Automated, tool independent test case prioritization

Matthias Ratert, Secusmart GmbH

www.eurostarconferences.com
Introduction

• Secusmart GmbH (Germany)
  – Secure voice, messaging & email (NATO Restricted)
  – Hardware based encryption and authentication in mobile networks as well as landline networks
  – Convenient, secure and with crystal-clear voice quality
  – Mobile clients for Nokia, Blackberry, Android and iPhone

• Matthias Ratert
  – 14 years experience in Software development & testing
  – Worked within automotive and mobile phone industry
Test case

- A few (long?) cases
  - Detailed test steps
  - A lot of expected results

- More cases
  - Adding more tests
  - Splitting tests

- Many cases
  - Short test cases
  - Hierarchical and linked
  - Just a few expected results

- A lot cases
  - Just headlines
  - No expected results

What do you prefer?
Test case

- Are you afraid to expand your test suite continuously?
  - Will there be time to execute all of them in the future?
  - Will I spend too much time to create & maintain the suite?
  - Will I be able to select the “important” test cases?

I was!
1\textsuperscript{st} & 2\textsuperscript{nd} test case prioritization

- \textbf{1\textsuperscript{st} prioritization: Necessity}
  - Is this test case needed?
  - Define the test case or not?
  - Everyone is doing this...

- \textbf{2\textsuperscript{nd} prioritization: Importance}
  - Planned execution & code/requirements coverage:
    - Regression tests (high importance)
    - Execute often (medium importance)
    - Execute seldom (low importance)
  - Normally part of the test case definition
  - Many Test Management tools are supporting this !!!
3rd prioritization: Urgency

- Test session planning with one or several input factors:
  - New functionality implemented
  - Functionality changed (e.g. CRs)
  - Bugs fixed
  - Risks (error status/statistics, important/critical areas)
  - Past test execution coverage (Tested before? How often?)
  - Past test result ratio (e.g. failed before)
  - Error guessing techniques
  - Internal milestone
  - Important deliverable
  - External (customer) deliverable
3\textsuperscript{rd} prioritization: Urgency

• Done by the Test Manager before & during the test session
  – Ensure an optimal test coverage
  – Maximize the rate of fault detection
  – Get more time for Exploratory and Non-Functional Testing

• No good tool support (AFAIK) $\rightarrow$ MANUAL WORK

• Time consuming
  – Thorough prioritization decisions need to be made

• Complex & error prone
  – Applying different strategies is sometimes even impossible

Can we support the Test Manager?
Automated test case prioritization

• Easy to automate:
  – Past test execution coverage (Tested before? How often?)
  – Past test result ratio (e.g. failed before)

  → Started with in the 1st thesis

• Good to automate:
  – Bugs fixed
  – Risks (error status/statistics incl. error severity)

  → Covered with the 2nd thesis
Automated test case prioritization

• Possible to automate:
  – New functionality
  – Functionality changed
  – Risks (important/critical areas)
  – Important deliverable

  → Provide the possibility to automate later

• Hard to automate:
  – Error guessing techniques
  – Internal milestone
  – External (customer) deliverable

  → Do not automated
The 2 diploma theses

• 1\textsuperscript{st} thesis:

  \textbf{Implement prioritization algorithm in TestLink}
  – Only test execution data from TestLink is used
  – Internal databases can be used / "tweaked" easily
  – Provide a priority filter for the Tester during test execution

• 2\textsuperscript{nd} thesis:

  \textbf{Implement tool independent, stand-alone prioritization tool}
  – All kind of data (e.g. error DB, requirements tool) are gathered
  – Each test management tool can be "tweaked"
### 1st thesis: The basic idea

<table>
<thead>
<tr>
<th>Test Case 1</th>
<th>Session n-1</th>
<th>Session n-2</th>
<th>Session n-3</th>
<th>Session n-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passed</td>
<td>Priority ----</td>
<td>Priority ---</td>
<td>Priority --</td>
<td>Priority -</td>
</tr>
<tr>
<td>Failed</td>
<td>Priority +</td>
<td>Priority ++</td>
<td>Priority +++</td>
<td>Priority ++++</td>
</tr>
<tr>
<td>Not executed</td>
<td></td>
<td></td>
<td>Priority ++++</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 2</th>
<th>Session n-1</th>
<th>Session n-2</th>
<th>Session n-3</th>
<th>Session n-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passed</td>
<td>Priority ----</td>
<td>Priority ---</td>
<td>Priority --</td>
<td>Priority -</td>
</tr>
<tr>
<td>Failed</td>
<td>Priority +</td>
<td>Priority ++</td>
<td>Priority +++</td>
<td>Priority ++++</td>
</tr>
<tr>
<td>Not executed</td>
<td></td>
<td></td>
<td>Priority ++++</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case 3</th>
<th>Session n-1</th>
<th>Session n-2</th>
<th>Session n-3</th>
<th>Session n-4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
1\textsuperscript{st} thesis: Test execution filter

2 approaches to provide the priority filter to the Tester:

1. The build-in TestLink prioritization
   - Minimal changes to TestLink
   - Only 3 filter level
   - Each priority level is selected exclusively, higher priority levels are not included in the filter
The build-in TestLink prioritization

- The urgency is set manually
- Only 3 priorities available (High, Medium, Low), not expandable

Priority = Test Case importance x urgency
1. Tester uses the TestLink filter to find the cases to be executed
2. Only cases with the selected priority are shown
The modified TestLink prioritization

- Now the urgency is set automatically at the beginning of a test session
- The whole algorithm and the priority classification are configurable
- Recommendation is to set:
  - 10% - 20% test cases to HIGH
  - 20% - 50% test cases to MEDIUM

### Importance
- High (3)
- Medium (2)
- Low (1)

### Urgency
- High (10)
- Medium (4)
- Low (1)

### Priority = Test Case importance x urgency

- High (30)
- High (12)
- Low (3)
- High (20)
- Medium (8)
- Low (2)
- High (10)
- Medium (4)
- Low (1)
1st thesis: Test execution filter

2 approaches to provide the priority filter to the Tester:

1. The build-in TestLink prioritization
   + Minimal changes to TestLink
   - Only 3 filter level
   - Each priority level is selected exclusively

2. A new filter in the test execution
   + As many filter level as wanted (chosen: 15)
     → good granularity
   + Each priority level includes all higher priorities
   + Manual urgency can still be used
   - A lot of changes to TestLink sources
TestLink with the new filter
2\textsuperscript{nd} thesis: Error DB involvement

<table>
<thead>
<tr>
<th>Test Case 1</th>
<th>Session n-1</th>
<th>Session n-2</th>
<th>Session n-3</th>
<th>Session n-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passed</td>
<td>Priority ----</td>
<td>Priority ---</td>
<td>Priority --</td>
<td>Priority -</td>
</tr>
<tr>
<td>Failed</td>
<td></td>
<td><strong>Priority change depends on the linked bug</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not executed</td>
<td></td>
<td>Priority ++++</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bug severity:
- Showstopper: Huge priority increase
- Major: Average priority increase
- Minor: Minimal priority increase

Bug status:
- Fixed / recently closed: Huge priority increase
- Open: Average priority increase
- Closed (a while ago): Minimal priority increase
2\textsuperscript{nd} thesis: Tool independency

Stand-alone tool with configuration dashboard:
- Which algorithm(s) to prefer?
- How many test cases in highest priority?
- Visualization of the priority distribution
- ...

1. Use interface & filter from 1\textsuperscript{st} thesis
2. Use available (open) APIs
• Thesis just started
• Goals:
  – Support different Test Management tools, at least TestLink and HP QualityCenter
  – Provide a configuration dashboard and metrics
    • Select different strategies
      – concentrate more (or fully) on error verification
      – find as many new bugs as possible
      – do mainly regression testing
      – ...
  – Modular & generic design to enable further interfaces and extension
Conclusion

By using automated test prioritization

- a huge and expanding test suite is not a risk anymore
- thorough test prioritization is ensured for every test session
- the test manager gets support especially in stress situations
- the test coverage is maximized over several releases
  - time-sliced test coverage
- even low severity tests will be prioritized at some point of time
- more bugs are found
THANK YOU

In case if any comments and/or questions feel free to contact me via:

Matthias Ratert
Secusmart GmbH