ST Final Review

Lec1

Fault, error, failure

- Fault – Incorrect lines of code
- Error – Faults cause incorrect (unobserved) state
- Failure – Errors cause incorrect (observed) behavior

Writing Junit Test

- 左边的是expect，右边是actual
- @Test

  public void Test1(int a, int b){

  int ans=add(a,b);

  assertEquals(0,ans);

  }
- Test exception
- Try catch and fail

  try{

  int ans=add(a,b);

  fail("");

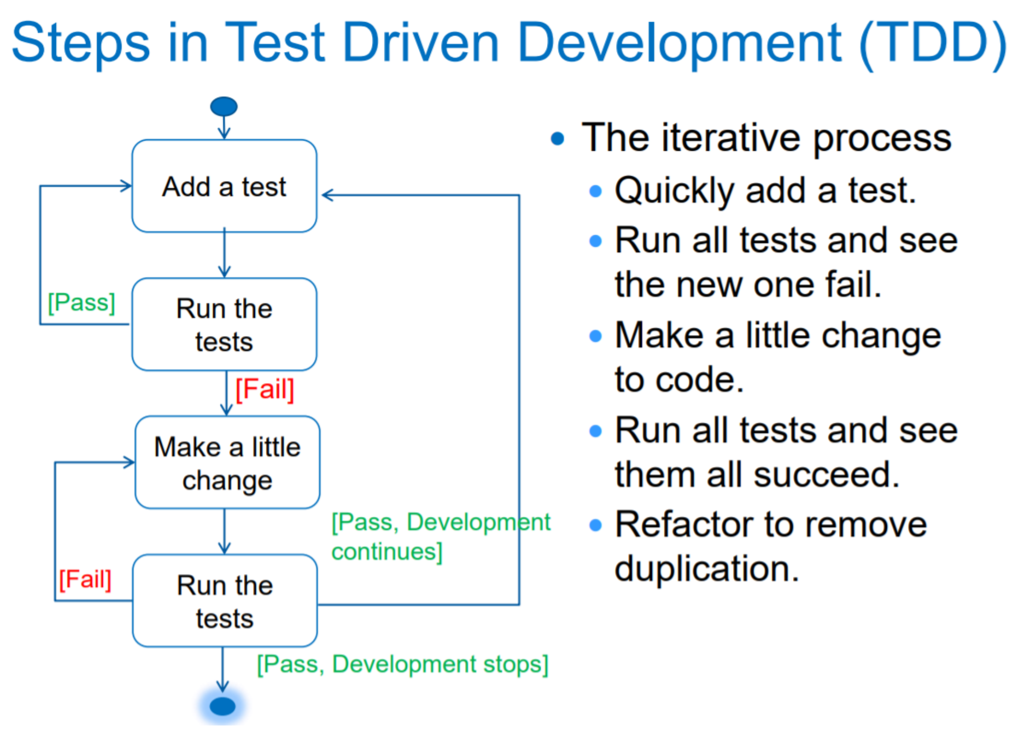
  }

  catch(Exception e){

  assertThat(e.getmassage(), is("test.txt (No such file or directory)"))}
- expected=Exception.class

  @Test(expected=Exception.class)

TDD(Test driven development)


Steps in Test Driven Development (TDD)

- The iterative process
  - Quickly add a test.
  - Run all tests and see the new one fail.
  - Make a little change to code.
  - Run all tests and see them all succeed.
  - Refactor to remove duplication.

IDM(Input domain modeling)

Interfaced-based & functional based

Interface-based Fun	Functionality-based
Develops characteristics directly from individual input parameters Consider syntax	Develops characteristics from a behavioral view of the program under test Consider domain and semantic knowledge
Example：relationship with zero(>0, =0, <0)	Example：Types of Triangle

Coverage Criteria for IDM

All Combinations (ACoC)	Each Choice Coverage (ECC)	Pair-Wise Coverage (PWC)	Base Choice Coverage (BCC)
All combinations of blocks from all characteristics must be used.	One value from each block for each characteristic must be used in at least one test case.	A value from each block for each characteristic must be combined with a value from every block for each other characteristic.	A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic

Lec2

Type of classes

- Data Class
- Utility Class
- Main Class

Statement Coverage & Branch coverage

Testing level

Level 0	There's no difference between testing and debugging
Level 1	The purpose of testing is to show correctness
Level 2	The purpose of testing is to show that the software doesn't work
Level 3	The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
Level 4	Testing is a mental discipline that helps all IT professionals develop higher quality software

Junit Test

- @Before call before every test method
- @After call after every test method
- @Test

Black box test & White box test

Better Test

- independent
- specified only one test
- (expected , actual)
- Use equals to compare Strings

Lec3

TDD (Test driven development)

- Continuous Integration
- User story
- user story=》test》refactor=》test=》 refactor =>...=>pass

Test-Driven Development (TDD) ensures that you built the software correctly, Not that the correct software was built.

BDD (Behavior-Driven Development)

BDD not only helps you develop software correctly, butit ensures you develop the correct software.

团队内部人员协作开发，交流多样化，使用对话和scenario来model程序如何运行

Criteria-Based Test Design

TR (Test requirements)

A specific element of a software artifact that a test case must satisfy or cover

coverage criteria

A rule or collection of rules that impose test requirements on a test set

Infeasible test requirements

test requirements that cannot be satisfied

Coverage Level

The ratio of the number of test requirements satisfied by T to the size of TR

Partitioning Domain

- Disjoint
- Complete

IDM (Input Domain Model)

Steps:

1. Base Choice coverage criterion (BCC)
2. Happy path
3. TR
4. infeasible TR
5. Revised infeasible test requirements
6. Test

Happy path

- all values are true

Lec5

Graph

- Path : A sequence of nodes – [n1 , n2 , ..., nM] : Each pair of nodes is an edge
- Length : The number of edges: A single node is a path of length 0
- Subpath : A subsequence of nodes in p is a subpath of p
- Reach (n) : Subgraph that can be reached from n
- Test Path : A path that starts at an initial node and ends at a final node
- SESE graphs : All test paths start at a single node and end at another node
  - Single-entry, single-exit

    - N0 and Nf have exactly one node
- Visit : A test path p visits node n if n is in p
- Tour : A test path p tours subpath q if q is a subpath of p
- Syntactic reach : A subpath exists in the graph
- Semantic reach : A test exists that can execute that subpath

Node Coverage (NC)

Test set T satisfies node coverage on graph G iff for every syntactically reachable node n in N, there is some path p in path(T) such that p visits n.

- Node Coverage (NC) : TR contains each reachable node in G.

Edge Coverage (EC)

TR contains each reachable path of length up to 1, inclusive, in G.

Edge-Pair Coverage (EPC)

TR contains each reachable path of length up to 2, inclusive, in G.

Complete Path Coverage (CPC)

TR contains all paths in G

If a graph contains a loop（循环）, it has an infinite number of paths .Thus, CPC is not feasible .

Specified Path Coverage（SPC） is not satisfactory because the results are subjective and vary with the tester

Specified Path Coverage (SPC)

TR contains a set S of test paths, where S is supplied as a parameter.

Simple Path

A path from node ni to nj is simple if no node appears more than once, except possibly the first and last nodes are the same

Prime Path

A simple path that does not appear as a proper subpath of any other simple path

Prime Path Coverage (PPC)

TR contains each prime path in G

Tour With Sidetrips

A test path p tours subpath q with sidetrips iff every edge in q is also in p in the same order

Tour With Detours

A test path p tours subpath q with detours iff every node in q is also in p in the same order

Round-Trip Path

A prime path that starts and ends at the same node

Simple Round Trip Coverage (SRTC)

TR contains at least one roundtrip path for each reachable node in G that begins and ends a round-trip path.

Complete Round Trip Coverage (CRTC)

TR contains all round-trip paths for each reachable node in G.

Data Flow Analysis

P-Use

- A usage node is a predicate use (P-Use) if variable v appears in a predicate expression (e.g., x>y)

C-Use

- A usage node is a computation use (C-Use) if variable v appears in a computation (e.g., x+y)

du-path

- A definition-use path (du-path) with respect to a variable v is a path whose first node is a defining node for v, and its last node is a usage node for v

Definition (def)

A location where a value for a variable is stored into memory

Use

A location where a variable’s value is accessed

Data Flow Test Criteria

All-defs coverage (ADC)

For each set of du-paths S = du (n, v), TR contains at least one path d in S

All-uses coverage (AUC)

For each set of du-paths to uses S = du (ni , nj , v), TR contains at least one path d in S.

All-du-paths coverage (ADUPC)

For each set S = du (ni, nj, v), TR contains every path d in S.

CFG (Control flow graph)

Lec7

Call Graph

- Nodes : Units (in Java – methods)
- Edges : Calls to units

- Node and edge coverage of class call graphs often do not work very well
- Individual methods might not call each other at all!

Call Coverage

TR contains each reachable node in the call graph of an object instantiated for each class in the class hierarchy

Object Call Coverage

TR contains each reachable node in the call graph of every object instantiated for each class in the class hierarchy.

Last-def

The set of nodes that define a variable x and has a def-clear path from the node through a callsite to a use in the other unit – Can be from caller to callee (parameter or shared variable) or from callee to caller as a return value

First-use

The set of nodes that have uses of a variable y and for which there is a def-clear and use-clear path from the callsite to the nodes

Lec8

Logic Coverage

Predicates

A predicate is an expression that evaluates to a boolean value

- logical operators:

  与或非，异或，等价，导致

Clauses

A clause is a predicate with no logical operators

Predicate Coverage (PC)

For each p in P, TR contains two requirements: p evaluates to true, and p evaluates to false.

Clause Coverage (CC)

For each c in C, TR contains two requirements: c evaluates to true, and c evaluates to false.

Active Clauses

- Determination :

  A clause ci in predicate p, called the major clause, determines p if and only if the values of the remaining minor clauses cj are such that changing ci changes the value of p

Active Clause Coverage (ACC)

For each p in P and each major clause ci in Cp, choose minor clauses cj, j != i, so that ci determines p. TR has two requirements for each ci: ci evaluates to true and ci evaluates to false.

General Active Clause Coverage (GACC)

For each major clause c, choose minor clauses such that c determines the predicate • Clause c has to evaluate to true and false • Minor clauses do not need to be the same

Restricted Active Clause Coverage (RACC)

For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci determines p. TR has two requirements for each ci : ci evaluates to true and ci evaluates to false. The values chosen for the minor clauses cj must be the same when ci is true as when ci is false, that is, it is required that cj (ci = true) = cj (ci = false) for all cj .

Correlated Active Clause Coverage (CACC)

For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci determines p. TR has two requirements for each ci : ci evaluates to true and ci evaluates to false. The values chosen for the minor clauses cj must cause p to be true for one value of the major clause ci and false for the other, that is, it is required that p(ci = true) != p(ci = false).

Inactive Clause Coverage (ICC)

- Inactive clause coverage takes the opposite approach – major clauses do not affect the predicates

For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci does not determine p. TR has four requirements for each ci : (1) ci evaluates to true with p true, (2) ci evaluates to false with p true, (3) ci evaluates to true with p false, and (4) ci evaluates to false with p false.

General Inactive Clause Coverage (GICC)

For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci does not determine p. The values chosen for the minor clauses cj do not need to be the same when ci is true as when ci is false, that is, cj (ci = true) = cj (ci = false) for all cj OR cj (ci = true) != cj (ci = false) for all cj .

Restricted Inactive Clause Coverage (RICC)

For each p in P and each major clause ci in Cp, choose minor clauses cj , j != i, so that ci does not determine p. The values chosen for the minor clauses cj must be the same when ci is true as when ci is false, that is, it is required that cj (ci = true) = cj (ci = false) for all cj .

Making Clauses Determine a Predicate

- pc = pc=true 异或 pc=false

结果为minor clause的值

Lec9

BNF grammar

Terminal Symbol Coverage (TSC)

TR contains each terminal symbol t in the grammar G.

Production Coverage (PDC)

TR contains each production p in the grammar G.

Derivation Coverage (DC)

TR contains every possible string that can be derived from the grammar G.

Mutation Testing

Ground string

- A string in the grammar

Mutation Operator

A rule that specifies syntactic modifications of strings generated from a grammar

Mutant

The result of one application of a mutation operator

- 一次一个mutation operator，每一个可能的operator都应该被尝试

Killing Mutants

Given a mutant m □ M for a derivation D and a test t, t is said to kill m if and only if the output of t on D is different from the output of t on m

Mutation Coverage (MC)

For each m ∈M, TR contains exactly one requirement, to kill m.

- Coverage in mutation = numbers of mutants killed
- The amount of mutants killed is called the mutation score

Mutation Operator Coverage (MOC)

For each mutation operator, TR contains exactly one requirement, to create a mutated string m that is derived using the mutation operator.

Mutation Production Coverage (MPC)

For each mutation operator, TR contains several requirements, to create one mutated string m that includes every production that can be mutated by that operator

Type of mutants

- Dead mutant : A test case has killed it
- Stillborn mutant : Syntactically illegal
- Trivial mutant : Almost every test can kill it
- Equivalent mutant : No test can kill it (same behavior as original)

RIPR model

- Reachability : The test causes the faulty statement to be reached (in mutation – the mutated statement)
- Infection : The test causes the faulty statement to result in an incorrect state
- Propagation : The incorrect state propagates to incorrect output
- Revealability : The tester must observe part of the incorrect output

Strongly Killing Mutants

Given a mutant m ∈ M for a program P and a test t, t is said to strongly kill m if and only if the output of t on P is different from the output of t on m

Weakly Killing Mutants

Given a mutant m ∈ M that modifies a location l in a program P, and a test t, t is said to weakly kill m if and only if the state of the execution of P on t is different from the state of the execution of m on t immediately after l

- Weakly killing satisfies reachability and infection, but not propagation

Weak Mutation Coverage (WMC)

For each m ∈ M, TR contains exactly one requirement, to weakly kill m.

Lec11

Automated Debugging

- Statistical Fault Localization
- Dynamic Slicing
- Delta Debugging

Delta Debugging

Testing Test Cases

Minimizing Test Cases

1-minimal Input

Find a set of changes that cause the failure, but removing any change causes the failure to go away

任意移除一个字符，都能通过 test

global minimal

能fail test里面最短的

local minimal

他的subsequence没有能fail test的

2-minimal

任意移除2个字符，都能通过 test

Minimization Algorithm

从n=2和delta开始，如果分了之后能fail，n=n-1，delta变为任意一个能fail的部分

否则delta不变，n=n*2

Lec13

Symbolic Techniques for Debugging and Testing

Fuzzing

Mutation-based vs. Generation-based

- Mutation-based fuzzer
  - Pros: Easy to set up and automate, little to no knowledge of input format required
  - Cons: Limited by initial corpus, may fall for protocols with checksums and other hard checks
- Generation-based fuzzers
  - Pros: Completeness, can deal with complex dependncies (e.g, checksum)
  - Cons: writing generators is hard, performance depends on the quality of the spec

Lec14

Preconditions

Postconditions