Test-Driven Automation: 
Adopting Test-First Development to Improve Automation Systems Engineering Processes

Dietmar Winkler  Stefan Biffl  Thomas Östreicher

Institute of Software Technology and Interactive Systems, 
Vienna University of Technology

dietmar.winkler@tuwien.ac.at
http://qse.ifs.tuwien.ac.at
Motivation

- **Added value of automation systems with software components:**
  - Required *flexibility* of automation systems.
  - Increased *complexity* of software components in automation systems products.
  - Increased *product quality* by systematic software engineering approaches during development and operation.

- **Observations in Automation Systems Development Practice:**
  - Functional, testing, and diagnosis aspects are scattered over the code and hinder efficient (automated) systems testing.
  - Limitations in a systematic development process approach.

- **Challenges & Goals:**
  - Need for functional, testing, and diagnosis aspects during development and operation within a hierarchical systems design.
  - Efficient (automated) testing strategies.
  - Need for flexible and systematic systems development processes.
  - Application of Best-Practices learned from business IT software development in the automation systems domain (e.g., test-first approach and model application).
Test-First Development in Business IT Software Development

Test-Driven Development Steps:
1. **Think.** (a) selection of new requirements and (b) test case definition.
2. **Red.** Implementation and execution of test cases (failed).
3. **Green.** Implementation of functionality and test case execution until all tests are successful.

→ **Continuous Integration and Test:**
- Frequent test runs
- Immediate Feedback on test results (e.g., daily builds)
- Efficient regression testing.
- Automation and Tool support
Research Approach

- Need for flexible systems development and efficient testing approaches in the automation systems domain lead to:
  - Strict separation of functional, testing, and diagnosis aspects.
  - Application of test-first development to increase testing efficiency.

- Enriched automation systems development by systematic software engineering approaches, applying
  - Lessons learned from business IT software development.
  - Test-first development (TFD) in the automation systems domain.
  - Model-support to foster “test-first” in systematic systems development.

- Solution approach:
  - Concept for a test-driven automation component (TDA component).
  - Test-First development based on a systematic process approach (W-Model).
  - First evaluation in a prototype study: bottle sorting application.
Concept of a TDA Component

- **Strict separation** of functional, diagnosis, and test components.
- **Interaction via defined interfaces** within the TDA component and to lower-level and upper-level TDA components.
- **Hierarchical systems** design of typical automation systems.

Component structure:
- **Logic**: Represents functional aspects.
- **Diagnosis**: (a) Pass messages from logical aspects to lower-level components and (b) monitor systems responses.
- **Test**: Set system in a certain state (functional behaviour) including tests of systems states which should not be reached (e.g. an error state).
Test-First in Software Development

- Based on the W-Model (Baker et al, 2007) or V-Modell XT (Biffl et al, 2006).
Test-Levels in TDA

- Based on the W-Model focusing on
  - Requirements based test cases.
  - Test cases based on architecture, design and integration.
  - Component-based test cases.

- Test case definition on every level
  - Help Identifying success-critical issues on different levels.
  - Addresses individual stakeholder groups.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deliverables</th>
<th>Test Level</th>
<th>Stakeholders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Definition</td>
<td>Use Cases</td>
<td>System / Acceptance Testing</td>
<td>Customer, Factory Setting</td>
</tr>
<tr>
<td>Component Specification</td>
<td>State-Charts</td>
<td>Component Testing</td>
<td>Individual Engineer</td>
</tr>
<tr>
<td>Implementation of TDA Components</td>
<td>Function Blocks</td>
<td>Developer Testing</td>
<td>Individual Engineer</td>
</tr>
</tbody>
</table>
Sorting Application Prototype

- Bottle sorting application to illustrate TDA component application.
- Basic requirements:
  - Identification of individual items on a conveyor.
  - Stop at the appropriate loading station (stopper unit).
  - Grapping of the item (vacuum gripper) and moving the item into a box.
- Component Structure.
TDA Component Structure

- TDA component encapsulates functional, diagnosis, and test aspects.
  - **Automation control** via automation interface (e.g., sorting call)
  - **Diagnosis**: reporting the sorting status for (a) higher-level diagnosis components and (b) handling unit automation control unit.
  - **Test** control to set the system in a **certain state**, e.g., an error state (set/clear gripper error). **Observation** of system response via diagnosis and system **reaction** via automation aspects.

```plaintext
<<interface>>
Automation
+ sort() : void

<<interface>>
Diagnosis
+ partsSorted() : int

<<interface>>
Test
+ clearGripperError() : void
  + setGripperError() : Void
```

Automation Functionality  Diagnosis Functionality  Test Functionality

Handling Unit
System Level Test

- Expected user behaviour on requirements level.
- Test Cases based on use case models and requirements from user perspective.

Advantages:
- Common “language” between different disciplines.
- Early defect detection.
- Enhanced understanding of the customer requirements.
- Test cases as vehicle for communication between stakeholders

<table>
<thead>
<tr>
<th>No</th>
<th>Desc.</th>
<th>Level</th>
<th>Type*</th>
<th>Pre-condition</th>
<th>Input</th>
<th>Expected Result</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sorting a Part</td>
<td>System</td>
<td>NC</td>
<td>Handling Unit in idle Position</td>
<td>Command to sort part</td>
<td>Handling Unit in idle Position and part sorted</td>
<td>Handling Unit in idle position</td>
</tr>
<tr>
<td>2</td>
<td>Through-put</td>
<td>System</td>
<td>NC</td>
<td>Handling Unit in idle Position</td>
<td>Command to sort part</td>
<td>Part has been sorted in less or equal than 10sec.</td>
<td>Handling Unit in idle position</td>
</tr>
</tbody>
</table>

* Definition of test cases (type) according to (a) normal and regular cases (NC), (b) special cases (SC), and error cases (EC).
Integration & Unit Tests

- **State charts** are common practices in the automation systems domain.
- Ability for automated code generation.
- Modelling of state charts including error states.
- Example: handling unit on component level.

State Chart: Handling Unit

<table>
<thead>
<tr>
<th>No</th>
<th>Desc.</th>
<th>Level</th>
<th>Type*</th>
<th>Pre-condition</th>
<th>Input</th>
<th>Expected Result</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gripper move to Pos</td>
<td>Comp</td>
<td>NC</td>
<td>Handling Unit idle</td>
<td>Sort part</td>
<td>Gripper moved to intended position</td>
<td>Gripper is in intended position</td>
</tr>
<tr>
<td>2</td>
<td>Axis got stuck</td>
<td>Comp</td>
<td>EC</td>
<td>Handling Unit in idle Position</td>
<td>Sort part error after 3s</td>
<td>Positioning Unit reports an error; Handling Unit idle</td>
<td>Handling Unit in idle position</td>
</tr>
</tbody>
</table>
Mocking with State Charts

- Mocking is necessary to
  - **Simulate missing systems behaviour**, i.e., code which is not available so far.
  - Simulation of **hardware behaviour**.
  - Simulation of **error cases**, i.e., system states which cannot be reached during regular machine behaviour.

- State charts support modelling error states for testing purposes.
Lessons Learned & Future Work

- Increased flexibility and (software) complexity in the automation systems domain lead to **new challenges** in software construction.

- **Lessons learned from business IT software development** can help systems engineers in constructing high-quality products in short iterations.

- The concept of “Test-Driven Automation” applies:
  1. Systematic **software engineering process** in the automation systems domain.
  2. **Test-first development** on various levels of details to systematically increase product quality in short iterations.
  3. A strict **separation of functional, test, and diagnosis aspects** (TDA component structure) enables systematic product development including interaction over clearly defined interfaces.

- Lessons learned from a pilot application showed the expected benefits in a small show case application.

- **Future work includes**
  - Refining the process model and the **TDA structure** including diagnosis functions.
  - Investigating the **scalability** of the TDA concept in a larger project context.
  - Elaborating on a larger pilot application with industry partners with focus on **data collection** to empirically investigate the expected benefits of the novel TDA concept.
Test-Driven Automation: Adopting Test-First Development to Improve Automation Systems Engineering Processes

Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems

http://qse.ifs.tuwien.ac.at
Dietmar.Winkler@tuwien.ac.at
Referenzen