Testing and Analysis of Component-Based Software - Architecture Perspectives

Keynote Speaker: Jerry Gao

July, 2007

San Jose State University
email: jerrygao@email.sjsu.edu
URL: http://www.engr.sjsu.edu/gaojerry

Presentation Outline

- The Evolution of Component-Based Software Development
- What Have Been Done in Component-Based Software Testing?
- Open Issues, Challenges, and Needs in Testing CBS
- Roles of Architecture-Based Testing in Component-Based Software
- Architecture Perspectives in Testing Complex Components
- Architecture Perspectives for CBS System Testing and Evaluation
- Architecture-Based Modeling & Analysis for CBS System Evolutions
- Major Problems, Challenges, and Needs in Architecture-Based Testing
The Evolution of Software Reuse

<table>
<thead>
<tr>
<th>Level of Abstraction</th>
<th>Increasing Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platforms, or Middleware</td>
<td>Complex Components</td>
</tr>
<tr>
<td>Class Libraries</td>
<td>Components</td>
</tr>
<tr>
<td>Function Library</td>
<td>Frameworks</td>
</tr>
<tr>
<td>Limited Implementation Reuse</td>
<td>Design Reuse and Implementation</td>
</tr>
<tr>
<td>Large-Scale Implementation Reuse</td>
<td>All level reuse</td>
</tr>
<tr>
<td>Class Libraries</td>
<td>Components</td>
</tr>
<tr>
<td>Function Library</td>
<td>Frameworks</td>
</tr>
<tr>
<td>Limited Implementation Reuse</td>
<td>Design Reuse and Implementation</td>
</tr>
<tr>
<td>Large-Scale Implementation Reuse</td>
<td>All level reuse</td>
</tr>
<tr>
<td>Platforms, or Middleware</td>
<td>Complex Components</td>
</tr>
</tbody>
</table>


Software Development Processes in CBS

<table>
<thead>
<tr>
<th>Process Type</th>
<th>Reusable Components</th>
<th>Business-Specific Components</th>
<th>Application Systems</th>
<th>Integrated Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business-Based Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System-Oriented CBS Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service-Oriented CBS Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Evolution of Component-Based Software Development
CBS Software Development Trend and Roadmap

CBB Assets
- Assembly Components
- Business Components
- Framework Components
- Reusable Components

Reusable Components
- Application-Based Architecture
- Component-Based Architecture
- Domain Architecture-Driven Development Process
- Assembly-Driven Development Process
- Production-Line Architecture
- Reuse-Driven Development Process
- Application-Driven Development Process

Evolution-Oriented Architecture
- Architecture Perspectives
- Testing and Analysis of Component-Based Software – Architecture Perspectives

Different Types of Software Components
- Platform
- Package
- Framework or Middleware
- Class Library
- Function Component
- Business Component
- Container & Compositor
- Connect-ware (Connectors & Adaptors)
- Customizable Component
- Parameterized Component

Evolution of Component-Based Software Development

Jerry Gao, 7/2007
CBS Development Needs and Trend

Effective manage reusable components
Reuse component artifacts at all levels
Reuse component in all phases
Effective reuse on production line
Effective reuse cross production lines

Easy replacement & upgrade
Easy assembly & re-assembly
Easy configure & re-configure
Easy re-test and maintenance

Reuse-Oriented CBS Development

Evolution-Oriented CBS Development

What Have Been Done in CBS Testing?

Since the late 1990s, researchers have worked on component-based software testing.

• Many papers + two technical books published in CBS testing in last 10 years.

• The research results mainly focus on component testing in the following areas:
  • Understanding distinct problems and challenges in CBS testing
  • Component testability (verification, measurement, and analysis)
  • Testable components (self-test/BIT/testable components)
  • Component unit testing (test models, test generation, and adequacy)
  • Component re-testing and coverage analysis
  • Component test frameworks and automation solutions
  • Component performance measurement and evaluation metrics
What Have Been Done in CBS Testing?

- The existing research results mainly focus on component-based software testing in the following areas:
  - Problems and challenges in CBS system testing
  - Component integration (UML-based, architecture-based…)
  - CBS system regression testing and maintenance
    - Addressing retesting issues in component interactions and interfaces
    - Model-based regression testing, change and impact analysis
  - CBS system performance testing and evaluation
    - Aim at CBS system performance evaluation models, metrics, and frameworks
  - GUI component-based testing

Major Problems in Testing Complex Components

- Poor Component Testability
  - Poor quality of component artifacts
  - Available component API interface for functions and services
  - No standard well-defined test interfaces
  - Components are structured without concerning testing

- Lack of Component Quality Information
  - Many in-house components, library, frameworks are not well-tested
  - Lack adequate test models, test criteria and methods

- Very Difficult and Costly To Test and Maintain
  - Most of them are structured and integrated in Ad-Hoc approaches
  - The interactions between internal components may not well-defined and they commonly are not traceable and checkable.

- Complex Component Architecture Models Not Facilitating Testing & Evolution
  - No consideration to facilitate function validation
  - No consideration to facilitate performance evaluation

- No Auto-Test Solutions and Tools to Support Complex Component Testing
Open Issues, Challenges, and Needs

What are the open issues and challenges in component testing?

- How to validate complex components?
  (i.e. customizable & parameterized components, middleware and platforms)
  - Testing and re-test analysis problems for complex components
    - Test complexity, coverage analysis, test cost analysis
  - Test and re-test adequacy issues for complex components
    - Test adequacy models, criteria and evaluation
  - Testability enhancement, verification, and evaluation for complex components
    - How to make complex components testable in a scalable way?

What are the challenges and needs in complex component testing?

- How to validate auto-generated enterprise components?
  - Test models, process, test coverage criteria and test methods
- How to cope with complex enterprise components made of auto-generated components plus other components?
- Test automation for complex components
  - Scale-up test driver and test stub generation technology and solution
  - Support various connections to other components and test tools
Open Issues, Challenges, and Needs

What are the open issues and challenges in CBS system testing?

• Systematic regression testing analysis for CBS system testing
  • Systematic CBS system re-testing complexity and cost analysis
  • Systematic CBS system change and impact analysis
• Test adequacy for CBS system testing and regression validation
  • Adequate test models and test criteria concerning changed CBS systems
• Increase CBS system testability for system evolution
  • Increase system traceability, controllability, and observer ability on component interactions and connections
  • Develop and use testable connect-ware and middleware
  • Facilitate system integration between components
  • Facilitate system performance validation and evaluation

What are the major needs in CBS testing?

• Testable architecture models and technology for complex components
• Evolution-oriented component test automation tools and solutions for complex components
  • Reusable and supporting plug-in-and-test
  • Measurable and support plug-in-and-measure
• Increase CBS system testability for system evolution
  • Facilitate system integration between components
  • Facilitate system performance validation and evaluation
• Evolution-oriented CBS system technology and solutions to support regression testing during system evolution:
  • Replace components, test and analysis
  • Change interactions, test and analysis
  • Re-measurable
**Major Observations of Software Architecture**

Major observations about software architecture based on existing definitions:

- Software architecture represents the system’s organization & structure.
- Software architecture defines elements/components and their relationships, including interfaces, collaborations, connections, and constraints.
- Software architecture presents a system’s functional partitions as functional components, middle-ware, and connectors.
- Software architecture presents high-level architectural styles and principles.

---

**Component-Based Software Architecture Artifacts**

Software Architecture Specification:
- Formal Language (ADL, UML, SysML,...)
- Informal Text

---

- Component
  - Code-based Architecture
  - Design-based Architecture
  - Dependent Interface Specification
  - Component API Specification

- Complex Component
  - Structure Specification
  - Configuration Specification
  - Dependent Interface Specification
  - Component API Specification

- CBS Software
  - Software Architecture Specification
  - Component Connectivity & Connection Protocol
  - Configuration Specification
  - External System Interface Specification
Three Common Approaches To Increasing Component Testability

- **Method #1: Framework-based testing facility**
  - Creating well-defined framework (such as a class library) is developed to allow engineers to add program test-support code into components according to the provided application interface of a component test framework.

- **Method #2: Build-in tests (Self-test components)**
  - Adding test-support code and built-in tests inside a software component as its parts to make it testable.

- **Method #3: Systematic component wrapping for testing**
  - Using a systematic way to convert a software component into a testable component by wrapping it with the program code that facilitates software testing.

**Self-Test Component Architecture Model**

- **Self-test component**
- **Self-test interface**
  - Self-test
  - Self-run
  - Self-configure
  - Self-report
  - Exception Handler

**Component API**

**Major features:**
- Support two operation modes: test mode and normal execution mode
- Component tests are embedded inside components
- Useful for user-oriented acceptance tests
- Code and test change dependent
**Testable Component Architecture Model**

- Component Application Interface
- Testable Component
- Component Standard Test Interface
- Independent Component (IC)
- Component API Test Wrapper

- Standard & Consistent Component Test Interface
- Automatic Generation of Test Wrapper
- Using A Systematic Way to Make Testable Components
- Easily Interact with Reusable Test Tools and Unit Test Environment

**Component Test Framework for Testable Components**

- Component Management System
- Component Test Tools
- Component Test Framework
- Component Test Interface
- Testable Component
A Reusable Test-Bed for Testable Components

A Case Study for Testable Components

Cost - in terms of Line of code

- Total line of code
- Max line of code
- Min line of code
A Case Study for Testable Components

Cost - in terms of time (minute)

Driver with fixed data
Driver with non-fixed data
Component Testability tool

A Case Study for Testable Components

Auto-Generated Enterprise Components and Driven Applications

Data-Oriented Semantic Description
Event-Oriented Semantic Description
Workflow-Oriented Semantic Description
State-Oriented Semantic Description

Enterprise Middle/Component Generator

Application System

Reusable components
Created components

Jerry Gao, 7/2000
Testing Issues & Challenges for Auto-Generated Components

Major observations about software architecture based on existing definitions:

- Software architecture represents the system’s organization & structure.
- Software architecture defines elements/components and their relationships, including interfaces, collaborations, connections, and constraints.
- Software architecture presents a system’s functional partitions as functional components, middle-ware, and connectors.
- Software architecture presents high-level architectural styles and principles.

A Distributed Component Evaluation Environment

Communication Interface to Performance Agents
- Performance Server Controller
  - Performance Server Controller
  - Performance Server Configuration
  - Performance Monitor
  - Performance DB Access Program
  - GUI of the Performance Server

Component Performance Repository
## System Overhead

### Table 1. System CPU Overhead for Performance Tracking Code

<table>
<thead>
<tr>
<th>Type</th>
<th>Avg PT w/o Tracking Code (milliseconds)</th>
<th>Avg PT with Tracking Code (milliseconds)</th>
<th>Overhead (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>22</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>Throughput</td>
<td>10004.2</td>
<td>10004.4</td>
<td>0.2</td>
</tr>
<tr>
<td>Reliability</td>
<td>10004.6</td>
<td>10008.6</td>
<td>4</td>
</tr>
<tr>
<td>Availability</td>
<td>10036.6</td>
<td>10038</td>
<td>1.4</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>24.2</td>
<td>30</td>
<td>5.8</td>
</tr>
</tbody>
</table>

### Table 2. Code Wrapping Effort

<table>
<thead>
<tr>
<th>Performance Type</th>
<th>Effort in inserting lines of tracking code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>&lt; 5</td>
</tr>
<tr>
<td>Throughput</td>
<td>5 to 10</td>
</tr>
<tr>
<td>Reliability</td>
<td>5 to 10</td>
</tr>
<tr>
<td>Availability</td>
<td>5 to 10</td>
</tr>
<tr>
<td>Resource Utilization</td>
<td>&lt; 5</td>
</tr>
</tbody>
</table>

## Application Examples and Case Study

![Bar chart showing test costs for measuring function speed, throughput, and availability.](chart.png)

1 - Test Cost for Measuring Function Speed  
2 - Test Cost for Measuring Throughput  
3 - Test Cost for Measuring Availability
A Framework and System for Component-Based System Performance Measurement

Application Examples and Case Study

<table>
<thead>
<tr>
<th>Hours</th>
<th>Manual</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

1 – Instrument Code Cost for Speed
2 - Measurable Code Cost for Throughput
3- Measurable Code Cost for Availability

Architecture-Based Software Testing for Modern CBS

Architecture-Based Test Planning and Analysis
- Architect & Design Testable Components (Complex Components)
- Architecture Artifacts
- Architecture-Based Test Models, Methods, and Criteria
- Architecture-Based CBS System Evaluation & Measurement
- Architecture-Based CBS Change Impact Analysis for Regression Testing
Architecture-Based Software Testing for Modern CBS

What are the major tasks and scope of architecture-based testing for CBS?

- Plan and analyze software testing based on well-defined software architecture artifacts (Complexity and Cost)
- Develop software architecture-based test models, methods, and criteria
- Analyze and conduct changes and impacts for component and software evolution based on well-specified architecture artifacts
- Evaluate and measure CBS software based on architecture artifacts
- Architect and design for component and software testability
  - Study how to develop and present testable complex components and software in architecture perspectives

Why Architecture-Based Testing Is Important For Modern CBS?

- Complex components with poor testability
  - lead very high test costs and difficulty to vendors and users
- Modern components with complex component structures, compositions, and configurations
  - lack of test models, test adequacy criteria, methods and tools
- Auto-generated components
  - provide new challenges in component validation
- Frequent component changes & upgrades
  - make CBS regression testing very difficult during evolution of large CBS
- Diverse component reuse and customization in large-scale CBS
  - bring an urgent need on a cost-effective roadmap for system black-box testing
Component-Based Software Architecture Perspectives

- User Interface Components
- Network Components
- Middle-ware Components
- Business Components
- Data-Centered Components

Component Interactions
- API-based interactions
- Message-based interactions
- Event-based interactions

CBS Architecture Perspectives - Component Interactions

Component Public Accessible Data
Component API
Provided Component Functions

Component Interactions
- API-based interactions
- Message-based interactions
- Event-based interactions

Evolution of Component-Based Software Development
Testing and Analysis of Component-Based Software – Architecture Perspectives

CBS Architecture Perspectives
– Complex Component Structure

Component Package

Class Library

UML Package Structure

Inheritance hierarchy

Class Relation Graph

CBS Architecture Perspectives
– Component Composite Structure

Container/Composite Component

Container Component

Statically Composition

Dynamically Composition

The Component Composition Structure is a Part of CBS Architecture Information
Testing and Analysis of Component-Based Software – Architecture Perspectives

CBS Architecture Perspectives - Component Customization and Configuration

Customizing Component A

- Component A
  - Disabled Functions
  - Selective Functions
  - Enabled Functions

Component A’

Enabled Functions

Customized Component

ADT Parameterized Component

AND Provided Functions

Parameterized Functions

Supported Data Types

DT1 DTn

Functional Parameterized Component

AND Provided Functions

Select-P

Parameterized Functions

Testing and Analysis of Component-Based Software – Architecture Perspectives

CBS Architecture Perspectives – Complex Component Level

Component Client View:
- Component API

External interaction:
- Component connection
- Component messaging

Component configuration and customization information:
- Component configuration model
- Component customization model

Internal structural view:
- Internal component connectivity
- Internal composition structure

Dynamic view:
- Dynamic state model
- Internal process/sequence view
Major Needs in Testing and Analysis for Complex Components

- Well-defined complex architecture specification technique:
  - present internal component-level structure (i.e. component interactions)
  - present customization, configuration, and composition relationships
- Well-defined complex component architecture for testing and upgrading:
  - testable architecture models or styles for complex components
  - increase testability of complex components
- Reusable and scale-up test bed and test automation environment
  - Change, plug-in, and re-test
- Reusable evaluation platform for measuring complex component
  - Change, plug-in, and re-evaluate
- Well-defined testing and analysis methods for complex components
  - Testing complexity and test cost analysis for planning
  - Change & impact analysis for complex components
  - Regression testing in test reuse selection

Required Complex Component Architecture Artifacts

Component Architecture Information:
- Testable Component Architecture Model
- Formal Component API Specification
- Formal Component Interaction Specification
- Internal Component Structure Specification
- Internal Component Relation Specification
CBS Architecture Perspectives – System Level

Dynamic View
- Component sequences
- State-based model
- Action-based model

Function Partition View:
- Function Partition Structure Model

Software Architecture & Structure Views:
- Component connection model
- Software architecture model
- Software assembly & configuration model

User View:
- GUI Structure Model
- User Navigation Model

External System View:
- External Interaction Model

CBS Architecture Perspectives – System Deliverables

A Component-Based System

System Source Code
System Binary Code
System Configuration/Customization
Component Descriptions
Architecture Descriptions

- System Structure-Based Architecture Specification (i.e. ADL)
- System Component Connectivity Specification
- System Function Partition Specification
- System Dynamic Behavior-Based Structure Specification
Fact: Existing component connection approaches and technologies show poor testability and ad-hoc ways:
- not traceable, not controllable, not monitorable
CBS Architecture-Based System Regression Testing

A Component-Based System

- Impacted component or
  Impacted component entity

- Non-impacted component or
  Non-Impacted component entity

Tasks:
- Change & impact analysis
- Re-test strategy
- Test case reuse and selection
- Complexity and cost analysis

- Changed test sets
- Changed test scripts

Major Needs in Architecture-Based Testing and Analysis for CBS

- Architecture specification and enable technologies
  - facilitate auto-system assembly, configuration, and customization
  - support component re-assembly and re-configuration

- Architecture-based models and regression test methodologies
  - automatic component change and impact analysis
  - automatic test reuse and selection at the component and system levels

- Architecture-based modeling and analysis methods
  - auto-evaluate CBS system test complexity and costs

- Well-defined component connect-ware and connector technology
  - facilitate component change and impact analysis
  - support tracking and monitoring of component interactions
Component Change and Impact Analysis

Affected Components

Derived: C1

use (g, f)

use (g, f)

use (n, m)

use (s, t)

composted

mess (x, y)

use (p, q)

Use (x, y)

mess (x2, y2)

composted

mess (x1, y1)

Component Connect-Ware Technology

New connect-ware:  ➔ traceable, observable, controllable, testable
➔ maintainable and measurable

New connect-ware: regulated, consistent, standard, well-defined and reusable.
Question & Answer