Automated functional testing with keywords

Hans-Joachim Brede, BREDEX GmbH
The aim & challenges of functional testing
Keyword-driven testing → the answer?
Examples
Why do we need functional tests?

- **JUnit Tests**
  - Written by developers
  - Don’t test requirements

- **Tests through the API**
  - Don’t test that GUI and workflows work as specified

- **All green for JUnit & API tests**
  - Doesn’t mean that the application is correct
The aims of functional GUI testing

- **Check requirements as the user will see them**
  ... software is designed for the user!

- **Complete workflows via the GUI**
  Does the *happy path* work?
  Do different ways of working give the same results?
  Is the application consistent for the user?
  Are false inputs correctly handled?
  Can the user cause the application to crash?
The challenges of automation

Tests have to be in-step with development
Created close / in parallel to development
Can run as soon as the feature is delivered
→ prompt feedback on quality and acceptance
→ early error recognition (lower costs to fix)
The challenges of automation

▸ **Maintenance**

Workflows, GUI, requirements change frequently
Tests have to keep running despite changes

▸ **Test creation**

Support test **planning** and test **design**
Quick and easy
Black-box (from user’s perspective, not developer’s)

▸ **Readable**

For business users – can check tests against requirements
Why not record my tests?

- Delay: have to wait until the application is ready
- Can only test what already works
  And tests the implementation, **not** the requirements
- Inflexible and bound to unnecessary details
- Redundancy in the test
  Similar actions recorded twice and more
- Need to spend time programming to ensure maintainability
Keyword-Driven Testing

Based on the same principles as development
Do it once and only once!
Modularity from the outset
Reusing modules makes tests easier to maintain

GUI tests are made up of recurring actions
Lend themselves well to being tested with keywords
Each keyword executes a certain action / actions
→ library of keywords
More complex keywords made by combining other keywords
What’s behind the keyword?

► In some frameworks (e.g. FIT, Quality Center)
  Code written by automation experts in the team

► In other tools (e.g. GUIDancer)
  Library of basic keywords is present in the tool
  Tests aren’t written in program code
Benefits for the development process

- **Tests are readable**
  Can be checked against the requirements

- **Prompt feedback**
  Keyword creation can happen before code is written
  Tests run on regular builds → errors found early

- **Maintenance reduced**
  Reused modules → central changes update the whole test
  No programming work to maintain tests

- **Modularity**
  Easy to change, add and delete modules in the test
What to bear in mind with keywords

- **Test design is important**
  - What is reused
  - Flexible modules

- **Finding keywords**
  - Library must be well structured
  - Naming conventions help test team to write tests

- **Keyword explosion?**
  - Think carefully about when a keyword is necessary
Eclipse examples

Show view

<table>
<thead>
<tr>
<th>Component</th>
<th>Action</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>Select item from menu</td>
<td>Window/Show View/Other...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Wait for window</td>
<td>Show View</td>
</tr>
<tr>
<td>Tree</td>
<td>Select item from tree</td>
<td>&lt;path-to-view&gt;</td>
</tr>
<tr>
<td>Ok Button</td>
<td>Check enabled</td>
<td>True</td>
</tr>
<tr>
<td>Ok Button</td>
<td>Click</td>
<td>Once</td>
</tr>
</tbody>
</table>
More Eclipse examples

Open Editor

<table>
<thead>
<tr>
<th>Component</th>
<th>Action</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReUse Keyword: Show View</td>
<td></td>
<td>&lt;view&gt;</td>
</tr>
<tr>
<td>&lt;Tree&gt;</td>
<td>Check existence of item</td>
<td>&lt;path-to-item&gt;</td>
</tr>
<tr>
<td>&lt;Tree&gt;</td>
<td>Select item from tree</td>
<td>&lt;path-to-item&gt;</td>
</tr>
<tr>
<td>&lt;Tree&gt;</td>
<td>Select from context menu</td>
<td>&lt;context-menu-path&gt;</td>
</tr>
</tbody>
</table>
Summary / results

- 10% of project cost for test automation
- 10% the above for maintenance
- Exponential growth of test coverage through reuse

… now I understand why you don’t need capture-replay