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
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
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

Changes in Statechart Diagrams
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]

Before

- W1
- W6
- W3A.2
- W3B.2

Transaction Manager

Account

ATM Transaction Log

July 14, 2005    ECE750- Topic 11
Collaboration Diagram

After

<<Business Logic>>
Withdraw
Transaction Manager

<<Entity>>
Account

<<Entity>>
ATMCard

<<Entity>>
ATMTransaction Log

W1

W2A.1
W3A.2
W3B.2

W2 [Within daily limit]
W3A
W3B

W5
W2A.1 [Exceed daily limit]
W3A.1
W3B.1

W1.1
W1.2
W3.1