
Software Testing
by Claire Lohr

Definition

An activity in which a system or component

under specified conditions,

the results are observed or recorded,

and an evaluation is made of some aspect

of the system or component.


Type Of Testing
Testing Level
Test Management
Testing Strategies
Test Documentation
Software Testing
Test Execution
Test Design
Test Coverage Of Code

Attribute

- the Software engineer's intuition and experience.
- Specification.
- Code.
- Dataflow.
- Fault.
- Usage.
- the Nature of the application

Type Of Testing

- **Equivalence class partitioning** (이하 EQ테스팅)

- **Boundary value.**

- **Decision table.**

- **Exploratory.**

- **Operational profile.** `

Equivalence class partitioning

- **According to two attributes:**
  - **Valid / Invalid.**
  - **Example: age 0-120 (valid)**
  - **age <0 and >120 (Invalid)**
- **step**
  - **Define valid and invalid values.**
  - **Test as many valid classes together.**
  - **Do one test for each invalid class.**

Equivalence class partitioning

동등 분할 기법(Equivalence Partitioning)

프로그램의 입력 영역을 테스트 케이스가 도출될 수 있는 데이터의 유형들로 나누는 테스트 기법으로 입력조건을 2개 이상의 균등 유형으로 분할하여 테스트 케이스를 설계하는 기법이다.

주어진 입력 조건에 따른 테스트 케이스 수를 결정하는데 유용하다.

입력조건	유효 균등 유형	무효 균등 유형	전체 테스트 유형의 수
성별의 구분은 1(남)이나 2(여)를 사용한다	성별이 1 또는 3 중에 하나	성별이 1 또는 2가 아닌 경우	2
학급번호의 범위는 1반 – 9반이다	1≤학급번호≤9	학급번호 < 1 학급번호 > 9	3
성적은 숫자이다	성적은 0 – 9 사이의 숫자로 구성	성적이 공백이나 숫자가 아닌 경우	2

- 테스트 유형의 수를 결정하는데 사용한다.

Boundary value

- **Must test the following:**

- Legally defined minimum.

- Legally defined maximum.

- The first possible value below the legally min.

- The first possible value above the legally max.

: Age ex: 0, 120, -1, 121

Boundary value

경계 값 분석(Boundary-Value Analysis)

균등 분할 기법을 보완해주는 테스트 케이스의 설계 기법으로 균등 유형의 어떤 요소를 선택하는 것이 아니라 유형의 주변에서 테스트 케이스를 만들어 내는 기법이다.

단지 입력 조건에 초점을 맞추지 않고 출력 영역으로부터 테스트 케이스를 유도해내는 방법이다.

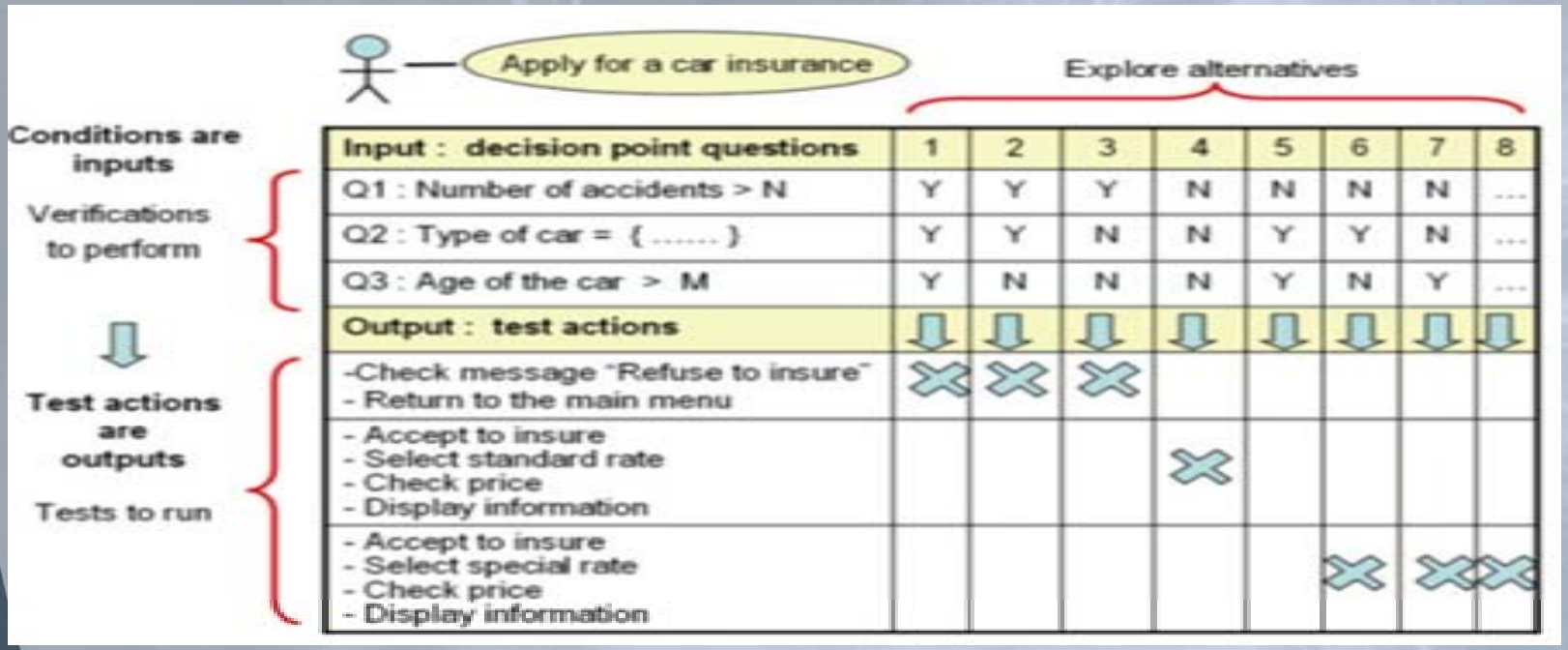
... -3, -2, -1, 0	1, 2, 3, ... 99, 100	101, 102, 103, ...

- 테스트 대상의 입력 값의 범위 가정 수 1~100 이라면 균등 분할 기법에서는
  - 유효 균등 유형은 1~100 이고,
  - 무효 균등 유형은 0, -1, -2, -3 과 101, 102, 103 이 된다.
  - 이 경우 경계 값은 1, 0, 100, 101 이 되며 이것을 그림으로 표시하면 위와 같다.

- 주어진 테스트 유형에서 테스트 케이스를 도출하는데 적절하다.

Decision table

- ***Lists all conditions and all resultant outputs.***
  - **Y :yes**
  - **N: no**
  - **1:yes and no.**



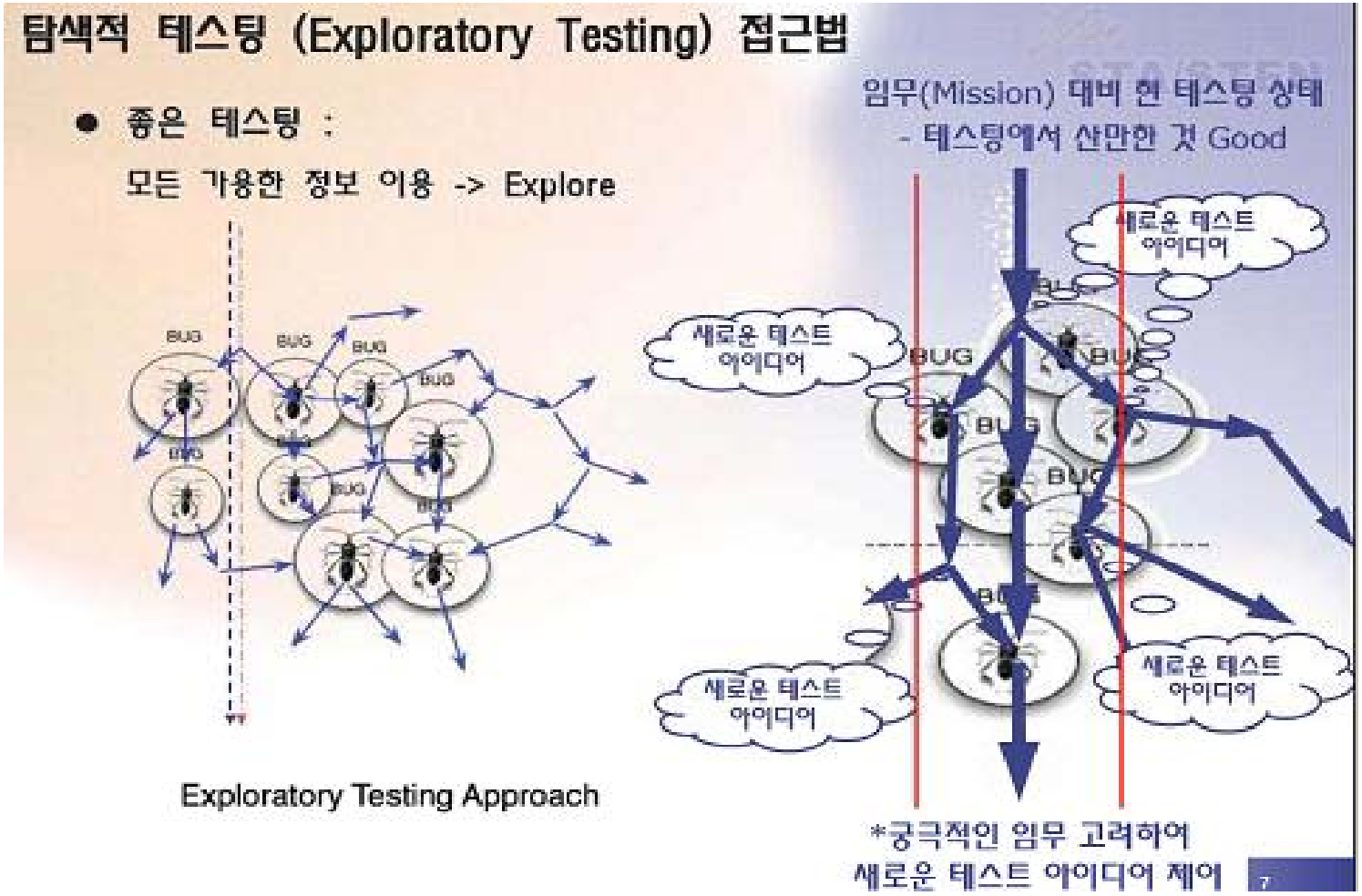
Input : decision point questions	1	2	3	4	5	6	7	8
Q1 : Number of accidents > N	Y	Y	Y	N	N	N	N	...
Q2 : Type of car = { ..... }	Y	Y	N	N	Y	Y	N	...
Q3 : Age of the car > M	Y	N	N	N	Y	N	Y	...
Output : test actions	⇩	⇩	⇩	⇩	⇩	⇩	⇩	⇩
-Check message "Refuse to insure" - Return to the main menu	X	X	X					
- Accept to insure - Select standard rate - Check price - Display information				X				
- Accept to insure - Select special rate - Check price - Display information						X	X	X

< Decision table >

Exploratory

- **Emphasizing planning the focus of the testing process.**
- **Doesn't require pre-planning.**
- **When test begins, modifications are still allowed.**
- **Test results modify the plan accordingly.**

Exploratory



Operational profile

- **Number of tests to be done follows the model of how much it is used during operation.**
- **Test more for features that are used more.**

Testing Levels

- **Levels vary from organization to organization, and from project to project.**

Some parameters:

- Size.

- Complexity.

- Safety criticality.

- Experience of staff.

- Desire for certification(ISO, CMMI)

Testing Levels

- ***acquisition***
- ***development***
- ***Operational process:==>operational testing***
  - ***Software units, DB, integrated components, requirements and integration.***
- ***Maintenance process:***
  - ***modified and un-modified parts, migration verification, parallel testing, all development levels for improvements.***
- ***Supporting process:***
  - ***Verification and validation.***

Testing Levels

1. 컴퍼넌트 테스트(Component Testing)
   컴포넌트 테스팅은 개별적으로 테스트가능한 소프트웨어
   (예를 들어, 모듈, 프로그램, 객체, 클래스 등)안의 결점을 찾고, 기능을 검증한다.

2. 통합 시험(Integration Testing)
   통합 테스팅은 컴포넌트들 사이의 인터페이스, 운영 시스템, 파일 시스템,
   하드웨어나 시스템 사이의 인터페이스와 같은 시스템의 여러 부분에
   대한 상호작용(interactions)을 테스트한다

3. 시스템 테스팅(System Testing)
   시스템 테스팅은 개발 프로젝트나 프로그램의 범위에 의하여
   정의된 전체 시스템/제품의 동작과 관련이 있다.
   시스템 요소가 적절히 통합되고 할당된 기능을 수행하는지를 검증한다.

4. 인수/승인 테스팅(Acceptance Testing)
   인수 테스팅은 시스템을 사용하는 고객이나 사용자가 전담하여 수행하는 경우가
   대부분인데, 다른 관련자도 참여할 수 있다.
   반드시 최종 단계의 테스팅이라고 보기는 어렵다.

Testing strategies

- **Testing all combinations is impossible!**

- **Macro:**

Time to market

Amount of functionality to be delivered

Quality of product

- **Micro:**

More automation to speed process

Better selection of test cases

Less turnovers in testing staff.

Testing strategies

- **For functionality changes:**
  - **Tracing requirements to see cases affected by changes.**
  - **Modular testing.**
- **For quality of product:**
  - **More variety in the test types.**
  - **Better tools.**
  - **Better unit test tools for developers**
- **to control test effort costs**
  - **use a project management tool to estimate test activities and to track the actual expenditures**
  - **add root cause analysis**

Test Design

- **Structured**
  - **Boundary value**
  - **Decision table**
  - **Equivalence class partitioning**
- **Un structured**
  - **Random**
  - **Ad hoc**
  - **Exploratory**

Test Design

- 소프트웨어(시스템) 내부구조(코드)의 참조여부에 따라 블랙박스 기법과 화이트박스 기법으로 양분하는 방법 이외에 테스트 설계의 근원(Origin)을 기준으로 명세 기반 기법, 구조 기반 기법, 경험 기반 기법으로 이해하기 용이하게 분류할 수 있다.

정적 테스팅	동적 테스팅	
	블랙 박스 테스팅	화이트박스 테스팅
▪ Inspection ▪ Orthogonal Array Testing ▪ Prior Defect History Testing ▪ Risk Based Testing ▪ Run Chart ▪ Statistical Profile Testing	▪ Boundary Value Testing ▪ Cause-Effect Graphing ▪ Control Flow Testing ▪ CRUD Testing ▪ Decision Tables Testing ▪ Equivalence Class Partitioning ▪ Exception Testing ▪ Finite State Testing ▪ Free Form Testing ▪ Positive and negative Testing ▪ Prototyping ▪ Random Testing ▪ Range Testing ▪ Regression Testing ▪ State Transition Testing ▪ Thread Testing	▪ Basis Path Testing ▪ Branch Coverage Testing ▪ Condition Coverage Testing ▪ Data Flow Testing ▪ Loop Testing ▪ Mutation Testing ▪ Sandwich Testing ▪ Statement Coverage Testing
	▪ Desk checking (블랙박스, 화이트박스)	

Test Coverage of code

- **Impossible to cover all the code.**
- **Depends on the definition of "covering"!**
- **Does it mean executing, reading, testing?**
- **Use Tom McCabe's basis technique.**

Tom McCabe's basis technique

1. Draw a flow graph.(nodes and edges)

2. Compute the metric cyclomatic complexity (nodes).

   : No.edges-no.Nodes+2=cc

3. Choose paths. (cover all nodes and edges)

Tom McCabe's basis technique

구조 테스트

1982 년 T. Mccabe 에 의해서 제안된 화이트박스 테스트의 대표적인 방법으로 프로그램의 논리적 복잡도를 측정한 후 이 척도에 따라 수행할 기본 경로들의 집합을 정의한다.

프로그램의 논리적 복잡도는 프로그램의 순환적 복잡도라고 부르며 이 복잡도를 측정하기 위해서 프로그램의 수행 경로를 프로그램 라인으로 표시하는 노드와 수행 방향을 표시하는 화살표의 그래포로 나타낸 후 다음 공식에 적용한다.

- 복잡도(V) = 화살표의수(A) -노드의 수(N) + 2
- 이때 각 노드는 원칙적으로 프로그램 라인 한 줄에 해당됨

구조 테스트수행 절차는 다음과 같다.

- 상세 설계나 원시 코드를 기초로 논리 흐름도를 작성한다.
- 공식을 이용하여 논리적 복잡도를 구한다.
- 모든 독립 경로들은 테스트 케이스를 준비할 기본 경로들의 집합이 되며, 이 집합의 각 경로마다 적절한 테스트 케이스를 준비한다.

Test Execution

- **Defined in a test plan**

  (Pseudo code and flow graph,

  requirement summary, test scenario/cases)

- **Cover all specifications is important.**

  (categorize according to testable or not)

- **Define components:**

  hardware, software, personnel, data, automated tools

Test Execution

Table 2

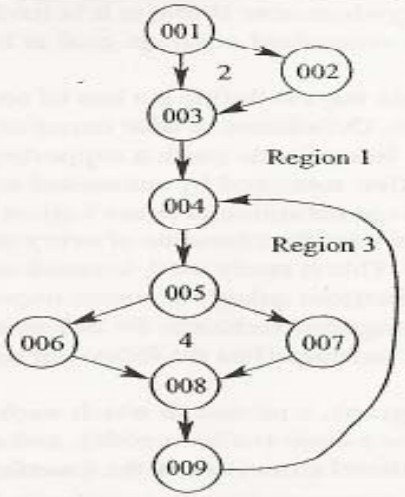
Code statements (in simplistic pseudocode)	Corresponding flowgraph
001 If A 002 then B 003 End if 004 For 1 to n, do 005 If more data, 006 Then add to total C 007 Else send error message D 008 End if 009 End for The Cyclomatic Complexity is equal to the number of regions, which is 4. Four possible test cases would follow these paths: 001,003,004,005,006,008,009 001,002,003,004,005,008,009 001,003,004,005,007,008,009 001,003,004,005,006,008,009,004,005,007,008,009 In this case, Cyclomatic Complexity could be reduced to Actual Complexity of 2, with the following paths: 001,003,004,005,006,008,009 001,002,003,004,005,007,008,009,004,005,006,008,009	

Table 3

ID #	Requirement summary	Test scenario/cases
001	Selection of items for purchase.	PUR002/35-40, 66-85 PUR003/1-57
002	Entry and validation of credit card information	CRC001/1-36 CRC002/5-89 CRC003/5-34, 75-115
003	Limit the maximum items purchased at one time as a fraud countermeasure	PUR001/56-77 PUR002/41-65

<plan>

Testing documentation

- **Media vary depending on level of detail:**

  - Word doc.

  - Spread sheets.

  - Databases.

  - Within automated test tools.

and you need IEEE Testing documentation!!!

IEEE Testing documentation

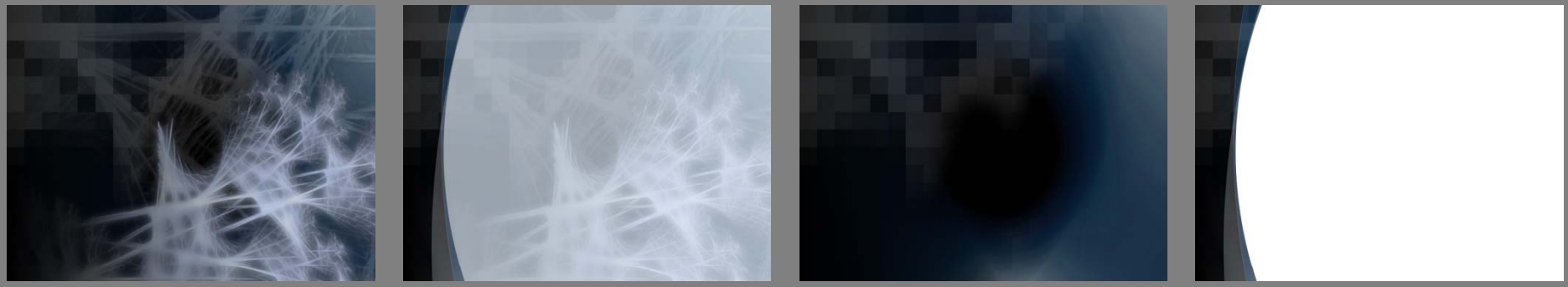
- Test plan
- Test design
- Test cases
- Test procedures
- Test logos
- Incident reports
- Test summary report
- Test item transmittal report

Test management

- Estimating schedules.
- Planning for staffing and training.
- Identifying and planning tasks.
- Monitoring the execution of the plans and re-planning based on results.

FINAL

- 테스팅 ≠ 디버깅
- 테스트 활동은 테스트를 수행하기 **전과 후**에도 존재하며, **테스팅 계획과 제어 같은 활동**이나 **테스트 조건(Test condition)**의 선택, **테스트 케이스의 디자인, 테스트 수행 결과 점검, 테스트 완료 및 통과 조건의 평가,** 테스트 프로세스와 테스트 중인 시스템에 대한 **리포트, 마무리 또는 마감(Finalizing or closure)**과 같은 일련의 활동들을 포함한다 또한 테스팅은 문서(소스 코드 포함)의 리뷰와 정적 분석**(Static nalysis)**에 의한 테스팅을 포함한다.
- 현재 국내에서는 테스팅에 대한 인식이 많이 부족한 실정 반면 요구하는 능력을 갖춘 인력이 거의 없다



-END-

컴퓨터공학과 **200412307** 김상은