Using Dynamic Execution Data to Generate Test Cases

Rozita Dara
Shimin Li
Weining Liu
Angi Smith Ghorbani
Ladan Tahvildari

University of Waterloo
Canada

Research In Motion
Canada

Star Technologies
University of Waterloo, Ontario, Canada

Blackberry
Background

• Joint project of
  
  - Software Technologies Applied Research (STAR) laboratory led by Prof. Ladan Tahvildari from University of Waterloo, Canada
  
  - Research In Motion (RIM) – A leading designer, manufacturer and marketer of innovative wireless solutions (e.g. BlackBerry) for the worldwide mobile communications market

• This project aims at reducing the amount of required testing time and associated resources to decide whether a software product of RIM meets its quality release criteria
Outline

• Motivation
• SV&V Testing Model
• SV&V Testing Process Flow
• Components of Testing Process
• Contributions & Summary
• Thank You
Motivation

• Satisfy newly added test goals
  – Find bugs earlier
  – Achieve reduced bug escape rate
  – Achieve required code coverage

• Reduce time to market
  – Reduce SV&V testing time
  – Improve testing effectiveness (e.g. improve test case quality)
SV&V Testing Model
SV&V Testing Process Flow

1. Software Requirements
2. Historical Testing Data
3. Requirement Modeling
4. Test Case Generation
5. Test Goals
7. Test Case Suites
8. Builds
9. Test Execution
10. Test Execution Data (Coverage, # of defects, etc.)
11. Product Test (PT) Model
Testing Process – Requirement Modeling

1. Requirement Review
2. Feature Identification
3. Test Scenario Creation
4. Test Variable Identification
5. Test Value Profiling

Software Requirements

Historical Testing Data

Product Test (PT) Model
- Features To Be Tested
- Test Scenarios
- Variable-Value Matrix
- Priority Matrix
- Traceability Matrix
- Test Value Profile
Testing Process – Requirement Review

1. Requirement Review
   - Reviewing and understanding new requirements from testing point of view
   - Ensuring that requirements are complete, unambiguous, logically consistent and testable
   - Example: An online hotel room reservation system
     R1 – Users must be able to search and reserve rooms by specifying check-in/check-out date, room type, # of rooms, and # of persons
Testing Process – Feature Identification

1. Requirement Review
2. Feature Identification
3. Test Scenario Creation
4. Test Variable Identification
5. Test Value Profiling

Product Test (PT) Model
- Features To Be Tested
- Test Scenarios
- Variable-Value Matrix
- Priority Matrix
- Traceability Matrix
- Test Value Profile

Software Requirements

Historical Testing Data

Feature Identification
- Identifying features to be tested from the reviewed requirements (what to test)
- Example: Features for R1
  - R1 – Users must be able to search and reserve rooms by specifying check-in/check-out date, room type, # of rooms, and # of persons
  - F1 – Users are able to search rooms
  - F2 – Users are able to book the rooms found
Testing Process – Test Scenario Creation

1. Requirement Review
2. Feature Identification
3. Test Scenario Creation
4. Test Variable Identification
5. Test Value Profiling

Test Scenario Creation
- Determining how to test identified features (how to test)
- Analyzing use cases for each feature
- Defining (or reusing) one or more test scenarios for each feature

Example: Test scenarios for F1 & F2
TS1 – Verify that users are able to search rooms by specifying room type, check-in/check-out dates, and # of persons, using supported browsers and operation systems
TS2 – Verify that users are able to book found rooms by specifying the number of rooms
• **Test Variable Identification**
  - Deriving test variables and their values from test scenarios
  - Classifying test variables into two categories:
    - Functional Variables - represent the functional units/areas of the system under test
    - Environment Variables - represent the environment conditions affecting the execution of a function unit
  - Example: **Test variables and values for TS1 and TS2**

<table>
<thead>
<tr>
<th>Functional Variable</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoomReservation</td>
<td>SearchRoom</td>
</tr>
<tr>
<td></td>
<td>BookRoom</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoomTypes</td>
<td>Single</td>
</tr>
<tr>
<td></td>
<td>Suite</td>
</tr>
<tr>
<td></td>
<td>Family</td>
</tr>
<tr>
<td>OperationSystems</td>
<td>WinXP</td>
</tr>
<tr>
<td></td>
<td>Vista</td>
</tr>
<tr>
<td>Browsers</td>
<td>IE7</td>
</tr>
<tr>
<td></td>
<td>Firefox</td>
</tr>
</tbody>
</table>
Testing Process – Test Value Profiling

- **Test Value Profiling**
  - Associating test values with code coverage and defect count information by looking up the data from historical testing data repository
  - Example: Test value profile

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Values</th>
<th>Code Coverage</th>
<th># of defects found</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Method_1</td>
<td>Method_2</td>
</tr>
<tr>
<td>RoomTypes</td>
<td>Single</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Suite</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Family</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OperationSystems</td>
<td>WinXP</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Vista</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Browsers</td>
<td>IE7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Firefox</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
SV&V Testing Process Flow

1. Software Requirements
2. Historical Testing Data
3. Requirement Modeling
4. Test Case Generation
5. Test Execution
6. Builds

Flow Diagram:
- Requirement Modeling
- Historical Testing Data
- Test Case Generation
- Test Execution
- Builds

Arrows indicate the flow of data:
- Software Requirements → Requirement Modeling
- Historical Testing Data → Requirement Modeling
- Requirement Modeling → Test Case Generation
- Test Case Generation → Test Execution
- Test Execution → Test Execution Data (Coverage, # of defects, etc.)
- Build → Requirement Modeling
- Build → Test Case Generation
- Build → Test Execution

Integrations:
- Product Test (PT) Model
- Test Goals
- Source Code Analysis Report

ICSM'09 Edmonton 13
Testing Process – Test Case Generation

1. Variable Classification
2. Value Profile Updating
3. Test Result Analysis
4. Value Combination

Test Goals Achieved?

Yes -> Stop Generating Test Cases
No -> Test Case Suites

From Test Execution

Product Test (PT) Model
Source Code Analysis Report
Test Execution Data (Coverage, # of defects etc.)
Test Goals

To Test Execution

Historical Testing Data
Variable Classification

- Categorizing related functional variables into groups, called Test Frame Sets
- Each test frame set is passed to the Value Combination sub-process to generate test cases
- Test cases generated in each test frame set constitute a test case suite
Testing Process – Value Profile Updating

• **Value Profile Updating**
  – Updating current test value profile with the test execution data (e.g. code coverage, # of defects found, etc.) from the last iteration
  – Saving the updated profile to test data repository
  – Computing and updating the weight for each test value
  – The weight for a test value is determined based on the number of methods covered and number of bugs found by the test value
  – Example: **Test execution data and value weight**

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Code Coverage</th>
<th># of defects found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case</td>
<td>Method_1</td>
<td>Method_2</td>
</tr>
<tr>
<td>TC001</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>TC002</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TC003</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Values</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoomTypes</td>
<td>Single</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>Suite</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Family</td>
<td>0.75</td>
</tr>
<tr>
<td>OperationSystems</td>
<td>WinXP</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Vista</td>
<td>0.25</td>
</tr>
<tr>
<td>Browsers</td>
<td>IE7</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Firefox</td>
<td>1</td>
</tr>
</tbody>
</table>
Testing Process – Test Result Analysis

Test Result Analysis

- Analyzing test execution data to determine if the test goals are achieved
- If the goals are met, the process stops generating test cases; otherwise, it runs the **Value Combination** sub-process to generate an additional set of test cases
Value Combination

- Based on the updated value weights from **Value Profile Updating**, automatically adjusting the pre-defined combination algorithm to generate a set of test cases to achieve the test goals
- The combination algorithm could be any of the linear/multivariate statistical methods, genetic algorithm, and machine learning algorithms
- Example: **Generated test cases**
  
  TC003 – SearchRoom[Suite][2009-09-23][2009-09-26][1][IE7][WinXP]
  
  TC004 – BookRoom[2][IE7][WinXP]
Contributions & Summary

• Modeling requirements written in natural language into structured format to automate test case generation
• Utilizing historical test execution data and source code information during test case generation phase to improve effectiveness of test cases
• Generating test cases in an iterative manner to incrementally achieve test goals, resulting in smaller number of test cases
• Evaluating and adapting different test case generation algorithms (i.e. test value combination algorithms) for RIM SV&V testing
Using Dynamic Execution Data to Generate Test Cases

Rozita Dara
Shimin Li
Weining Liu
Angi Smith Ghorbani
Ladan Tahvildari

University of Waterloo
Canada

Research In Motion
Canada