

Adaptive Scenario-Based Testing Using UML

W.T Tsai, R. Paul*, Zhibin Cao, Bingnan Xiao, Lian Yu
Department of Computer Science and Engineering
Arizona State University Tempe, AZ 85287
*Department of Defence, Washington DC

Overview

- Introduction
- Testing with UML
- Scenario Specification, Analysis, and Patterns
 - Formalized Scenarios
 - Requirement Patterns and Verification Patterns
 - Case Studies
 - Evaluation
- Conclusion

Introduction

- Testing is important and will be even more important as we proceed to extreme programming or agile development
 - At each stage, the system needs to be tested, and regression testing applied to ensure that those unchanged parts are not adversely impacted.
- Testing is still the most effective way for quality assurance.

Introduction (cont.)

- Because we need to test often and the system keeps on changing, we must be able to develop test cases rapidly to test the new features, and we must identify those proper test cases needed to perform regression testing.
- In other words, test reusability is important. Several test reusability approaches are available:
 - Regression testing
 - OO test framework [Firesmith 1996, McGregor 1996, Poonawalla 1998, Tsai 2002a, Tsai 2002b]
 - Rapid generation of test cases/scripts using scenarios
 - Verification patterns
- We will focus on the last two topics here.

Testing with UML

- Software testing also need to be designed to be effective.
- We can derive system scenarios from the UML diagrams for testing.
- UML has 12 kinds of diagrams in 3 categories:
 - Structural diagrams
 - Behavior diagrams (use case diagrams, sequence diagrams, activity diagrams, collaboration diagrams, and state chart diagrams)
 - Model management diagrams.

Testing with UML (cont.)

- Use cases:
 - A collection of related scenarios that describe actors and operations in a system and can be organized hierarchically.
 - Use cases can be formalized (such as preconditions, operations, and post-conditions) and used for testing.
 - Based on use cases, sequence diagrams can be generated to show the events that external actors generate, their execution order, and inter-system events.

Testing with UML (cont.)

- But use cases may be too high level to derive test cases, so we need to transform a use case into scenarios, and then from a scenario to thin threads, and finally from a thin thread to test cases:

Use cases → Scenarios → Thin-Threads → Test cases

Testing with UML (cont.)

- Scenario:
 - (a scenario is) A **specific sequence** of actions and interactions between actors and the system under discussion. [1]
- Thin Thread:
 - (a thin thread is) A **complete trace** (E2E) of data/message using a **minimally** representative sample of **external input data** transformed through an interconnected set of system (architecture) to produce a **minimally** representative sample of **external output data**. The execution of a thin thread demonstrates a method to perform **a specified function**. [DoD 2000]

Scenario Specification & Analysis

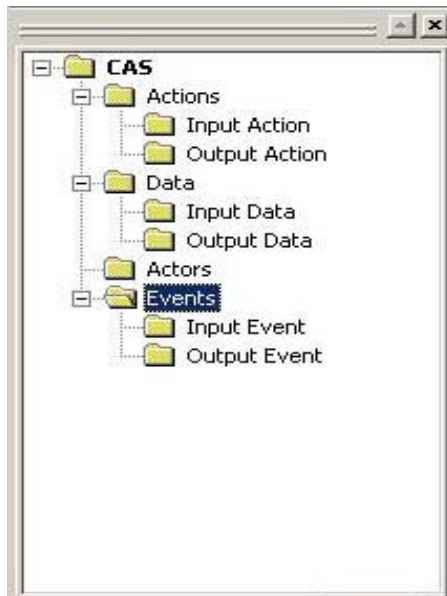
- Each scenario has the following information attached:
 - Input/Output: Inputs and expected outputs are identified for each scenario.
 - Grouping: Scenarios can be grouped into a hierarchical tree structure.
 - Dependency: Useful in regression testing and in impact analysis after system modifications.
 - Risk: Useful for scheduling and prioritizing various tests during the system development and maintenance .

Example: Car alarm system

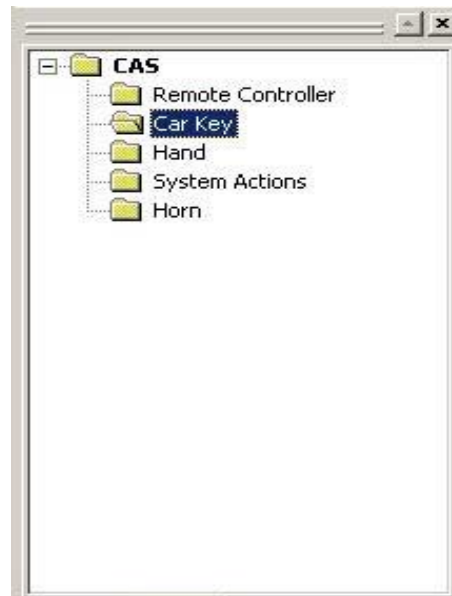
- Car alarm system description:
 - The system will alarm whenever the system detects a possible attack.
 - Users can use remote controller to arm/disarm the alarm system and to lock/unlock the doors.
 - User can use car key to lock/unlock doors, open trunk/hood, and ignite.
 - The alarm system will response according to the user actions and the car current status.

Example: Scenario Specification & Analysis (Group)

- System scenarios are annotated by identifying Configurations and Conditions
- System scenarios are grouped so that they can be analyzed together based on commonality



Configuration Group



Scenario Group



Condition Group

Example: Scenario Specification & Analysis (Dependency)

- Once identified, they can be used to perform dependency analysis:
 - input-output dependency,
 - input-input dependency,
 - output-output dependency,
 - condition-dependency or
 - control dependency.

Dependency Information:

The screenshot shows a 'General Information' dialog box with the following fields and options:

- Name:** [Empty text box]
- Created Date:** 2002-08-22 14:57:27
- Last Modified:** 2002-08-22 14:57:27
- Source:** Key unlock/open the door,
- Target:** Key unlock/open trunk, hor
- Description:** [Empty text area]
- Type:**
 - Function
 - Input
 - Output
 - Input/Output
 - Persistent Data
 - Execution
 - Pre-Condition
 - Post-Condition
 - Control

Buttons: OK, Cancel, Help

Example: Scenario Specification & Analysis (Ripple Effect Analysis)

- Based on the dependency information, we can do ripple effect analysis. Colors indicate the degrees of impact: blue indicates changed scenario, red direct impact, and pink indirect impact.

REA	Name	Creator	Create Time	Modifier	Moc
2	Key lock when open, horn sound If either of the doors is opened, turning on the car-key makes the alarm horn beep three times (at this time, the operations by the remote cor	admin	2002-08-02 00:14:11	admin	200
1	Key unlock/open the door, horn does n... The alarm system does not sound if one uses the car key to unlocked and open the door	admin	2002-08-02 00:21:03	admin	200
0	Key unlock/open trunk, horn does not s... The alarm system does not sound if one uses the car key to unlock and open the trunk	admin	2002-08-02 00:21:56	admin	200

Ripple Effect Analysis

Example: Scenario Specification & Analysis (Risk Analysis)

- Each scenario is assigned a value to define risks.
- Scenarios with high total risk number will be scheduled to be tested with high priority.

The screenshot shows a software interface for Risk Information. On the left, a tree view shows the following structure:

- General Information
- [-] Dependency Information
 - Dependency with other
 - Dependency with condit
 - Dependency with config
- [+] Test Information
 - Risk Information**

The main area is titled "Risk Information" and contains two tables:

Risk Category Master List:

Risk	
Safety	
Security	

Project Risks:

Risk	Value
Critical part	2
Dependency	1
Frequently used	3
Function	2
Reliability	1

Buttons at the bottom: OK, Cancel, Help.

Other Features

- Completeness and Consistence Analysis:
 - Analyze the requirement coverage of the designed scenarios.
 - Analyze if there is a conflict between two requirements.
- Test Generation:
 - Generate test cases by converting scenarios into thin-threads and attaching data to the thin-threads
- Scenario Templates:
 - Scenarios in or cross the projects can have the similar architectures.
 - It describes the scenario by the variant part and the invariant architectures represented by the template.
 - Changes are narrowed down to the variant part.
- Verification Patterns:
 - A pre-defined verification mechanism that can be used to verify a group of behavioral requirements that describe similar temporal pattern or cause-effect relations

Conventional Verification

- Verification has often been done in a case-by-case for each individual.
- Each new requirement requires a new verification implementation.
- If a requirement is changed, its verification procedure must be changed too.
- Implementing a verification procedure for each requirement can be expensive, and the code delivered can have a diversity of design and quality.
- As high as 70% embedded system testing effort focuses on debugging test scripts.

Adaptive Testing Using Scenario Templates & Verification Patterns

- We can generate test cases/scripts rapidly after system is changed with these features.
- Change of system may result in
 - Reusing the same template/pattern
 - In most cases, only need to change the parameters.
 - Identify a new template/pattern

Adaptive Testing Using Scenario Templates & Verification Patterns (cont.)

- **Applicability**
 - Scenario template/verification pattern are targeted for real-time/embedded systems.
- **Reusability & Extensibility**
 - Vertical
 - Existing templates/patterns can be reused and extended for new versions of the system.
 - Horizontal (cross-project)
 - Products of one product family can be useful for other product families. We have found patterns identified in medical devices can be useful telecommunications with minor modifications.

Requirement Patterns

- Requirement patterns:
 - A specific temporal pattern or cause-effect relation that can be used in representing a number of requirements.
- Representation:
 - A timeline format depicting the temporal relations of the events involved.
 - Texts format for more detailed explanations.

Scenario Templates

- As scenarios come from requirements, scenario templates are generated according to the requirement patterns.
- Scenario templates construct the abstract scenario architectures as the invariant logical parts for the scenarios from the requirement patterns.
 - Similar business applications share the business logic, which can be represented by scenario templates.
 - When template changed, all of the inherent scenarios will be changed accordingly.

Verification Patterns

- Even though a typical system has numerous scenarios, most of them can be classified into few scenario templates.
- It is possible to develop a verification pattern for each scenario template, by specifying the scenario templates and their corresponding verification patterns.

Verification Patterns (cont.)

- Verification patterns
 - A verification pattern is a pre-defined verification mechanism that can be used to verify a group of behavioral requirements that describe similar temporal pattern or cause-effect relations, or
 - A verification pattern is normally associated with a requirement pattern and used to verify all the requirements that can be classified as belong to that pattern.

Case Study: Requirement Patterns

- Requirement Patterns and their coverage of requirements (Guidant):

Pattern	Coverage
Basic requirement pattern	40
Key-event driven requirement pattern	15
Timed key-event driven requirement pattern	5
Key-event driven time-sliced requirement pattern	7
Command-response requirement pattern	8
Lookback requirement pattern	6
Mode-switch requirement pattern	8
Interleaving requirement pattern	6
Total	95

Case Study: Elevator Controller

- Three patterns covers about 95% requirements
 - Implicit-time pattern
 - Command-response pattern
 - Condition-event driven pattern
- 90% effort reduction

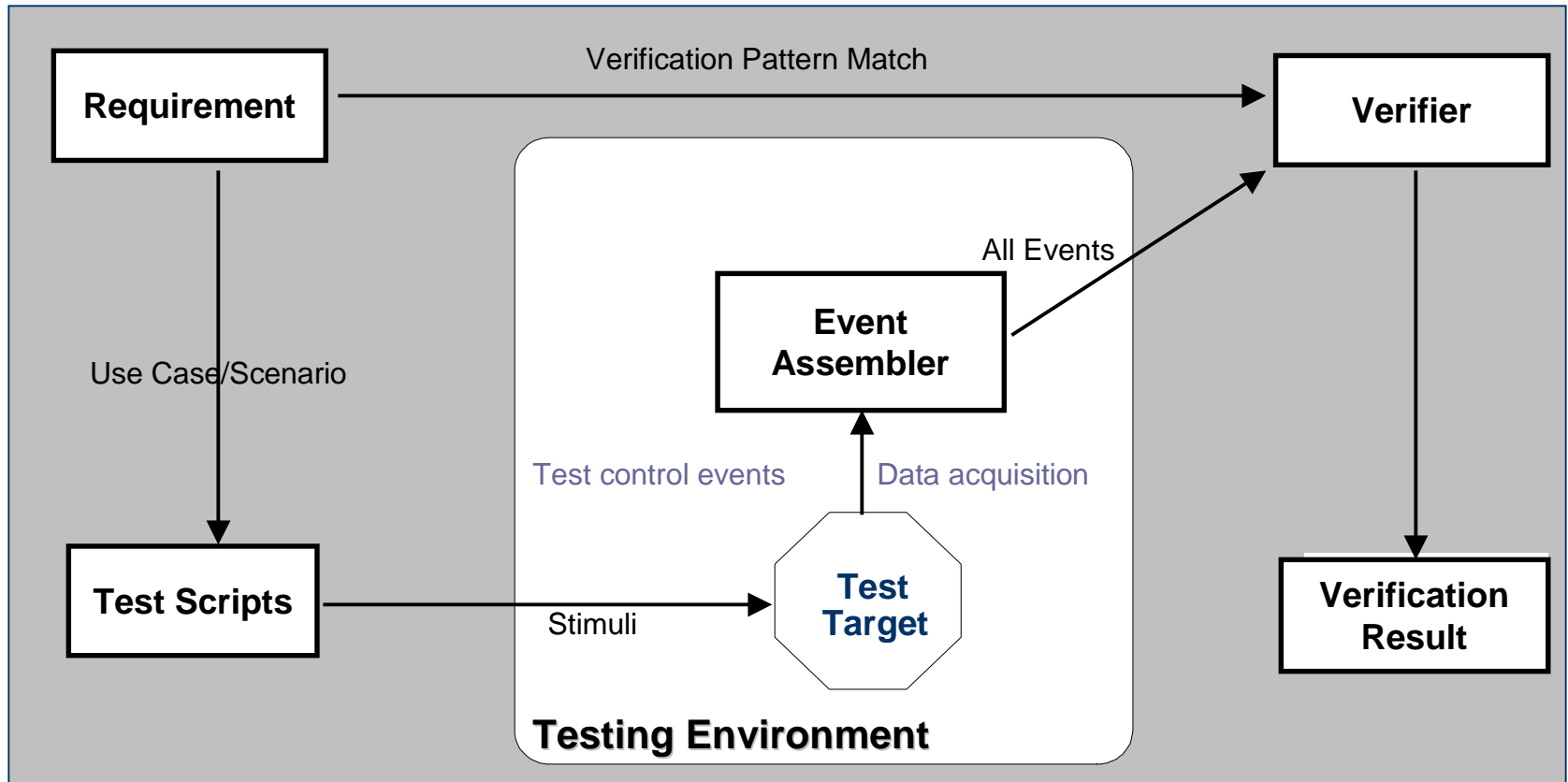
Case Study: Car-Alarm System

- Only four patterns can cover 100% requirements
 - Implicit-time requirement pattern
 - Explicit-time requirement pattern
 - Duration triggering pattern
 - Chain-reacting pattern
- Effort reduction about 90%

Verification Patterns

- General requirements
 - Application-independent.
 - Easy to adapt – platform independent.
 - Easy to expand and maintain.
 - Separate the test design and the test framework
- It is a part of the OO testing framework

Verification Patterns (cont.)



Evaluation

- Handling Changes
 - Adding Requirement
 - Classify the requirement, or create a new requirement pattern
 - Represent the requirement using appropriate pattern, or implement a special verifier for the requirement.
 - Removing Requirement
 - Delete it from the pattern verifier it belongs, or
 - Remove its specialized verifier from the test list.
 - Change in Requirement
 - *Timing change* - Change the timing parameters in the requirement representation
 - *Sequence change* - Change the sequence of the requirement representation, or Reclassify the changed requirement
 - *Action change* - Update the requirement representation using events that represent the new actions. Normally the requirement item stays with the same requirement pattern.

Evaluation (cont.)

	Feature Group 1 (no pattern)	Feature Group 2 (using pattern)
No. of Requirement Items	99	114
Use Verification Framework	No	Yes
Lines of Code (.h)	60,211	6,200
Lines of Code (.cpp)	76,412	10,100
Total Lines of Code	136,623	16,300
LOC / Requirement Item	1380	143

Verification Patterns – Case Study Results

			Experienced			Not experienced	
			Group A	Group B	Group C	Group D	Group E
Data contributor: Process X: without V-Framework Process Y: with V-Framework			Experienced in existing testing system	Experienced in the verification framework	Experienced in existing testing system and migrating to the framework	Learning existing testing system	Learning the verification framework
Process used			Process X	Process Y	Process Y	Process X	Process Y
Learning curve	Domain knowledge		Proficient	Proficient	Proficient	Novice	Novice
	Testing Techniques		Proficient	Proficient	Novice	Novice	Novice
Productivity	New Requirement		40 (10 ~ 50)	10 (4~50)	16(4~50)	90(80~100)	66(50~100)
	Requirement Changes	Timing Change	6(2~10)	4(2~8)	6(2~10)	20(8~40)	14(8~20)
		Action Change	30(10~50)	20(4~50)	25(4~50)	54(30~100)	36(20~100)
		Sequence Change	27(10~40)	8(4~20)	10(4~20)	42(20~60)	20(10~40)

Conclusion

- Integrated with UML
- Rapid & adaptive testing for real-time safety-critical applications.
- Keep communication and get feedback from industry
- Plan to develop a practical prototype

References

1. Craig Larman, *Applying UML and Patterns, an Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, 2002
2. W.T. Tsai, X. Bai, R. Paul, W. Shao, and V. Agarwal, *End-to-End Integration Testing Design*, to appear in Proc. of IEEE COMPSAC, 2001
3. X. Bai, W. T. Tsai, R. Paul, K. Feng, and L. Yu, "Scenario-based Modeling and Its Applications to Object-Oriented Analysis, Design, and Testing", Proc. of IEEE WORDS 2002
4. F. Zhu, "A Requirement Verification Framework for Real-time Embedded Systems", Ph.D. thesis, University of Minnesota, Minneapolis, MN 55445, 2002
5. D. G. Firesmith, "Pattern Language for Testing Object-Oriented Software", Object Magazine, Vol. 5, No. 9, Jan. 1996.
6. A. Cockburn and J. Highsmith, "Agile Software Development", Addison-Wesley, 1999
7. J. D. McGregor, and A. Kare, "Parallel Architecture for Component Testing of Object-Oriented Software", Proc. of Annual Software Quality Week, Software Research Institute, May 1996.
8. A.K. Onoma, W.T. Tsai, M. Poonawala, and H. Sukanuma, "Regression Testing in an Industrial Environment", Communications of the ACM, Vol. 41, No. 5, May 1998, pp. 81-86.
9. <http://www.extremeprogramming.org/>
10. <http://agilemanifesto.org/>