Lectures (J. Liivi 2-111)

• Lecture 1 (14.02) – Introduction to Software Testing
• Lecture 2 (21.02) – Basic Black-Box Testing Techniques
• Lecture 3 (28.02) – BBT advanced: Combinatorial Testing
• Lecture 4 (07.03) – Basic White-Box Testing Techniques
• Lecture 5 (14.03) – Test Lifecycle, Test Tools, Test Automation
• Lecture 6 (21.03) – Test Levels / Behavioural Testing / GUI Testing / Visual Testing
• Lecture 7 (28.03) – BBT advanced: State-Transition Testing & Exploratory Testing
• Lecture 8 (04.04) – BBT advanced: Usability Testing and A/B Testing
• Lecture 9 (11.04) – WBT advanced: Data-Flow Testing / Mutation Testing
• Lecture 10 (18.04) – WBT advanced: Symbolic Execution, Static Code Analysis, Document Inspections, Code Review
• Lecture 11 (25.04) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
• 02.05 - no lecture
• Lecture 12 (09.05) – Industry Guest Lecture (to be announced)
• Lecture 13 (16.05) – Exam Preparation
Structure of Lecture 5

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

Waterfall model

- Programmers
- Testers

Agile model(s)

- Tester
- Customer
- Programmer

Idea: Testing in collaboration
V-model

- Requirements
  - Functional specification
  - Architecture design
  - Module design
    - Coding
- Integration testing
- Unit testing
- System testing
- Acceptance testing

Build

Test
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
V-model

Test Levels

- Acceptance testing
- System testing
- Integration testing
- Unit testing
- System testing
- Test Levels

- AT
- ST
- IT
- UT

Build

- Requirements
- Functional specification
- Architecture design
- Module design
- Coding

LTAT.05.006 / Lecture 05 / © Dietmar Pfahl 2019
Testing OO Code

- Unit (intra-class) tests
- Integration (inter-class) tests
- System tests
- Acceptance (Validation) tests
## OO-Testing Approaches

### Intra-Class Testing

<table>
<thead>
<tr>
<th>Super/subclass relations</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>State machine testing</td>
<td></td>
</tr>
</tbody>
</table>

| Augmented state machine  | Structural |
| Data flow model          |            |
| Exceptions               |            |
| Polymorphic binding      |            |

### Inter-Class Testing

<table>
<thead>
<tr>
<th>Hierarchy of clusters</th>
<th>Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional cluster testing</td>
<td></td>
</tr>
</tbody>
</table>

| Data flow model          | Structural |
| Exceptions               |            |
| Polymorphic binding      |            |

### System and Acceptance Testing (unchanged)
Intra-Class Testing Example

State Machine Testing:

• The state of an object is modified by operations (methods)
• Methods can be modeled as state transitions

• State machine model can be derived from specification (BBT), code (WBT), or both
• Test cases are sequences of method calls that traverse the state machine model
• In addition: Cover the CFG
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, full, 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

```
Stack Class

Data for Stack
::

Member functions
for Stack
create(s, size)
push(s, item)
pop(s, item)
full(s)
empty(s)
show_top(s)
```

*Fig. 6.3*

*Sample stack class with multiple methods.*

```
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*
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?

Think of test cases as sequences of method calls.
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) {
        if(size >= values.length) \( \text{Requires a full stack} \)
            resize();
        if(size < values.length) \( \text{Else branch is infeasible} \)
            values[size++] = x;
    }

    int pop() {
        if(size > 0) \( \text{May imply coverage in push and resize} \)
            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;
    }
}
```

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:

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

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

D4

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

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

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

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

    int pop() {
        if (size > 0) \(\Rightarrow\) 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: \(\Rightarrow\) Requires a full stack
D2: \(\Rightarrow\) Else branch is infeasible
D3: \(\Rightarrow\) May imply coverage in push and resize
D4: 

D3: false
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: false
D2: true
D1: false
D4: true ... false
D3: true
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;
    }
}
```

What is the McCabe Complexity of this program?
Stack Example – CFGs

```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;
    }
}
```

---

**UNIVERSITY OF TARTU**

**INSTITUTE OF COMPUTER SCIENCES**
Stack Example – CFGs

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

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

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

    int push() {
        int p = values[size];
        tmp[i] = values[i];
        values = tmp;
        return p;
    }

    int pop() {
        return values[size--];
    }

    void resize() {
        values = new int[size];
    }

    // McCabe metrics
    int push() {
        return values[0];
    }
    int pop() {
        return values[0];
    }
    int resize() {
        return values[0];
    }

    // Total McCabe metrics
    int total() {
        return 3 * (push() + pop() + resize());
    }

    // 2nd version
    int total2() {
        return 2 * (push() + pop() + resize());
    }
}
```

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
            false
            true
            false
        if
            true
            false
        else
            true
            false
    }

    int resize()
        if
            true
            false
            true
            false
        else
            true
            false
    }

    if
        true
        false
for
    true
    false
```

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 – CFG (Class)

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

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

    int pop() {
        if (size > 0) \(\text{May imply coverage in push and resize}\)
            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
            throw new EmptyStackException();
        }
        if (true) {
        }
        int[] tmp = new int[values.length * 2];
        for (int i = 0; i < values.length; i++)
            tmp[i] = values[i];
        values = tmp;
    }

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

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

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

    if (true) {
        return values[size--];
    } else {
        throw new EmptyStackException();
    }
}
```

McCabe: 24-19+2*1=7

#edges-#nodes+2*#programs (program==class)
public class Stack {
  int[] values = new int[3];
  int size = 0;
  void push(int x) {
    if (size >= values.length) \(\leftarrow\) Requires a full stack
      if (true)
        if (false)
          if (true)
            resize()
    else
      throw new EmptyStackException();
  }
  int size() {
    if (true)
      if (false)
        resize()
    else
      if (true)
        if (false)
          if (true)
            resize()
    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;
  }
}

McCabe: 24 - 19 + 2 * 1 = 7
#edges - #nodes + 2 * #programs
(program == class)
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>Challenges</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Requirements change all the time</td>
<td>• Let them change, test design is part of each task</td>
</tr>
<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></td>
</tr>
<tr>
<td>• How can we trust?</td>
<td></td>
</tr>
</tbody>
</table>
Structure of Lecture 5

- Test Lifecycle
- Test Tools
- Test Automation
- 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
- Architectural design
- Test design tools
- Detailed design
- Static analysis tools
- Code
- Test management tools
- Test execution and comparison tools
- Performance simulator tools
- Acceptance test
- System test
- Integration test
- Dynamic analysis tools
- Debugging tools
- Unit test
- Coverage tools
- Unit test
- System test
- Acceptance test
- Performance simulator tools
- Dynamic analysis tools
- Debugging tools
- Integration test
- Unit test
- Coverage tools
- Test management tools
- Static analysis tools
- Code
- Detailed design
- Architectural design
- Requirement specification

Test Tools – in the Process

- Requirement specification
- Architectural design
- Test design tools
- Detailed design
- Static analysis tools
- Code
- Test management tools
- Test execution and comparison tools
- Performance simulator tools
- Acceptance test
- System test
- Integration test
- Dynamic analysis tools
- Debugging tools
- Unit test
- Coverage tools
- Unit test
- System test
- Acceptance test
- Performance simulator tools
- Dynamic analysis tools
- Debugging tools
- Integration test
- Unit test
- Coverage tools
- Test management tools
- Static analysis tools
- Code
- Detailed design
- Architectural design
- Requirement specification

Test Tools – in the Process

- Requirement specification
- Architectural design
- Test design tools
- Detailed design
- Static analysis tools
- Code
- Test management tools
- Test execution and comparison tools
- Performance simulator tools
- Acceptance test
- System test
- Integration test
- Dynamic analysis tools
- Debugging tools
- Unit test
- Coverage tools
- Unit test
- System test
- Acceptance test
- Performance simulator tools
- Dynamic analysis tools
- Debugging tools
- Integration test
- Unit test
- Coverage tools
- Test management tools
- Static analysis tools
- Code
- Detailed design
- Architectural design
- Requirement specification
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

2. Test planning
   • Code comprehension
     – Show different views of the artefact

3. Test design and development
   • Syntax and semantics analysis
     – Check and warn

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

- Test data generator
- Requirements-based test design tool
- Capture/replay
- Coverage analysis

Often integrated with test execution tools
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

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pekka</td>
<td>Pukaro</td>
<td>1244515</td>
</tr>
<tr>
<td>Teemu</td>
<td>Tekno</td>
<td>587245</td>
</tr>
</tbody>
</table>
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

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

Test Suite

<table>
<thead>
<tr>
<th>TC1</th>
<th>TC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step1</td>
<td>Step2</td>
</tr>
</tbody>
</table>

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’

Tests can be run (replay) ...
Record Button switched off ...

Selenium commands
Data (values)
Location on Web-page (Target)
may use xpath, css, id, field, etc.

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

Typical problem: Test fails

What went wrong?

Time is needed to validate the login information…
The test must WAIT for the next web page to load before proceeding.
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.
import org.openqa.selenium.
import org.openqa.selenium.
import org.testng.
import org.testng.
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("//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("cpar1")).sendKeys("10");
    // Enter value 50 in the second number of the percent Calculator
driver.findElement(By.id("cpar2")).sendKeys("50");
    // click Calculate Button
driver.findElement(By.xpath("//a[@id='content']/table/tbody/tr/td[2]/input")
    // Get the Result Text based on its xpath
    String result = driver.findElement(By.xpath("//a[@id='content']/p[2]/span/font/b")
    // 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'

  Selects at current node ( . ) all elements ( // ) with id='menu' then takes the 3rd 'div' element ( div(3) ), then element 'a' (a) ... and then ...
  
  clicks on the link (math-calculator.html)
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("50")) {
            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

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.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("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("//[@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();
    }
}
```
Selenium-WebDriver API Commands and Operations

https://www.seleniumhq.org/docs/03_webdriver.jsp - selenium-webdriver-api-commands-and-operations

- Fetching a Web-page
- Locating UI Elements (WebElements)
- Getting text values
- User Input - Filling In Forms
- Moving Between Windows and Frames
- …

<table>
<thead>
<tr>
<th>By XPath:</th>
</tr>
</thead>
</table>

At a high level, WebDriver uses a browser's native XPath capabilities whenever possible. On those browsers that don't have native XPath support, we have provided our own implementation. This can lead to some unexpected behavior unless you are aware of the differences in the various XPath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Name</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>Firefox Driver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML,

```xml
<input type="text", name="example" />
<input type="text", name="site" />
```

The following number of matches will be found:

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>Internet Explorer</th>
<th>Firefox Driver</th>
<th>Internet Explorer Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input [&quot;example&quot;]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the "input" tag does not require the "type" attribute because it defaults to "text". The rule of thumb when using xpath in WebDriver is that you should not expect to be able to match against these implicit attributes.
Selenium-WebDriver API Commands and Operations

https://www.seleniumhq.org/docs/03_webdriver.jsp - selenium webdriver-api-commands-and-operations

- Fetching a Web-page
- Locating UI Elements (WebElements)

Locating elements in WebDriver can be done on the WebDriver instance itself or on a WebElement. Each of the language bindings exposes a “Find Element” and “Find Elements” method. The former returns a WebElement object matching the query, and throws an exception if such an element cannot be found. The latter returns a list of WebElements, possibly empty if no DOM elements match the query.

The “Find” methods take a locator or query object called “By”.

Examples of “By” strategies are shown on the next slides …
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Xpath

HtmlUnit:
Finds 1 arg for "input" tag
Finds 0 args for "INPUT" tag

Because:
- lower-cased
- not showing default args (i.e., "text" arg of "type" attribute)

By XPath
At a high level, WebDriver uses a browser's native XPath capabilities wherever possible. On those browsers that don't have native XPath support, we have provided our own implementation. This can lead to some unexpected behaviour unless you are aware of the differences in the various XPath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Name</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnit Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>Internet Explorer Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>No</td>
</tr>
<tr>
<td>Firefox Driver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML:

```html
<input type="text" name="example" />
<INPUT type="text" name="other" />
```

```java
List< WebElement > inputs = driver.findElements( By.xpath( "//input" ) );
```

The following number of matches will be found

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>HtmlUnit Driver</th>
<th>Firefox Driver</th>
<th>Internet Explorer Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>//input</td>
<td>1 (&quot;example&quot;)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>//INPUT</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the "input" tag does not require the "type" attribute because it defaults to "text". The rule of thumb when using xpath in WebDriver is that you should not expect to be able to match against these implicit attributes.
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Xpath

**HtmlUnit:**
Finds 1 arg for "input" tag
Finds 0 args for "INPUT" tag

Because:
- lower-cased
- not showing default args (i.e., "text" arg of "type" attribute)
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Xpath

InternetExplorer:
Finds 2 args for "input" tag
Finds 0 args for "INPUT" tag

Because:
- lower-cased
- showing all args of all attributes
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Xpath

Firefox:
Finds 2 args for "input" tag
Finds 2 args for "INPUT" tag

Because:
- not case-sensitive
- showing all args of all attributes
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By ID

**By ID**

This is the most efficient and preferred way to locate an element. Common pitfalls that UI developers make is having non-unique id’s on a page or auto-generating the id, both should be avoided. A class on an html element is more appropriate than an auto-generated id.

Example of how to find an element that looks like this:

```java
<div id="coolestWidgetEvah">...</div>
```

```java
WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
```
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Link Text

By Link Text

Find the link element with matching visible text.

Example of how to find an element that looks like this:

```java
<a href="http://www.google.com/search?q=cheese">cheese</a>
```

```java
WebElement cheese = driver.findElement(By.linkText("cheese"));
```
Data-Driven Testing

- Data-Driven Testing = Executing the same set of test steps with multiple (different) data
  - Typically in tabular format

- Test inputs and expected outcomes stored as data

- 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)
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=\"+", endRow=\"+", " +
                            "startCol=\"+", 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();
                System.out.println("error in getTableArray()");
            }
        }
    } catch (Exception e) {
        System.out.println("error in getTableArray()");
    }
    return (tabArray);
}

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

• Keywords also known as action words
• Keyword-driven testing improves data-driven testing:
  – Keywords abstract the navigation and actions from the script
  – Keywords and test data are read from an external data source
• When test cases are executed keywords are interpreted by a test library (=set of test scripts) which is called by a test automation framework
• Example 1: Login: admin, t5t56y; // 2 args
  AddCustomers: newCustomers.txt // 1 arg
  RemoveCustomer: Pekka Pukaro // 1 arg

• Example 2: click, checkbox, undo, ...
• More keywords (=action words) can be defined based on existing keywords
• Keyword driven testing ~= domain specific languages (DSL)
• Details: http://doc.froglogic.com/squish/4.1/all/how.to.do.keyword.driven.testing.html
• Another tool: http://code.google.com/p/robotframework/
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)
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.
GraphWalker  [http://graphwalker.org/](http://graphwalker.org/)

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

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.
GraphWalker  [http://graphwalker.org/](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
%> 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
...
```
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
java -jar graphwalker.jar offline -m Login.graphml "random(edge_coverage(100))"
```

```
%> 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
...
```
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
Structure of Lecture 5

- Test Lifecycle
- Test Tools
- Test Automation
- Lab 5
What can be automated?

What can be automated?

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

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

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 Tools
• Test Automation
• Lab 5
Lab 5 – Automated Web App Testing

Lab 5 (week 29: Mar 19 & 20) – Automated Web Application Testing (10 points)

Lab 5 Instructions & Tools

Submission Deadlines:
• Tuesday Labs: Monday, 25 Mar, 23:59
• Wednesday Labs: Tuesday, 26 Mar, 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 29: Mar 19 & 20) – Automated Web Application Testing (10 points)

Lab 5 Instructions & Tools

Find defects

Add test code (positive & negative)
Recommended Textbook Exercises

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

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

• Lecture 6:
  – Test Levels / Behavioural Testing / GUI Testing / Visual Testing

• In addition:
  – Read textbook Chapter 6