ECE750-Topic11: Component-Based Software

Evolution and Maintenance of CBSS

Ladan Tahvildari
Assistant Professor
Dept. of Elect. & Comp. Eng.
University of Waterloo
Outline

- Basic Concepts in Software Maintenance
- Laws of Software Evolution in CBS
- Challenges in Maintaining CBS
- UML-Based Testing of CBS
- Covering Some Research Papers
Types of Software Maintenance

- **Adaptive**
  - The changes that are triggered by an evolution of the environment of the system
  - 20 to 25% of the software maintenance effort

- **Corrective**
  - The changes that are triggered by a defect in the system
  - 20% of the software maintenance effort

- **Perfective**
  - The changes that are triggered by new users requirements or performance improvements attempts
  - 50 to 60% of the software maintenance effort
Testing Framework for Black-Box Testing of CBS

Diagram:
- Client
- Test Case Repository
- Testing Interfaces
- Normal Interfaces
- Component Developer
- Delivered Version
  - Test Case Analyzer
  - Test Specification
- Old Version
- new Version
What is Wrong with Black-Box Testing?

- Many faults may *be overlooked* or may *not be effectively detected* by black-box testing.

- *Complete functional testing is often infeasible* because of the complexity of the actual combination of functions present in a system.

- *Lack of accurate specification*
White-Box Testing

- White-box validation methods (also known as program-based testing methods, or structure based testing methods)

- They refer to the **systematic techniques for testers to design and generate test cases and data to achieve a certain test adequacy criteria** for a component based on its component program and structure.
What’s Wrong with White-ox Testing?

- *Can not be applied when source code is not available*
  
- Many white-box testing techniques *depend on instrumentation*, which can encounter great difficulty because of the heterogeneity

---

*UML-Based Testing of CBS*
Why UML?

- **Implementation Transparency**
  - UML provides high-level information that characterize component internal behavior.
  - UML also provides different levels of capacity and accuracy for component modeling.

- **Heterogeneity and Availability**
  - UML has emerged as industry standard for software modeling notations.

- **Evolvability**
Why UML?

- **Feasibility**
  - Provides different levels of capacity and accuracy for component modeling

- **Easy of Automation**
  - Many UML diagrams can be automatically processed
  - Test cases can be automatically generated
Collaborations

- Description of a collection of objects that interact to implement some behavior within a context

- *Describe the structure and behavior of a system*

- Graphical representation of a collaboration

- *Objects in a collaboration diagram are instances of classes in a class diagram*
Collaboration Diagrams

- The *objects* that are involved in an interaction and the structure of these objects

- Instances of allowable *sequences* of operation calls to an object

- The *semantics* of an operation

- The *operations* that are imported from other classes, thus enabling a collaboration with objects of the other class

- The *communication* pattern of objects in a collaboration

- The *execution* characteristics of objects
A Collaboration Diagram for an Operation
UML State Transitions and Events

- **Object State** combination of all attribute values and objects that the object contains

- **Dynamics of objects** are modeled through transitions among states

- **Event** is the specification of a significant occurrence that has a location in time and space
Test Case Generation for UML Statecharts

- Change event enabled transitions are used to define four levels of testing:
  - Transition Coverage Level
  - Full Predicate Coverage Level
  - Transition-Pair Coverage Level
  - Complete Sequence Level
Full Predicate Coverage

- **Boolean Expression:** An expression whose value can be either true or false.

- **Clause:** A boolean expression that contains no Boolean operators.

- **Predicate:** A boolean expression composed of clauses and zero or more Boolean operators.

- For each predicate $P$ on each transition, $T$ must include tests that cause each clause $c$ in $P$ to result in a pair of outcomes where the value of $P$ is directly correlated with the value of $c$. 
A Collaboration Diagram

<<external I/O device>>
:CardReader

1. Card Reader Input
   1.1A [Card Not recognized]: EjectCard
   2A.3: Eject
   2.6B.2: Confiscate

<<I/O device interface>>
:CardReader Interface

1.1: Card Data
   [Card recognized]
   1.2: Card Inserted

<<entity>>
:ATMCard

2.2, 2.6A.5: Card Data
2.1, 2.6A.4: Card Request

<<user interface>>
:CustomerInterface

2.6A.3: Pin Input
2A: Cancel Input

<<subsysten>>
:BankServer

2.5, 2.6A.8: Validate PIN
2.6B.1: Confiscate

<<state dependent control>>
:ATMControl

2.4, 2.6A.7: Card Data, PIN
2A.1: Cancel

2.6A.1: Invalid PIN Prompt
2.7: Display Menu
2A.2a: Display Cancel

2.6A.6: Customer Info

:ATM Transaction

2.3: 2.6A.6

1.3: GetPin

2.6A.1a, 2.6B.1a: Update Status

7/14/2005

ECE750-Topic 11
A Statechart

- **Idle**
  - Entry/Display Welcome

- **Waiting For PIN**
  - 2.4, 2.6A.7/2.5, 2.6A.8
  - 2.6A/2.6A.1, 2.6A.1a

- **Validating PIN**
  - 2.6/2.7, 2.7a

- **Waiting for Customer Choice**
  - 2A.1/2A.2, 2A.2a

- **Ejecting**
  - 2.6B/2.6B.1, 2.6B.1a

- **Confiscating**
Challenges for Maintenance

- When components are changed, how do we know the impact of the changes?
- How do we adequately maintain evolving component-based systems?
Changes in Collaboration Diagrams

- **Card Reader Input**
  - 1.1A: Card Not recognized: Eject Card
- **Card Reader Interface**
  - 1.2: Card Inserted
  - 2.6A.1: Cancel
  - 2.6B.1: Confiscate
  - 2.6B.2: Confiscate
  - 2.6A.2: Invalid PIN Prompt

- **ATMCard**
  - 1.4: PinPrompt
  - 2.2, 2.6A.5: Card Data
  - 2.1, 2.6A.4: Card Request

- **ATM Customer**
  - 2, 2.6A.3: Pin Input
  - 2A: Cancel

- **Customer Interface**
  - 2.3, 2.6A.6: Customer Info

- **Bank Server**
  - 2.5, 2.6A.8: Validate PIN
  - 2.6B[ThirdInvalid]: Third Invalid PIN

- **ATM Control**
  - 2A.1: Cancel
  - 2.6A.1: Invalid PIN Prompt

- **ATM Transaction**
  - 2.7a: Update Status
  - 2.6A.1a
Changes in Statechart Diagrams

- Idle
  - Entry/Display Welcome
- Waiting For PIN
  - 1.2/1.3
  - 2.4, 2.6A.7, 2.5, 2.6A.8
  - 2.6A, 2.6A.1, 2.6A.1a
- Validating PIN
  - 2.6, 2.6B, 2.6B.1, 2.6B.1a
  - 2.6B, 2.6B.1, 2.6B.1a
- Waiting for Customer Choice
- Ejecting
  - 2A.1, 2A.2, 2A.2a
- Confiscating
  - 2.6B, 2.6B.1, 2.6B.1a
Regression Testing for Corrective Maintenance

- Impacts of changes on control sequences
  - Collaboration Diagram
  - Statechart Diagram

- Impacts of changes on data dependencies
Data Dependencies from UML
Perfective and Adaptive Maintenance Activities

- **Constraint and Context**
  - Constraint: A boolean variable used to choose alternative paths
  - Context: A set of constraints associated with an execution path

- **Control Similarity Evaluation**
  - Contexts remain the same
  - Contexts with new constraints
  - Contexts with removed constraints
  - Contexts with new and removed constraints

- **Data Dependence Similarity Evaluation**
Collaboration Diagram

Context:
[valid] & [sufficient funds]
[Invalid]
[valid] & [Insufficient funds]
Collaboration Diagram

After

<<Business Logic>>
Withdraw
Transaction Manager

W3 [valid & sufficient funds]
W3A [Invalid]
W3B [valid & Insufficient funds]

W2 [Within daily limit]
W4

W1.1
W2
W3.1
W1.2

W5
W3A.1
W3B.1

<<Entity>>
Account

<<Entity>>
ATMCard

<<Entity>>
ATMTransaction Log