안 테스팅
17. 호환성 테스팅
18. 비교 테스팅
19. 알파 테스팅
20. 베타 테스팅
21. 시스템 테스팅

5. Software testing 의 유형

- SWEBOK 에서는 가능한 테스트의 유형에 대한 리스트를 제공한다. 이 리스트는 테스트의 유형을 다음과 같은 속성에 기반해 나눈다.

- 직관과 경험
- Specifications
- 코드
- 데이터 흐름
- 결함(Fault)
- 사용법(Usage)
- Application의 본질

4. Software testing 의 종류

1. 블랙박스 테스팅
2. 화이트박스 테스팅
3. 유닛 테스팅
4. 통합 테스팅
5. 기능 테스팅
6. 앤드-투-앤드 테스팅
7. 새너티 테스팅
8. 리그레션 테스팅
9. 인수 테스팅
10. 부하 테스팅
11. 스트레스 테스팅
12. 퍼포먼스 테스팅
13. 사용성 테스팅
14. 설치/삭제 테스팅
15. 회복 테스팅
16. 보안 테스팅
17. 호환성 테스팅
18. 비교 테스팅
19. 알파 테스팅
20. 베타 테스팅
21. 시스템 테스팅

4. Software testing 의 종류

1. 블랙박스 테스팅
2. 화이트박스 테스팅
3. 유닛 테스팅
4. 통합 테스팅
5. 기능 테스팅
6. 앤드-투-앤드 테스팅
7. 새너티 테스팅
8. 리그레션 테스팅
9. 인수 테스팅
10. 부하 테스팅
11. 스트레스 테스팅
12. 퍼포먼스 테스팅
13. 사용성 테스팅
14. 설치/삭제 테스팅
15. 회복 테스팅
16. 보안 테스팅
17. 호환성 테스팅
18. 비교 테스팅
19. 알파 테스팅
20. 베타 테스팅
21. 시스템 테스팅

4. Software testing 의 종류

1. 블랙박스 테스팅
2. 화이트박스 테스팅
3. 유닛 테스팅
4. 통합 테스팅
5. 기능 테스팅
6. 앤드-투-앤드 테스팅
7. 새너티 테스팅
8. 리그레션 테스팅
9. 인수 테스팅
10. 부하 테스팅
11. 스트레스 테스팅
12. 퍼포먼스 테스팅
13. 사용성 테스팅
14. 설치/삭제 테스팅
15. 회복 테스팅
16. 보안 테스팅
17. 호환성 테스팅
18. 비교 테스팅
19. 알파 테스팅
20. 베타 테스팅
21. 시스템 테스팅

5. Software testing 의 유형(Con.)

- 가장 많이 사용되는 다음의 유형에 대해서는 더 많은 정보가 있다.
  - Equivalence class partitioning
  - Boundary value
  - Decision table
  - Exploratory
  - Operational Profile

(1) Equivalence class partitioning

- 각각의 INPUT을 수용 가능한 범위에 대해 검사해서 INPUT에 대한 다음과 같은 클래스를 판별한다.

- Valid : 올바른 코드에 의해 성공적으로 처리될 수 있는 값의 리스트나 범위
- Invalid : 올바르지 않고 Software에서 허용되진 않지만, 그렇다고 해서 아예 잘못 된 결과를 초래하지는 않는 값의 리스트나 범위

① Equivalence class partitioning의 단계

- 가능한 입력 값을 Valid, invalid 클래스로 분류.
- Valid 클래스의 값은 최대한 많이 테스트.
- Invalid 클래스의 값은 각각에 대해 한번씩만 테스트. 역시 모든 값 테스트. Invalid 클래스는 Valid 클래스와는 달리 연계되지 않음.

(2) Boundary value testing

- INPUT에 대해 4개의 값을 테스트한다.
  - Valid 의 최소값.
  - Valid 의 최대값.
  - 최소값 – 1.
  - 최대값 + 1.

  (숫자에 대해선 매우 명확하다)

(3) Dicision table

- Dicision table은 모든 INPUT과 그 결과로 생긴 모든 결과를 테이블의 첫 컬럼에 열거. 그 뒤, 가능한 모든 INPUT상태의 조합에 대해 Rule이 있다.
- 특정한 INPUT에 대해 Y(yes), N(no), I(immaterial)로 나타낸다.

(4) Exploratory testing

- 테스팅 프로세스의 초점을 계획하는 것에 중점을 둔다
- 이전 Realease 와 Product Line과의 호환성이나, 한 프로젝트 내에서 처음부터 끝까지 일관성 있게 움직이는지를 테스트한다.

(5) Operation profile

- 운영 도중, 각각 시스템 기능에 대해 실행되는 테스트의 횟수를 알 수 있다. 실제 사용량이 측정될 수도 있다. 따라서 더 많이 사용되는 기능을 더 많이 테스트함으로써 소프트웨어의 견고성을 높힐 수 있다.

6. 테스트 레벨

- 소프트웨어가 개발되고 유지됨에 따라 한 레벨에서 한번 이상의 테스트가 수행된다.
- 작은 범위부터 큰 범위로 테스트한다.
- 소프트웨어의 범위, 테스트 목적, 테스트 테크닉, 환경에 따라 테스트 레벨이 변한다.

요인

- 시스템 크기
- 안전성의 중요도
- 수요자의 요구
- 복잡도
- 관리자의 경험/경력

7. 테스트 전략

- 테스트 전략은 보통 “Macro” 나 “Micro” 중 하나에 초점을 둔다.

Macro

- Time to market(테스트 개발과 수행 속도)
- 제공되어야 하는 기능의 양
- 제품의 품질(테스트의 완벽성)

7. 테스트 전략 (Con.)

- 개발을 빠르게 하기 위해서는
  - 테스트 수행을 더 빠르게 하기 위해 자동화 도입
  - 러닝 타임을 줄이기 위해 테스트 스탭 교체를 덜 하는 것
  - 실행되는 테스트 케이스를 잘 선택해서 심각한 문제를 빨리 찾을 수 있도록 함
- 제품의 퀄리티를 위해
  - 더 좋은 툴로 현재 테스트 커버리지를 측정
  - 개발자들을 위해 더 좋은 유닛 테스트 툴을 제공
  - 테스트 유형을 더 다양하게 만듬.

7. 테스트 전략 (Con.)

- 테스트 비용을 줄이기 위해
  - 프로젝트 관리 툴을 사용해 테스트 활동을 예측, 실제 비용 지출을 확인.
  - 기저 원인 분석을 추가해 발생한 문제의 원천을 찾고 프로세스를 변경해 문제가 다시 발생하지 않도록 함.

8. 테스트 디자인

- 테스트 디자인은 예술과 과학을 접목해 각각의 테스트 레벨에서 사용할 가장 알맞은 테스트 기법을 선택하는 것이다. 모든 기법을 사용하려면 리소스를 너무 많이 먹을 것이고, 쓸모없는 테스팅 결과를 유발할 것이다. 대부분의 테스트 디자인의 목표는 최소한의 노력으로 최대한의 결과를 얻어내는 것이다.

8. 테스트 디자인 (Con.)

- 테스트 디자인은 “Structured”와 “Unstructured” 기법을 모두 포함한다.

- Unstructured 의 예
  1. Random
  2. Ad hoc
  3. Exploratory

- Structured 의 예
  1. Equivalence class partitioning
  2. Boundary value
  3. Decision table

8. 테스트 디자인 (Con.)

- Structured 테스트 기법의 장점
  - Linear 한 커버리지를 제공.
  - 모든 Attribute 가 같은 관점에서 같은 방식으로 테스트
- Unstructured 테스트 기법의 장점
  - Structured 기법보다 많은 문제를 발견
  - 심각한 문제를 발견할 확률 높음

9. 코드의 테스트 커버리지

- 코드 커버리지의 목적은 테스트 레벨에 따라 달라짐
- System testing에서는 커버리지가 측정될 수도 있지만 목표치는 100%에 근접하지 못 한다.
- 모든 명령문을 100% 커버할 수 있는 기법은 Tom McCabe's Basis Path Testing.

(1) Tom McCabe's Basis Path Testing.

- Flowgraph를 그린다.
  - 각각의 논리적 명령문을 원(노드)이라고 표현.
  - 결정 사항의 결과로 컨트롤이 이전되는 것을 화살표(엣지)
- Cyclomatic cimplexity를 계산
- (엣지갯수 – 노드갯수 + 2)
- 경로를 정한다

10. 테스트 범위

- 개발을 위한 원본 문서는 포괄적인 용어인 “Specifications”로 통합됨.
- 몇몇 테스트 레벨의 목표는 하나 또는 그 이상의 Specification을 모두 다루는 것
  - Specification이 어떤 방법으로 목록화되어야만 측정가능.

11. 테스트 실행

- 테스트 실행을 위해서는 Test Plan(계획)이 필요.
- 모든 입력과 절차의 실제 테스트는 테스트 경우와 테스트 사례와 테스트 절차를 자세하게 기술해야함
- 테스트 결과는 각각의 테스트 사례의 성공과 실패의 평가가 포함된 테스트 계획을 실행하는 동안에 기록됨 -> Incident report

12. 테스트 문서화

- 테스트 문서화는 다양한 매체에 기록한다.