Testing in the Component Age

What is the Component Age?
What is Testing?
Component Specification
Component Testing

Overview

- What is the Component Age?
- What is Testing?
- Component Specification
- Component Testing
What are Components? Why are they composable?

Standardized Interfaces!

- Tape Recorder
- CD Player 2
- Receiver
- Cinch, 160 mV, 47 kΩ

Why are we able to use Components?

Standardized and Simple Interfaces!

- Audio/Audiovisual
- Video Interface Example 2
- Volume: 3.27 dB
- Length: 3.27 sec

What is the Component Age?
What is Testing?
Component Specification
Component Testing
What are Software Components?

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties”. C. Szyperski, 1998

Are Components just „big“ Objects or Classes?

- Components ...
  - Are bigger than classes
  - Can be build from classes (but must not)
  - Adhere to component standards (not to programming language standards)
  - Can be deployed independently
  - Are subject to composition by third parties
  - Are (should be) executable in different operational environments
  - Are used to build distributed, heterogeneous services and systems
Programming Model of the Component Age

- APIs (API 1, API 2, API 3, API 4, API 5, etc.)
- Components (Component 1, Component 2, Component 3)
- Container
- Application Server

J2EE / EJB - Container Architecture

- Web Server
- Servlet Container
- EJB Container
- Browser

- APIs: JNDI, API 2, API 3, etc.
- Components: SessionBean 1, EntityBean 1, SessionBean 2
- DBMS (Oracle, MySQL)
- JDK
- JDO
- JDBC

What is the Component Age?
What is Testing?
Component Specification
Component Testing
Roles in the Component Age

- **Application Server Provider**
  - Low level system services
  - Typically OS vendor, middleware, or database vendor

- **Container Provider**
  - Deployment tools and component runtime support

- **Component Provider**
  - Producer of Components
  - Domain Expertise

- **Component Deployer**
  - Resolves external dependencies

- **Application Assembler**
  - Combines components and other software into application

- **System Administrator**
  - Configures and administers the enterprise’s computing and networking infrastructure

Software Components: Some Conclusions

- Components could facilitate software reuse, but …
- Component specifications today are informal and (often) incomplete
  - Java/IDL-Interfaces (APIs) only specify types
  - (In EJB) focus on ensured interfaces
  - “Formal” specification for the “standard” life cycle only (state charts)
- Some more observations
  - Business operations most often only specified informally
  - Component Provider (normally) has
  - … no access to “real world” requirements
  - … no control over the components usage
  - Component deployer and application assembler (normally) have …
  - … no access to the components source code
  - … no control over the components maintenance and evolution
  - … no way to get rid of the components extra functionality
Where are we now?

- What is the Component Age?
- What is Testing?
- Component Specification
- Component Testing

Why do we Test?

- Errare humanum est! (Err is human)
- Even with formal specification and program generation ...
  - Spec could be wrong
  - Generator doesn’t work as expected
  - Platform doesn’t work as expected
  - ...
- Testing checks that the Software ...
  - ... does what it should do and ...
  - ... doesn’t what it shouldn’t do
- But always remember that ...
  - ... Testing can only show the presence of faults (i.e. bugs), not their absence!!!
  [E.W. Dijkstra]
What is the Testing Process?

- Test Planning
- Test Specification
- Test Execution
- Result Recording
- Completion Checking
- Start
- End


When do we Test?

- Requirements
- Analysis
- Architecture
- Design
- Coding
- Acceptance Test
- System Test
- Integration Test
- Unit Test
- Test Planning
- Test Specification
- Test Execution
- Result Recording
- Completion Checking
- Start
- End

Test cases based on work artifacts (documents)
How do we Test?

- **Static Testing**
  - No executables necessary
  - Manually: reviews, inspections
  - Automated: source code analysis

- **Dynamic Testing**
  - Executables necessary
  - Test cases with input and expected behaviour/output
  - Test harness for unit and integration testing needed

---

Black Box Testing vs. White Box Testing

**Black Box Testing**
- Input parameters/State
- Specification needed!
- Test object
- Output parameters/State

**White Box Testing**
- Input parameters/State
- Specification and realization (sources) needed!
- Test object
- Output parameters/State
Where are we now?

- What is the Component Age?
- What is Testing?
- Component Specification
- Component Testing

Who Specifies Components?

- Component Providers Functionality
- Application Assembler Requirements
- Application Users Requirements
- Component Delivered Value
Is Component Specification Different?

Our observations on component software:

- Component Provider (normally) has
  - ... no access to "real world" requirements
  - ... no control over the components usage

- Component deployer and application assembler (normally) have ...
  - ... no access to the components source code
  - ... no control over the components maintenance and evolution
  - ... no way to get rid of the components extra functionality

- Component interfaces are the minimal contractual base between component provider and component customer (deployer / application assembler)
- Business operations most often only specified informally

What to Specify? Interfaces!

- Application-Client
- Component-Container
- Component-Application
- Component-DB-Server
- J2EE Server
- EJB Container
- Object Entity Bean
- Customer Entity Bean
- Order Entity Bean
- Component DB-Server
- Database Server
How to Specify Interfaces? Design by Contract (B. Meyer)

If you promise to call me with the precondition satisfied then I, in return, promise to deliver a final state in which the postcondition (and my invariant) is satisfied.

- Client uses suppliers (public) operations
  - Precondition = clients price
  - Postcondition = suppliers obligation
  - Invariant = "eternal" assumptions
- Contract
  - Iff the client satisfies the precondition, then the supplier guarantees the postcondition!
  - Client and supplier satisfy their invariants before and after each (public) operation execution
- A contract is a prescription, not only a description!

Specification of Component Interfaces with Assertions

- Precondition (of an operation)
  - States the properties that must hold whenever the operation is called.
  - Refers to input-parameters of the operation and the state of the component
- Postcondition (of an operation)
  - States the properties that the operation guarantees when it returns (assuming its precondition was satisfied)
  - Refers to output-parameters of the operation and the state of the component
- Both preconditions and postconditions describe properties of individual operations – but often there are more general properties
- Invariant (of the Component)
  - Expresses global properties of all instances of a component, which must be preserved by all operations
  - Refers to the state of the component
  - Must be satisfied before and after each execution of an operation
Example: BoundedStack (Stack with Bounded Capacity)

component BoundedStack
State preserving operations
  size():integer; // Number of elements
  MAXSIZE(): integer; // Maximum count of elements
  top():Object; // Pointer to topmost element
State changing operations
  BoundedStack(maxSize: integer);// Constructor
  ~BoundedStack(); // Destructor
  push(element: Object); // Stack element on top
  pop(); // Removes topmost element

Contract of Component BoundedStack

```java
component BoundedStack {
  /** invariant@ self.size() >= 0 AND self.size() <= self.MAXSIZE() */
  public BoundedStack (Integer maxSize) {
    /** pre@ maxsize > 0 */
    /** post@ self.MAXSIZE() = maxsize@pre */
  }
  public void push (Object item) throws FullStackException {
    /** pre@ self.size() < self.MAXSIZE() */
    /** post@ self.size() = self.size()@pre + 1 */
  }
  public Object top () throws EmptyStackException {
    /** post@ return != null */
  }
  public void pop () throws EmptyStackException {
    /** post@ self.size() = self.size()@pre - 1 */
  }
  public Collection all () {
    /** post@ (self.size() > 0 implies return.size() = self.size()) AND
      (self.size() = 0 implies return = null) */
  }
}
```
State-Based Specification for BoundedStack

Three (five) states:
- empty: size() = 0;
- filled: 0 < size() < MAXSIZE();
- full: size() = MAXSIZE();
- initial: before construction;
- final: after destruction;

Where are we now?

- What is the Component Age?
- What is Testing?
- Component Specification
- Component Testing
Black Box Testing of Components

- **Goal**
  - Validation of the components interface, i.e. its public operations (and members)

- **Value**
  1. Conformance of the components realization w.r.t. its specification
  2. Robustness of the component

- **Needs**
  - Assertions, i.e. pre- and postconditions for each public operation of the component under test (CUT) together with its invariant

- **Result**
  - Reusable and extendable component test cases which validate the components interface conformance and its robustness

---

### Contract-Based Test Cases for BoundedStack

**Context BoundedStack**

<table>
<thead>
<tr>
<th>Method</th>
<th>Precondition</th>
<th>Postcondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>invariant()</td>
<td>self.size() &gt;= 0 AND self.size() &lt;= self.MAXSIZE()</td>
<td></td>
</tr>
<tr>
<td>BoundedStack()</td>
<td>maxSize &gt; 0</td>
<td></td>
</tr>
<tr>
<td>push()</td>
<td>pre@ self.size() &lt; self.MAXSIZE()</td>
<td></td>
</tr>
<tr>
<td>top()</td>
<td>pre@ self.size() &gt; 0</td>
<td></td>
</tr>
<tr>
<td>pop()</td>
<td>pre@ self.size() &gt; 0</td>
<td></td>
</tr>
<tr>
<td>all()</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

- **Conformance testing**: precondition satisfied
- **Robustness testing**: precondition not satisfied
- **Test oracle**: postcondition satisfied
Life Cycle-Based Test Cases (State Based)

Three (five) states:
- empty: size() = 0;
- filled: 0 < size() < MAXSIZE();
- full: size() = MAXSIZE();
- initial: before construction;
- final: after destruction.

Applicable Black Box Testing Models and Methods

<table>
<thead>
<tr>
<th>Specification</th>
<th>Type</th>
<th>Concrete Syntax</th>
<th>Method / Coverage Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract based (declarative)</td>
<td>Formal language</td>
<td>UML-OCL</td>
<td>All conditions, MC/DC, ...</td>
</tr>
<tr>
<td>Formal theory</td>
<td>Object-Z, VDM</td>
<td></td>
<td>All clauses</td>
</tr>
<tr>
<td>Informal (natural language)</td>
<td>API</td>
<td></td>
<td>All functions</td>
</tr>
<tr>
<td>State based (operational)</td>
<td>Diagram with formal semantics</td>
<td>UML state chart</td>
<td>All states, all transitions, n-Paths, ..., all paths</td>
</tr>
<tr>
<td>Interaction based (operational)</td>
<td>Message based</td>
<td>SDL MSC, UML</td>
<td>All messages, all nodes, all branches, ..., all paths</td>
</tr>
<tr>
<td>Structure based</td>
<td>UML communication</td>
<td></td>
<td>All messages, all links</td>
</tr>
<tr>
<td>Function based (declarative/operational)</td>
<td>Informal (natural language)</td>
<td>UML use case diagram</td>
<td>Normal flow, all alternate flows, ..., all flows</td>
</tr>
<tr>
<td></td>
<td>Diagram with formal semantics</td>
<td>UML activity diagram</td>
<td>All actions, all transitions, n-Paths, ..., all paths</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UML sequence diagram</td>
<td>All messages, all nodes, all branches, ..., all paths</td>
</tr>
</tbody>
</table>
Component Testing Levels

- Container test
  - Standard conformity, performance, robustness, ...
- Component functional test
  - Standard conformity, delivered functionality, required functionality
- Component non-functional test
  - Deployability, robustness, performance, vulnerability, security, ...
- Component integration test
  - Interoperability, ...

Component Testing Viewpoints

- Container provider
  - Adheres to standards
  - Validates containers standard conformity
  - Provides black box standard conformity test cases
- Component provider
  - Provides specification (API + Assertions)
  - Validates component with black box and white box testing
  - Validates components standard conformity
    (Container provider / versions / ...)
  - Provides black box test cases
  - Confirms white box coverage (e.g. C1)
- Component deployer
  - Demands black box test cases (JUnit, ...)
  - Validates components deployability
  - Re-validates components delivered functionality
- Application assembler
  - Demands specification (API + Assertions)
  - Demands white box coverage
  - Validates components required functionality with more black box test cases
Some Lessons Learned

- Specify your requirements (!)
- Standardize your infrastructure
  - Application server provider / versions ...
  - Container provider / versions ...
  - Application architecture (product lines)
- Ask for component specification and test cases
- Automate your tests (regression testing will come)
- Ask for component maintenance and evaluation arrangements
- In doubt: ask for source code agreements!
- Educate your people
  - Component modelling, architecture, and design
  - Design by contract
  - Black box testing techniques

What Did We Cover?

- What is the Component Age?
  - Components are more than objects/classes
  - Reusability in predefined environments (application server, container, …)
  - New roles
- What is Testing?
  - Testing process: phases and levels
  - Black box vs. white box testing
- Component Specification
  - Specifications needed (especially for business operations)
  - Specify required / ensured interfaces
  - Use design by contract (pre- and post-conditions, invariants, exceptions)
- Component Testing
  - New testing viewpoints and levels
  - White box testing only for component provider
  - Do black box conformance and robustness testing
  - Ask for component specification and black box test cases
Literature and Links

- Winter, M: Testfallermittlung aus Komponentenschnittstellen. Beitrag zum Imbus QS-Tag 01, Nürnberg, 2001
- GI-TAV Arbeitskreis „Testen objektorientierter Programme“: Test von Komponenten. 18. GI-TAV Workshop, June 20./21. 2002, Hasso Plattner Institut, University of Potsdam
- GI Fachgruppe “Test, Analyse und Verifikation von Software” (TAV) www.gm.fh-koeln.de/~winter/tav
  Next Workshop: 17./18. February 2005, University of Appl. Sciences Bremen