Model-Driven Testing

Software Testing

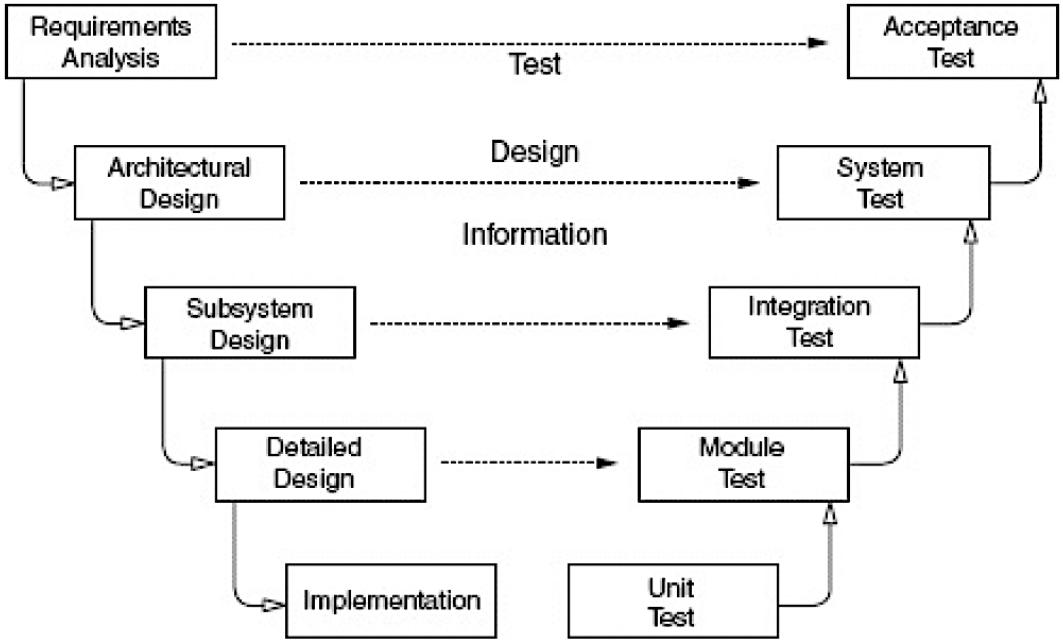
(3104313)

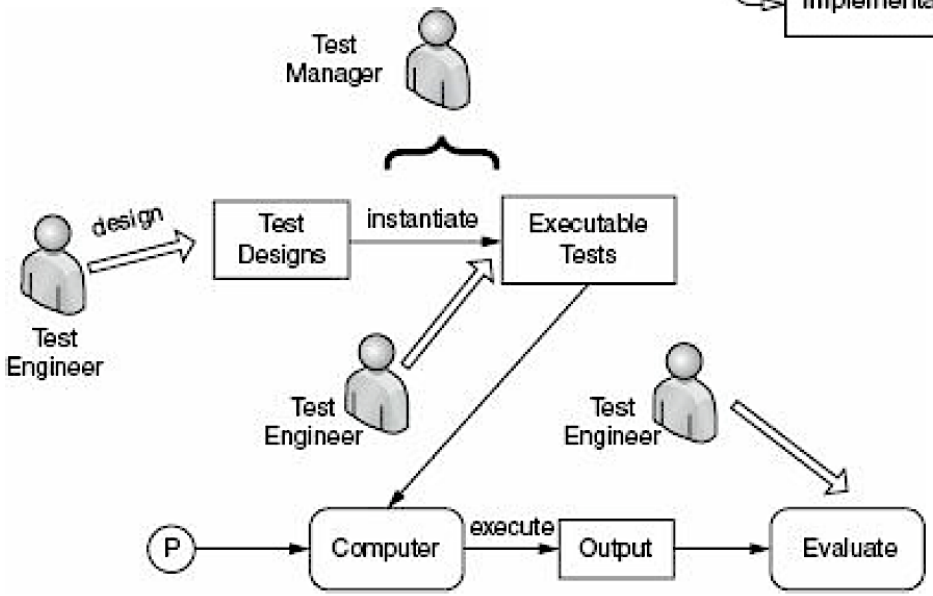
Amirkabir University of Technology

Spring 1399-1400

Recall …

- Testing Activities
- Testing Levels
- Coloured-box Testing
- ….





In-Class Exercise #5

Answer question (a) or (b), but not both, depending on your background.

a) If you do, or have, worked for a software development company, how much effort did your testing/QA team put into each of the four test activities? (test design, automation, execution, evaluation)

b) If you have never worked for a software development company, which of the four test activities do you think you are best qualified for? (test design, automation, execution, evaluation)

- You have 5 minutes
- Do the exercise individually
- Upload your answer in Moodle.

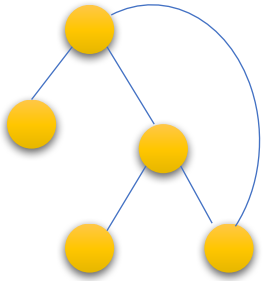
Coverage Criteria

- Coverage criteria give us structured, practical ways to search the input space.
- A coverage criterion is a **rule** or collection of rules that yield **test requirements**.

(not X or not Y) and A and B

```
If (x < y)
      z = x - y
else
      z = 2*x
```

A: {0, 1, >1}
B: {600, 700, 800}
C: {swe, cs, isa, infs}



Coverage Criteria

- Coverage criteria give us structured, practical ways to search the input space.
- A coverage criterion is a **rule** or collection of rules that yield **test requirements**.

Model-Driven Test Design

Illustrative Example

Software Artifact : Java Method

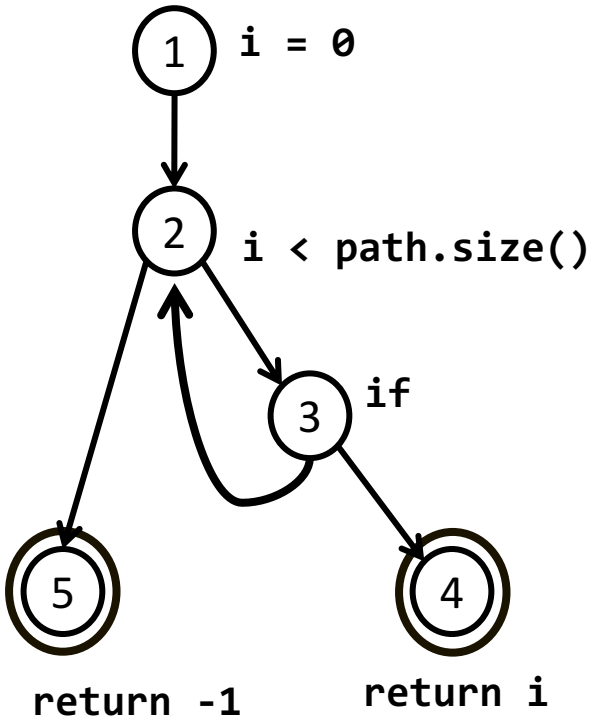
```
/**
    * Return index of node n at the
    * first position it appears,
    * -1 if it is not present
*/
public int indexOf (Node n)
{
     for (int i=0; i < path.size(); i++)
          if (path.get(i).equals(n))
               return i;
     return -1;
}
```

Illustrative Example

Software Artifact : Java Method

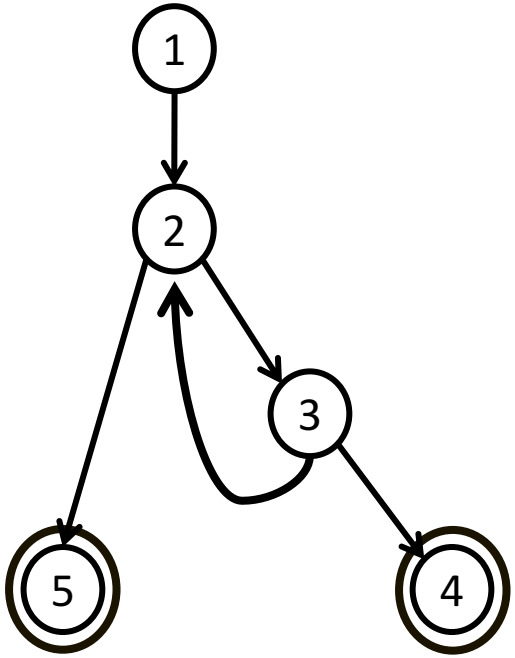
```
/**
    * Return index of node n at the
    * first position it appears,
    * -1 if it is not present
*/
public int indexOf (Node n)
{
     for (int i=0; i < path.size(); i++)
          if (path.get(i).equals(n))
               return i;
     return -1;
}
```

Control Flow Graph



Illustrative Example

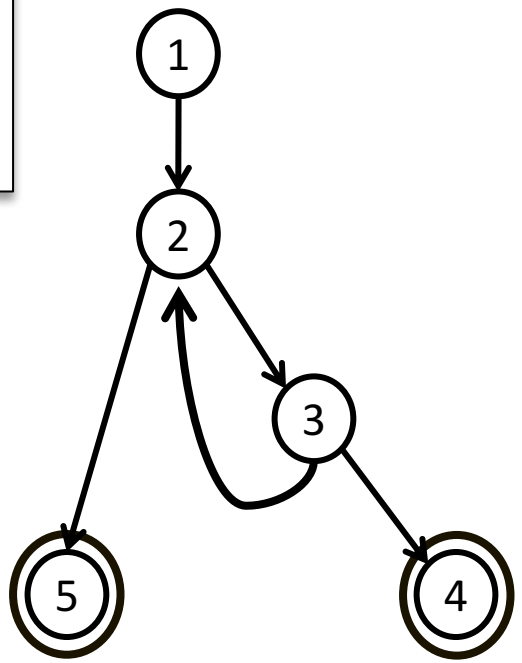
Graph Abstraction



Illustrative Example

Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

Graph Abstraction



Illustrative Example

Edge-Pair Coverage

➔ **6 Test Requirements**

1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

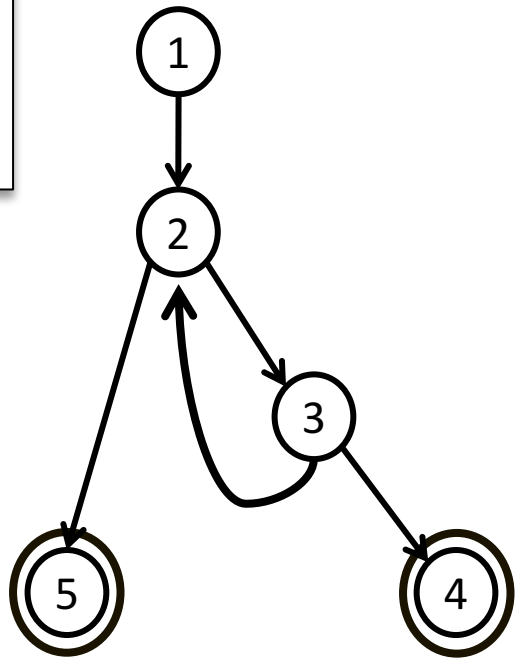
Edges

1 2
2 3
3 2
3 4
2 5

Initial Node: 1

Final Nodes: 4, 5

Graph Abstraction



Illustrative Example

Edge-Pair Coverage

➔ **6 Test Requirements**

1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

Edges

1 2
2 3
3 2
3 4
2 5

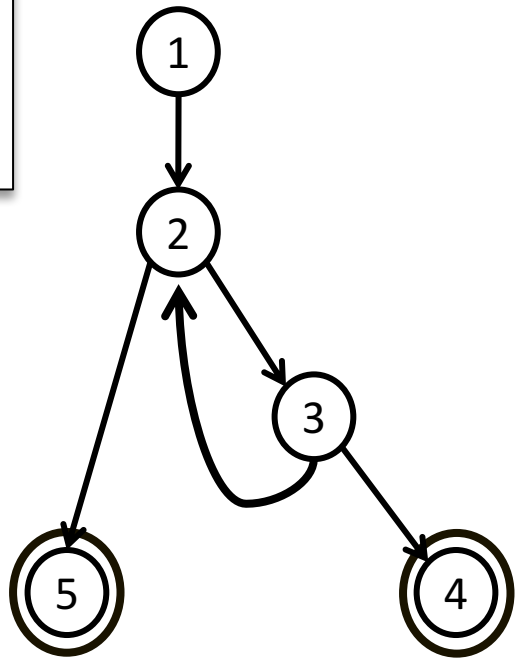
Initial Node: 1

Final Nodes: 4, 5

Test Paths

[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]

Graph Abstraction



Illustrative Example

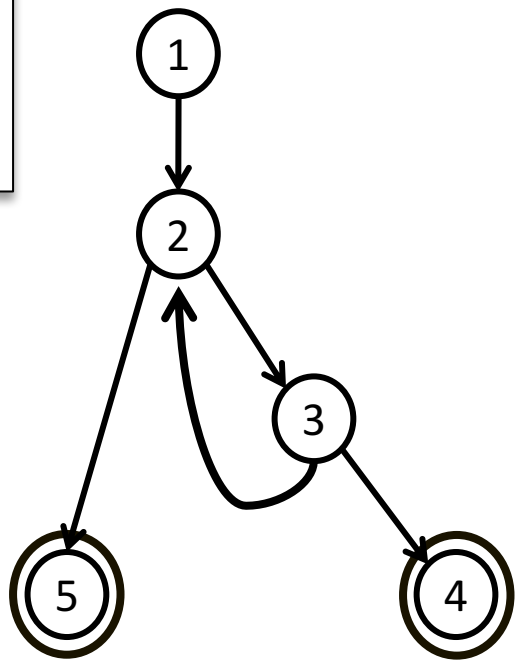
Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

Edge-Pair Coverage
➔ **6 Test Requirements**
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]

Values???

Graph Abstraction



Illustrative Example

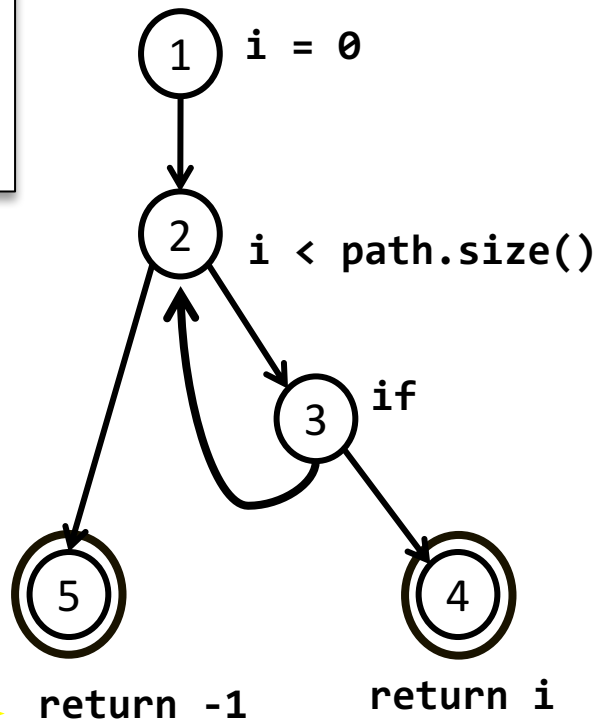
Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

Edge-Pair Coverage
➔ **6 Test Requirements**
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]

Values???

Control Flow Graph



and one more …

- Consider a simple e-signature client application.
  - When you logged in, a list of items that require your signature is displayed.
  - You may do nothing and exit the application!
  - You can select an item and view its details: then you may
    1. sign/reject and/or return back to the list, or
    2. print/save the details and exit!

In-Class Exercise #6

- What are the benefits of “model-driven test design” process? Mention few items.
- In what way and how would this process change/affect the traditional software testing? Discuss briefly.

- You have 10 minutes
- Do the exercise individually/in groups (up to 3)
- Upload your answer in Moodle.

Model-Driven Test Design

The **Model-Driven Test Design (MDTD)** process breaks testing into a series of small tasks that simplify test generation. Then test designers isolate their task, and work at a higher level of abstraction by using mathematical engineering structures to design test values independently of the details of software or design artifacts, test automation, and test execution.

Model-Driven Testing