Structure of Lecture 5

• Test Lifecycle
• Test Levels
• Test Tools
• Lab 5
Testing in different Process Types

Waterfall model

- Programmers
- Testers

Agile model(s)

- Tester
- Customer
- Programmer

Idea: Testing in collaboration
V-model

1. Requirements
2. Functional specification
3. Architecture design
4. Module design
5. Coding
6. Integration testing
7. Unit testing
8. System testing
9. Acceptance testing

Build → Test
V-model

**Bad:**

- Document driven
  - Relies on the existence, accuracy, and timeliness of documentation
  - Asserts testing on each level is designed based on the deliverables of a single design phase
- Communicates change poorly
  - Does not show how changes, fixes, and test rounds are handled (rework!)
- Based on simplistic waterfall model (‘Big Bang’)
  - Testing windows get squeezed
  - Difficult to fit into iterative development

**Good:**

- Intuitive and easy to explain
  - Matches the familiar waterfall model
  - Makes a good model for training of people
  - Shows how testing is related to other phases/activities of the waterfall process
Agile development (e.g., SCRUM) = Time is fixed, scope may change

- 30 days to complete iteration or sprint
- 90 days to complete release 1
- 90 days to complete release 2
- 180 days for whole project
Challenges of Testing in Agile Development

• Requirements change all the time
• Specification documents are never final
• Code is never ‘finished’, never ‘ready for testing’
• Limited time to test
• Need for regression testing in each increment
  – Developers always break things
  – How can we trust that the code is not broken?
Approaches to Testing in Agile Development

• Automated regression testing
  – Automated unit testing
  – Test-driven development
  – Daily builds and automated tests

• Stabilisation phase or increment
  – Feature freeze
  – Testing and debugging at the end of the increment or release

• Separate system testing
  – Independent testing
  – Separate testers or test team
A Combined Testing Approach
## Challenges – … and how to cope with them

<table>
<thead>
<tr>
<th>Requirements change all the time</th>
<th>Let them change, test design is part of each task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifications are never final</td>
<td>Focus on developing ‘finished’ increments, tracking on the task level</td>
</tr>
<tr>
<td>Code is never ‘finished’, never ready for testing</td>
<td>Testing is part of each development task</td>
</tr>
<tr>
<td>Not enough time to test</td>
<td>Trust comes from building-in the quality, not from the external testing ‘safety net’</td>
</tr>
<tr>
<td>Need to regression test everything in each increment</td>
<td>Automation is critical</td>
</tr>
<tr>
<td>Developers always break things again</td>
<td>How can we trust?</td>
</tr>
</tbody>
</table>
Structure of Lecture 5

- Test Lifecycle
- Test Levels
- Test Tools
- Lab 5
# Testing Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Definition and Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Testing (AT)</td>
<td>The level of the software testing process where a system is tested for acceptability. The purpose of AT is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.</td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td>The level of the software testing process where a complete, integrated system/software is tested. The purpose of ST is to evaluate the system’s compliance with the specified requirements.</td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td>The level of the software testing process where individual units are combined and tested as a group. The purpose of IT is to expose faults in the interaction between integrated units.</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>The level of the software testing process where individual units/components of a software/system are tested. The purpose of UT is to validate that each unit of the software performs as designed.</td>
</tr>
</tbody>
</table>
## Testing Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Who and How?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>How: Usually, Black Box Testing method is used; often the testing is done ad-hoc and non-scripted</td>
</tr>
<tr>
<td></td>
<td>How: Usually, Black Box Testing method is used</td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td>Who: Either Developers themselves or independent Testers</td>
</tr>
<tr>
<td></td>
<td>How:</td>
</tr>
<tr>
<td></td>
<td>- Any of Black Box, White Box, and Gray Box Testing methods can be used</td>
</tr>
<tr>
<td></td>
<td>- Test drivers and test stubs are used to assist in Integration Testing.</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>Who: Developers</td>
</tr>
<tr>
<td></td>
<td>How:</td>
</tr>
<tr>
<td></td>
<td>- White-Box Testing Method</td>
</tr>
<tr>
<td></td>
<td>- UT frameworks (e.g., jUnit), drivers, stubs, and mock/fake objects are used</td>
</tr>
</tbody>
</table>
Unit Testing

Driver

Classes to be tested

Tool example: JUnit

(c) 2008 Mauro Pezzè & Michal Young
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class JunitTest1 {
    private Collection collection;

    @BeforeClass
    public static void oneTimeSetUp() {
        // one-time initialization code
        System.out.println("@BeforeClass - oneTimeSetUp");
    }

    @AfterClass
    public static void oneTimeTearDown() {
        // one-time cleanup code
        System.out.println("@AfterClass - oneTimeTearDown");
    }

    @Before
    public void setUp() {
        collection = new ArrayList();
        System.out.println("@Before - setUp");
    }

    @After
    public void tearDown() {
        collection.clear();
        System.out.println("@After - tearDown");
    }

    @Test
    public void testEmptyCollection() {
        assertTrue(collection.isEmpty());
        System.out.println("@Test - testEmptyCollection");
    }

    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        assertEquals(1, collection.size());
        System.out.println("@Test - testOneItemCollection");
    }
}
Integration Testing – Procedural

Note: This has nothing to do with class hierarchy in OO (inheritence)
Integration Testing – OO

Fig. 6.8
Integration Testing / Scaffolding

Driver

Classes to be tested

Stubs

Tool example: JUnit

Tool example: JMockIt
System Testing

• Testing the system as a whole
  • Functional
    – Functional requirements and requirements-based testing
  • Non-functional
    – Performance, stress, configuration, security, ...
    – As important as functional requirements
    – Often poorly specified
    – Must be tested
• Often done by independent test group
  – Collaborating developers and testers
Types of System Testing

Fig. 6.10
Acceptance Testing

• **Internal Acceptance Testing** (Also known as **Alpha Testing**) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.

• **External Acceptance Testing** is performed by people who are not employees of the organization that developed the software.
  • **Customer Acceptance Testing** is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software.
  • **User Acceptance Testing** (Also known as **Beta Testing**) is performed by the end users of the software. They can be the customers themselves or the customers’ customers.
## Test Level vs Programming Paradigm

<table>
<thead>
<tr>
<th>Level</th>
<th>OO</th>
<th>Procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit</td>
<td>Method</td>
<td>Function or procedure</td>
</tr>
<tr>
<td>Module</td>
<td>Class</td>
<td>Group of functions or procedures</td>
</tr>
<tr>
<td>Integration</td>
<td>Cluster of classes</td>
<td>Subsystem</td>
</tr>
<tr>
<td>System</td>
<td>System</td>
<td>System</td>
</tr>
</tbody>
</table>
# Test Level vs Programming Paradigm

<table>
<thead>
<tr>
<th>Level</th>
<th>OO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit</td>
<td>Method</td>
</tr>
<tr>
<td>Module</td>
<td>Class</td>
</tr>
<tr>
<td>Integration</td>
<td>Cluster of classes</td>
</tr>
<tr>
<td>System</td>
<td>System</td>
</tr>
</tbody>
</table>

Today often called:
- Intra-method testing
- Inter-method testing
- Intra-class (Class) testing
- Inter-class testing
OO-Testing Approaches

Intra-Class Testing
- Super/subclass relations
- State machine testing
- Augmented state machine
- Data flow model
- Exceptions
- Polymorphic binding

Inter-Class Testing
- Hierarchy of clusters
- Functional cluster testing
- Data flow model
- Exceptions
- Polymorphic binding

System and Acceptance Testing (unchanged)
Intra-class State Machine Testing

Basic idea:

• The state of an object is modified by operations
• Methods can be modeled as state transitions
• Test cases are sequences of method calls that traverse the state machine model

• State machine model can be derived from specification (functional testing), code (structural testing), or both
Class Testing and State

In addition, if it is decided that the class is the smallest component to test, testers must decide if they are able to adequately cover all necessary features of each method in class testing. Some researchers believe that coverage objectives and test data need to be developed for each of the methods, for example, the `create`, `pop`, `push`, `empty`, and `show_top` methods associated with the stack class shown in Figure 6.3. Other researchers believe that a class can be adequately tested as a whole by observation of method interactions using a sequence of calls to the member functions with appropriate parameters.

Again, referring to the stack class shown in Figure 6.3, the methods `push`, `pop`, `full`, `empty`, and `show_top` will either read or modify the state of the stack. When testers unit (or component) test this class what they will need to focus on is the operation of each of the methods in the class and the interactions between them. Testers will want to determine, for example, if `push` places an item in the correct position at the top of the stack. However, a call to the method `full` may have to be made first to determine if the stack is already full. Testers will also want to determine if `push` and `pop` work together properly so that the stack pointer is in the correct position after a sequence of calls to these methods. To properly test this class, a sequence of calls to the methods needs to be specified as

```
push(s, elem1)
Show_top(s) -> elem1
pop(s, x) -> x=elem1
pop(s, x) -> x=?
```

The result of a method call depends on the state of the stack.

**Fig. 6.3**
Sample stack class with multiple methods.
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length)
            resize();
        if (size < values.length)
            values[size++] = x;
    }

    int pop() {
        if (size > 0)
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

- `push(elem1)`
- `pop() -> elem1`
- `pop() -> exception`
- `push(elem1)`
- `push(elem4) -> resize()`
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;
    void push(int x) {
        if(size >= values.length)
            resize();
        if(size < values.length)
            values[size++] = x;
    }
    int pop() {
        if(size > 0)
            return values[size--];
        else
            throw new EmptyStackException();
    }
    private void resize() {
        int[] tmp = new int[values.length * 2];
        for(int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

How many test cases needed to cover all feasible branches?
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length)
            resize();
        if (size < values.length)
            values[size++] = x;
    }

    int pop() {
        if (size > 0)
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

How many test cases needed to cover all feasible branches?

4 decisions
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) { // Requires a full stack
        if (size >= values.length)
            resize();
        if (size < values.length) // Else branch is infeasible
            values[size++] = x;
    }

    int pop() { // May imply coverage in push and resize
        if (size > 0)
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize(){
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

How many test cases needed to cover all feasible branches?

4 decisions
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) // Requires a full stack
            resize();
        if (size < values.length) // Else branch is infeasible
            values[size++] = x;
    }

    int pop() {
        if (size > 0) // May imply coverage in push and resize
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

2 test cases are enough:

```java
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}
```

Stack stack0 = new Stack();
```java
int int0 = -510;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
```
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if(size >= values.length) ← Requires a full stack
            resize();
        if(size < values.length) ← Else branch is infeasible
            values[size++] = x;
    }

    int pop() {
        if(size > 0) ← May imply coverage in push and resize
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize(){
        int[] tmp = new int[values.length * 2];
        for(int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

2 test cases are enough:

D1: Stack stack0 = new Stack();
    int int0 = -510;
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.push(int0);
    stack0.pop();
}
```
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) \[Requires a full stack\]
            resize();
        if (size < values.length) \[Else branch is infeasible\]
            values[size++] = x;
    }

    int pop() {
        if (size > 0) \[May imply coverage in push and resize\]
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

2 test cases are enough:

D1

D2

D3: false
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) \textit{Requires a full stack}
            resize();
        if (size < values.length) \textit{Else branch is infeasible}
            values[size++] = x;
    }

    int pop() {
        if (size > 0) \textit{May imply coverage in push and resize}
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

2 test cases are enough:

<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Stack stack0 = new Stack();
int int0 = -510;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
Stack Example

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) // Requires a full stack
            resize();
        else // Else branch is infeasible
            values[size++] = x;
    }

    int pop() {
        if (size > 0) // May imply coverage in push and resize
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

What is the McCabe Complexity of this program?
public class Stack {
  int[] values = new int[3];
  int size = 0;
  void push(int x) {
    if (size >= values.length) \[Requires a full stack\]
      resize();
    if (size < values.length) \[Else branch is infeasible\]
      values[size++] = x;
  }
  int pop() {
    if (size > 0) \[May imply coverage in push and resize\]
      return values[size--];
    else
      throw new EmptyStackException();
  }
  private void resize() {
    int[] tmp = new int[values.length * 2];
    for (int i = 0; i < values.length; i++)
      tmp[i] = values[i];
    values = tmp;
  }
}
Stack Example – CFGs

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void Push()
    if (true) { if (true) { if (false) { return; } } }
    else return;

    int Pop()
    if (false) { return 0; }
    else return 1;

    int Resize()
    if (true) { return 1; }
    else return 2;

    McCabe:
    Push(): 2+1=3 or 7-6+2*1=3
    Pop(): 1+1=2 or 5-5+2*1=2
    Resize(): 1+1=2 or 4-4+2*1=2

    Total: 3+2+2=7 or 16-15+2*3=7

    2nd version:
    #edges-#nodes+2*#programs
    (program==method)
```

**Stack Example – CFGs**

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void if()
        if (true)
            true
        else
            false
   ()
    void for()
        for (true)
            true
        false
   ()

    int push()
        Push(): 2+1=3 or 7-6+2*1=3
   ()

    int pop()
        Pop(): 1+1=2 or 5-5+2*1=2
   ()

    int resize()
        Resize(): 1+1=2 or 4-4+2*1=2
   ()

    int total()
        Total: 3+2=2 or 16-15+2*3=7
   ()

    2nd version:
    edges-nodes+2*programs
    (program==method)
```

**McCabe:**
- Push(): 2+1=3 or 7-6+2*1=3
- Pop(): 1+1=2 or 5-5+2*1=2
- Resize(): 1+1=2 or 4-4+2*1=2
- Total: 3+2=2 or 16-15+2*3=7
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) \[Requires a full stack\]
            resize();
        if (size < values.length) \[Else branch is infeasible\]
            values[size++] = x;
    }

    int pop() {
        if (size > 0) \[May imply coverage in push and resize\]
            return values[size--];
        else
            throw new EmptyStackException();
    }

    private void resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
Stack Example – CFG (Class)

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length) // Requires a full stack
        if 
        else 
            throw new EmptyStackException();
    }

    int[] tmp = new int[values.length * 2];
    for (int i = 0; i < values.length; i++)
        tmp[i] = values[i];

    values = tmp;
}

private void resize()
```
Stack Example – CFG (Class)

```java
public class Stack {
    int[] values = new int[3];
    int size = 0;

    void push(int x) {
        if (size >= values.length)
            Requires a full stack
        else
            throw new EmptyStackException();
    }

    int resize() {
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }
}
```

McCabe:
24 - 19 + 2 * 1 = 7

#edges - #nodes + 2 * #programs (program == class)
State Diagram for Stack Example

- **non_exist**
- **Stack()**
- **empty** (size = 0)
- **pop()**
- **error**

Pushing an element:
- [size < length] push(x) / size++
- [size = length] push(x) / resize(); size++

Popping an element:
- [size = 1] pop() / size--
- [size > 1] pop() / size--

**Note:** resize() is a private method
length := length*2
Stack stack0 = new Stack();
try {
    stack0.pop();
} catch (EmptyStackException e) {
}

---

**TC1**

- **non_exist**
  - Stack()
  - empty (size = 0) → pop()
    - not_empty (0 < size <= length)
      - [size < length] push(x) / size++
      - [size = 1] pop() / size--
      - [size > 1] pop() / size--
      - [size = length] push(x) /
        resize(); size++
  - error

**Note:** resize() is a private method
  * length := length*2
TC2

non_exist Stack() empty (size = 0) pop() error

[size < length] push(x) / size++

[size = 1] pop() / size--

[size > 1] pop() / size--

[size = length] push(x) /
resize(); size++

Note: resize() is a private method
length := length*2

Stack stack0 = new Stack();
int int0 = -510;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
**TC3**

(to cover all state-transitions)

- **non_exist**
  - Stack()
- **empty** (size = 0)
  - pop()
  - **not_empty** (0 < size <= length)
  - push(x)
  - [size = 1] pop() / size--
  - [size > 1] pop() / size--
  - [size = length] push(x) / resize(); size++

Note: *resize() is a private method*

length := length*2

---

Stack stack0 = new Stack();
int int0 = -510;
stack0.push(int0);
stack0.push(int0);
stack0.push(int0);
stack0.pop();
Structure of Lecture 5

• Test Lifecycle
• Test Levels
• Test Tools
• Lab 5
Tools – the Workbench

• Good for
  – repeating tasks
  – organising data

• Requires training

• Must be introduced incrementally

• No “silver bullet”

Evaluation criteria

• Ease of use
• Power
• Robustness
• Functionality
• Ease of introduction
• Quality of support
• Cost
• Company policies and goals
Test Tools – in the Process

- Requirement specification
- Test design tools
  - Architectural design
  - Detailed design
  - Static analysis tools
  - Code
  - Test management tools
- Test execution and comparison tools
  - Unit test
  - Coverage tools
- Performance simulator tools
  - Integration test
  - System test
- Acceptance test
  - Dynamic analysis tools
  - Debugging tools
- System test
  - Acceptance test
There is no shortage of Test Tools

- Defect Tracking (98)
- GUI Test Drivers (71)
- Load and Performance (52)
- Static Analysis (38)
- Test Coverage (22)
- Test Design Tools (24)
- Test Drivers (17)
- Test Implementation (35)
  - assist with testing at runtime - memory leak checkers, comparators, and a wide variety of others
- Test case Management (24)
- Unit Test Tools (63)
- 3 different categories of others

Other links to test tool overviews:
- http://www.aptest.com/resources.html

From http://www.testingfaqs.org/
Tools categories by purpose

1. Reviews and inspections
2. Test planning
3. Test design and development
4. Test execution and evaluation
5. Test support
Tools categories by purpose

1. Reviews and inspections
   • Complexity analysis
     – Identify problem areas
   • Code comprehension
     – Show different views of the artefact
   • Syntax and semantics analysis
     – Check and warn

2. Test planning

3. Test design and development

4. Test execution and evaluation

5. Test support
Tools categories by purpose

1. Reviews and inspections
2. Test planning
3. Test design and development
4. Test execution and evaluation
5. Test support

• Templates for test plan documentation
• Test schedule and staffing estimates
• Complexity analyser

To large extent: general project management tools
Tools categories by purpose

1. Reviews and inspections
   • Test data generator
   • Requirements-based test design tool
2. Test planning
   • Capture/replay
   • Coverage analysis
3. Test design and development
4. Test execution and evaluation
   Often integrated with test execution tools
5. Test support
Tools categories by purpose

1. Reviews and inspections
2. Test planning
3. Test design and development
4. Test execution and evaluation
5. Test support

- Test case management
- Capture/replay
- Coverage analysis
- Memory testing (leaks)
- Simulators and performance analysis
  - HW emulators
  - SW simulators (mocking)
  - Load generators
Tools categories by purpose

1. Reviews and inspections
2. Test planning
3. Test design and development
4. Test execution and evaluation
5. Test support

- Issue reporting
  - Report, Dispatch, Follow-up

- Configuration management
  - Manage, control and coordinate changes
Functional (Web-)Testing Approaches

1. Recorded Scripts

2. Engineered Scripts

3. Data-driven Testing

4. Keyword-driven Testing

5. Model-based Testing
Recorded Scripts

- Unstructured
- Scripts generated using capture and replay tools
- Relatively quick to set up
- Mostly used for regression testing
- Scripts non-maintainable, in practice
  - If the system changes they need to be captured again

Capture Replay Tools
- Record user’s actions to a script (keyboard, mouse)
  - Tool specific scripting language
- Scripts access the (user) interface of the software
  - Input fields, buttons and other widgets
- Simple checks can be created in the scripts
  - Existence of texts and objects in the UI
  - Data of GUI objects
Recorded Scripts

- Example with Selenium IDE

Some web-application to be tested ...

Record Button switched on ...

http://opensource.demo.orangehrm.com
Recorded Scripts

- Example with Selenium IDE

Some web-application to be tested...

Make sure Record button is ON!
Open Base URL in browser
Login using values:
  - Login Name: demo
  - Password: demo
Click ‘Login’ button
Recorded Scripts

• Example with Selenium IDE

... next actions ...

Highlight ‘Welcome demo’ text
Verify that text is present
- command: VerifyTextPresent
Click ‘Logout’

... then stop recording ...
Recorded Scripts

- Example with Selenium IDE

**Test Case**

(`=Test Scenario`)

**consists of several**

**Test Steps**

- Open Base URL in browser
- Login using values:
  - Login Name: demo
  - Password: demo
- Click Login button
- Highlight ‘Welcome demo’ text
- Verify that text is present
- Click ‘Logout’

TCs can be saved and exported into several programming languages (java, python, c#, etc.)

Selenium commands

Data (values)

Location on Web-page (Target) may use xpath, css, id, field, etc.
Engineered Scripts

- Scripts are well-designed (following a systematic approach), modular, robust, documented, and maintainable
- Separation of common tasks
  - E.g. setup, cleanup/teardown, and defect detection
- Test data is still embedded into the scripts
  - One driver script per test case
- Test code is mostly written manually
- Implementation and maintenance require programming skills which testers (test engineers) might not have
- “Just like any other software development project”
Engineered Scripts – Example
Engineered Scripts – Example

Click on 'Math Calculators'
Engineered Scripts – Example

We got a list ‘Math Calculators’ and ... clicked on ‘Percent Calculator’
Engineered Scripts – Example

We enter ‘10’ and ‘50’
Engineered Scripts – Example

We click ‘Calculate’ and get the result (‘5’).
How do we test this with ‘Engineered Scripts’?

(this will be done in-depth in Lab 5)
Engineered Scripts

• Example with Selenium WebDriver

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class TestNGClass {
    WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchapp() {
        // Puts an implicit wait, will wait for 10 seconds before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.navigate().to("http://www.calculator.net");
        driver.manage().window().maximize();
    }

    @Test
    public void calculatepercent() {
        // Click on Math Calculators
        driver.findElement(By.xpath("//[@id='menu']//div[3]/a")).click();
        // Click on Percent Calculators
        driver.findElement(By.xpath("//[@id='menu']//div[4]/div[3]/a")).click();
        // Enter value 10 in the first number of the percent calculator
        driver.findElement(By.id("cpa1")).sendKeys("10");
        // Enter value 50 in the second number of the percent calculator
        driver.findElement(By.id("cpa2")).sendKeys("50");
        // Click Calculate Button
        driver.findElement(By.xpath("//[@id='content']//table//tbody/tr//td[2]/input")).click();
        // Get the Result Text based on its xpath
        String result = driver.findElement(By.xpath("//[@id='content']//p[2]/span/font/b")).getText();
        // Print a Log In message to the screen
        System.out.println("The Result is " + result);
        if(result.equals("5")) {
            System.out.println("The Result is Pass");
        } else {
            System.out.println("The Result is Fail");
        }
    }

    @AfterTest
    public void terminateTest() {
        driver.close();
    }
}
```
Engineered Scripts

- Example with Selenium WebDriver
  - Click on 'Math Calculator'

-xpath notation

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.By;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.By.

public class TestNGClass {
    WebDriver driver = new FirefoxDriver();

    BeforeTest
    public void launchapp() {
        // Puts an implicit wait, will wait for 10 seconds before throwing an exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.get("http://www.c4gif.com");
        driver.manage().window().maximize();
    }

given Test
    public void calculatepercent() {
        // Click on Math Calculators
        driver.findElement(By.xpath("//a[@id='menu']/div[3]/a")).
            click();

        // Click on Percent Calculators
        driver.findElement(By.xpath("//a[@id='menu']/div[4]/div[3]/a")).
            click();

        // Enter value 10 in the first number of the percent calculator
        driver.findElement(By.id("cpart1")).sendKeys("10");

        // Enter value 50 in the second number of the percent calculator
        driver.findElement(By.id("cpart2")).sendKeys("50");

        // Click Calculate Button
        driver.findElement(By.xpath("//a[@id='content']/table/tbody/tr/th[2]/input")).
            click();

        // On its xpath
        WebElement element = driver.findElement(By.xpath("//a[@id='content']"));
        String result = element.getText();

        if (result.equals("Pass")) {
            System.out.println("Result is Pass");
        } else {
            System.out.println("Result is Fail");
        }
}
```
Engineered Scripts

- Example with Selenium WebDriver

Click on 'Percent Calculator'

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By; import org.openqa.selenium.FirefoxDriver;
import org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver; public class TestNGClass {
    WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchapp() {
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://www.calculate.net/");
        driver.manage().window().maximize();
    }

    @Test
    public void calculatepercent() {
        driver.findElement(By.xpath("//[@id='menu']/div[3]/a")).click();
        // Click on Math Calculators
        driver.findElement(By.xpath("//[@id='menu']/div[4]/div[3]/a")).click();
        // Enter value 50 in the first number of the percent Calculator
        driver.findElement(By.id("cpart1")).sendKeys("50");
        // Enter value 50 in the second number of the percent Calculator
        driver.findElement(By.id("cpart2")).sendKeys("50");
        // Click Calculate Button
        driver.findElement(By.xpath("//[@id='content']/tbody/tr/td[2]/input")).click();
        // Get the Result Text based on its xpath
        String result = driver.findElement(By.xpath("//[@id='content']//p[2]/span/font/b")).getText();
        // Print a Log In message to the screen
        System.out.println("The Result is " + result);
        if(result.equals("55"))
            System.out.println("The Result is Pass");
        else
            System.out.println("The Result is Fail");
    }

    @AfterTest
    public void terminateTest() {
        driver.close();
    }
}
```
Engineered Scripts

- Example with Selenium WebDriver

Enter ‘10’ – first number
Enter ‘50’ – second number

Click on ‘Calculate’
Get result with getText() – ‘5’
Engineered Scripts

- Example with Selenium WebDriver

Checks if result equals '5' (10% of 50)
Data-Driven Testing

• Data-Driven Testing = Executing the same set of test steps with multiple (different) data

  – Test inputs and expected outcomes stored as data
   – Typically in tabular format

• Test data are read from an external data source
• One driver script can execute all of the designed test cases

Note that in previous example test data (‘10’ and ‘50’) was embedded in the test case definition
public class DataProviderExample extends SeleneseTestCase{

    @BeforeClass
    public void setUp() throws Exception {
        ...
    }

    @DataProvider(name = "DP1")
    public Object[][] createData1() throws Exception{
        Object[][] retObjArr=getTableArray("test/Resources/Data/data1.xls", "DataPool", "imdbTestData1");
        return(retObjArr);
    }

    @Test (dataProvider = "DP1")
    public void testDataProviderExample(String movieTitle, String directorName, String moviePlot, String actorName) throws Exception
    {
        //enter the movie title
        selenium.type("q", movieTitle);
        //they keep switching the go button to keep the bots away
        if (selenium.isElementPresent("nb15go_image"))
            selenium.click("nb15go_image");
        else
            selenium.click("xpath=/descendant::button[@type='submit']");

        selenium.waitForPageToLoad("30000");
        //click on the movie title in the search result page
        selenium.click("xpath=/descendant::a[text()="'+movieTitle+'"]");
        selenium.waitForPageToLoad("30000");
        //verify director name is present in the movie details page
        verifyTrue(selenium.isTextPresent(directorName));
        //verify movie plot is present in the movie details page
        verifyTrue(selenium.isTextPresent(moviePlot));
        //verify movie actor name is present in the movie details page
        verifyTrue(selenium.isTextPresent(actorName));
    }

    @AfterClass
    ...
}

Defines where to find the data (uses Java Excel API)
Data-Driven Testing

Reads the data from the data table ...
(method getTableArray(...))

```java
public String[][] getTableArray(String xlFilePath, String sheetName, String tableName){
    String[][] tabArray=null;
    try{
        Workbook workbook = Workbook.getWorkbook(new File(xlFilePath));
        Sheet sheet = workbook.getSheet(sheetName);
        int startRow,startCol, endRow, endCol,ci,cj;
        Cell tableStart=sheet.findCell(tableName);
        startRow=tableStart.getRow();
        startCol=tableStart.getColumn();
        Cell tableEnd= sheet.findCell(tableName, startCol+1,startRow+1, 100, 64000, false);
        endRow=tableEnd.getRow();
        endCol=tableEnd.getColumn();
        System.out.println("startRow=",startRow," endRow=",endRow," startCol=",startCol," endCol=",endCol);
        tabArray=new String[endRow-startRow-1][endCol-startCol-1];
        ci=0;
        for (int i=startRow+1;i<endRow;i++,ci++){
            cj=0;
            for (int j=startCol+1;j<endCol;j++,cj++){
                tabArray[ci][cj]=sheet.getCell(j,i).getContents();
            }
        }
    }
    catch (Exception e)    {
        System.out.println("error in getTableArray()");
    }
    return(tabArray);
}
```

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>Director/Name</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Birds</td>
<td>Alfred Hitchcock</td>
<td>Hawaii returns to America and attempts to make contact with disgraced FBI agent stabbed and survives a vengeful vigilante. Rod Taylor</td>
</tr>
<tr>
<td>Pan's Labyrinth</td>
<td>Guillermo del Toro</td>
<td>A boy is forced to travel to a curiously sunken land where his destiny unfolds.</td>
</tr>
<tr>
<td>The Bourne Ultimatum</td>
<td>Paul Greengrass</td>
<td>Bourne dodges new, superior assassins as he searches for his unknown past while a government agent tries to track him down. Matt Damon</td>
</tr>
<tr>
<td>Point Break</td>
<td>Kathryn Bigelow</td>
<td>An FBI agent goes undercover to catch a gang of bank robbers who may be terrorists.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>London</td>
</tr>
<tr>
<td>John</td>
<td>New York</td>
</tr>
<tr>
<td>Mary</td>
<td>Paris</td>
</tr>
<tr>
<td>Kate</td>
<td>LA</td>
</tr>
<tr>
<td>George</td>
<td>Washington</td>
</tr>
</tbody>
</table>

MTAT.03.159 / Lecture 05 / © Dietmar Pfahl 2018
Data-Driven Testing

• External test data can be edited without programming skills
  Test design and framework implementation are now separate tasks:
  – design task can be given to someone with the domain knowledge
    (business people, customers) and
  – framework implementation to someone with programming skills.

• Avoids the problems of embedded test data
  – Data are hard to understand in the middle of all scripting details
  – Updating tests or creating similar tests with slightly different test data
    types/structures always requires programming (-> copy-paste scripting)

• Follow this link for a fully worked example of Data-Driven Testing with Selenium:
Data-Driven Testing


```python
data = open('testdata.tsv').read()
lines = data.splitlines()[1:]  # [1:] excludes the header row

for line in lines:
    testId, number1, operator, number2, expected = line.split('	')
    # Actual test functionality excluded
```
Keyword-Driven Testing

Keyword-driven test data file adds one level of abstraction

Similar to data-driven testing
But instead of testing functions directly, handler looks for keywords and based on that derives required test-data
Purpose: increase flexibility and re-usability
Keyword-Driven Testing

Several layers of keywords possible
Benefit: can define new keywords using existing ones
Model-based Testing

• System under test is modelled
  – UML-state machines, domain specific languages (DSL)
• Test cases are automatically generated from the model
  – The model can provide also the expected results for the generated test cases
  – More accurate model -> better test cases
• Generate a large amount of tests that cover the model
  – Many different criteria for covering the model
  – Execution time of test cases might be a factor
• Challenges:
  – Personnel competencies
  – Data-intensive systems (cannot be modelled as a state-machine)
• Simple MBT tool http://graphwalker.org/
GraphWalker  [http://graphwalker.org/](http://graphwalker.org/)

Example:
A regression test for the login function of the Spotify Desktop Client

The feature is supposed to work like this:

- In a freshly installed client, and the client is started, the Login dialog is expected to be displayed.
- The user enters valid credentials and the browse view is expected to start.
- If the user quits, or logs out, the Login dialog is displayed once again.
- If the user checks the Remember Me checkbox, and logs in (using valid credentials), the client starts, and, next time the user starts the client, it will start without asking the client for credentials.
**Basic Flow**

For testing the 2 first steps, a model would look something like this:

1. The **Start** vertex is where the tests start.
2. In **e_Init**, we remove all cache, and kill any previous client processes.
3. **v_ClientNotRunning** will assert that there is no Spotify client process running.
4. **e_Start** starts the client.
5. **v_LoginPrompted** asserts that the login dialog is displayed and correctly rendered.
6. **e_ValidPremiumCredentials** enters a valid username and password and clicks the Sign In button.
7. **v_Browse** asserts that the Browse view is correctly displayed.

**Example:**
A regression test for the login function of the Spotify Desktop Client

**GraphWalker** [http://graphwalker.org/](http://graphwalker.org/)
GraphWalker [http://graphwalker.org/]

Example:
A regression test for the login function of the Spotify Desktop Client

Complete Model (all steps)
GraphWalker  [http://graphwalker.org/]

Example:
A regression test for the login function of the Spotify Desktop Client

```%> java -jar graphwalker.jar offline -m Login.graphml
"random(edge_coverage(100))"
```

```
e_Init
v_ClientNotRunning
e_StartClient
[rememberMe=!!validLogin]

v_LoginPrompted
e_ToggleRememberMe
[rememberMe=!rememberMe]

v_Browse
e_ValidPremiumCredentials
[validLogin=true]
e_Logout
e_InvalidCredentials
[validLogin=false]

v_ClientNotRunning
e_Close
e_Init
[rememberMe=false; validLogin=true]
```

...
Example:
A regression test for the login function of the Spotify Desktop Client

This command generates a random test sequence achieving 100% edge (branch) coverage.

%> java -jar graphwalker.jar offline -m Login.graphml "random(edge_coverage(100))"

e_Init
v_ClientNotRunning
e_StartClient
v_LoginPrompted
e_InvalidCredentials
v_LoginPrompted
e_ValidPremiumCredentials
v_Browse
e_Logout
v_LoginPrompted
e_Close
v_ClientNotRunning
...

GraphWalker [http://graphwalker.org/](http://graphwalker.org/)
Functional Testing Approaches

1. Recorded Scripts
   – Cheap to set up, quick & dirty

2. Engineered Scripts
   – Structured

3. Data-driven Testing
   – Data separation

4. Keyword-driven Testing
   – Action separation, DSL

5. Model-based Testing
   – Modeling & Automatic test case generation
What can be automated?

- Test generation
  - Test case (steps with data & oracle)
  - Test data (input)
  - Test oracle (expected output)
  - Test verdict (PASS/FAIL decision)
- Test execution
  - E.g., regression testing
- Test reporting
- Debugging
  - Fault localisation (using failure/error information)
What can be automated?

- Test generation
  - Test case
  - Test data (input)
  - Test oracle (expected output)
- Test execution
  - E.g., regression testing
- Test selection
- Test reporting
- Debugging
  - Fault localisation

Often, people mean automated test execution when they talk about test automation.
When to automate?

- Test Automation should be used by considering the following:
  - Large and critical projects
  - Projects that require testing the same areas frequently
  - Requirements not changing frequently
  - Accessing the application for load and performance with many virtual users
  - Stable software
  - Availability of time/effort (for set-up, execution, maintenance, etc.)
Test automation promises

1. Efficient regression test
2. Run tests more often
3. Perform difficult tests (e.g. load, outcome check)
4. Better use of resources
5. Consistency and repeatability
6. Reuse of tests
7. Earlier time to market
8. Increased confidence
Common problems

1. Unrealistic expectations
2. Poor testing practice
   “Automatic chaos just gives faster chaos”
3. Expected effectiveness
4. False sense of security
5. Maintenance of automatic tests
6. Technical problems (e.g. Interoperability)
7. Organizational issues
Limits of automated testing

• Does not replace manual testing
• Manual tests find more defects than automated tests
  – Does not improve effectiveness
• Greater reliance on quality of tests
  – Oracle problem
• Test automation may limit the software development
  – Costs of maintaining automated tests
Structure of Lecture 5

• Test Lifecycle
• Test Levels
• Test Tools
• Lab 5
Lab 5 – Automated Web App Testing

Lab 5 (week 33: Apr 17 – Apr 18) – Automated Web Application Testing (10%)

Lab 5 Instructions & Tools

Submission Deadlines:

- Tuesday Labs: Monday, 23 Apr, 23:59
- Wednesday Labs: Tuesday, 24 Apr, 23:59

- Penalties apply for late delivery: 50% penalty, if submitted up to 24 hours late; 100 penalty, if submitted more than 24 hours late
Lab 5 – Automated Web App Testing

Lab 5 (week 33: Apr 17 – Apr 18) – Automated Web Application Testing (10%)

Lab 5 Instructions & Tools

Instructions

Web Application

Selenium IDE

Test Code

Find defects

Add test code (positive & negative)
Recommended Textbook Exercises

• Chapter 6
  – 1, 3, 6, 7, 8, 9, 12
Next Weeks

• Lab 5:
  – Automated Web Application Testing (Selenium)

• Lecture 6:
  – Static Analysis (Symbolic Execution, Inspection & Review) and Defect Estimation
  – Test Documentation, Organisation and Process Improvement

• In addition to do:
  – Read textbook chapters