Model-Based Testing with Use Cases

MBT: Generate *Tests* (and possibly test cases) from a Model

From Use Cases To Test Purposes

System Testing

- Fundamental truth about OO software testing: individual verification of components cannot guarantee a correctly functioning system.
  - We need to test the system against the requirements
  - Binder suggests 3 patterns for system-level testing
- UML 's use cases are typically assumed to capture the requirements when in fact they each capture a set of scenarios associated to some requirement(s)…
- Complete, consistent and verifiable requirements are necessary to develop an effective test suite ☹
  - Use cases are in English and thus not test-ready.
  - <<uses>> and <<extends>> are transitive: Binder suggests at least checking every fully expanded UC.
  - A scenario graph of a use case can help understanding the paths to test.

Binder 's Format for Testing Patterns

- The proposed format is:
  - Name: suggests a general approach
  - Intent: kind of test suite produced by this pattern
  - Context: When does this pattern apply?
  - Fault Model: What kinds of faults are to be detected?
  - Strategy: How is the test suite designed and coded?
  - Oracle: How can we derive expected results?
  - Automation: How much is possible?
  - Entry and Exit Criteria: Pre- and Post conditions to use
  - Consequences: Advantages and disadvantages

Pattern 1: Extended UC Test

- Intent: Build a system-level test suite by modeling *essential* capabilities as extended use-cases
- Context: Applies if most, if not all, essential requirements of the SUT can be expressed as extended Use Cases
- Strategy: A UC specifies a family of responses to be produced for specific combinations of external input and system state. This pattern represents these relationships as a decision table.
- To use eUCs we need to determine operational variables
- Operational variables are inputs, outputs, and environment conditions that:
  - lead to « significantly different » paths of a use case
  - abstract the state of the system under test
  - result in « significantly different » system responses

ATM Example

- Figure 8.4: Use Case Diagram for an ATM system
- Table 14.1 considers different resulting paths of a same use case in terms of input and output combinations
- This viewpoint is re-expressed in table 14.2 in terms of operational variables for each use case:
  - We need 4 variables to capture all combinations
  - We will discuss combinational models in detail later but we must understand NOW that the variants do not overlap!
    - » We have *partitioned* the input and output space successfully!
- Finally, we can *minimally* ensure every variant is made true at least once, and false at least once.
  - A true test case is a set of values that satisfies all conditions in a variant
  - A false test case has at least one condition false
  - see table 14.3

Use Case Diagram for ATM

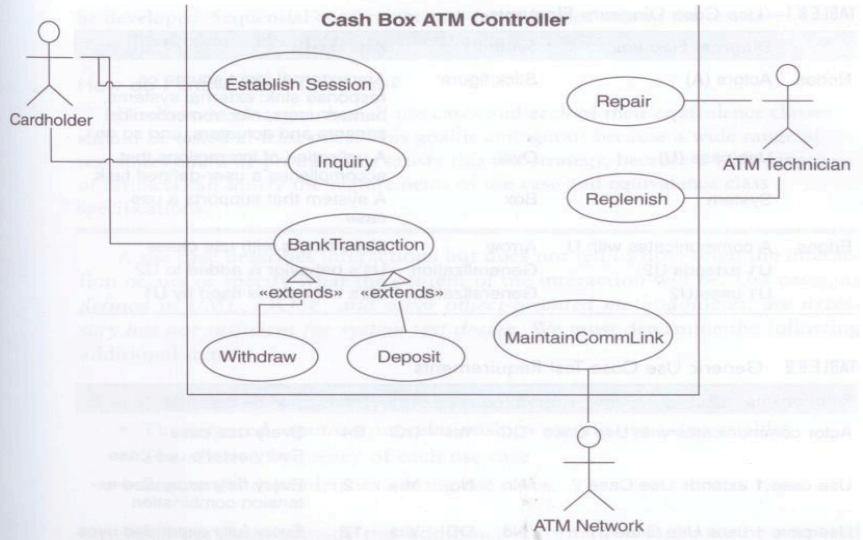

FIGURE 8.4 Use Case Diagram, cash box example.

I/O for Use Cases

TABLE 14.1 Some Use Cases and Scenarios for an Automatic Teller Machine

Use Case	Actor	Possible Input/Output Combinations
Establish Session	Bank customer	(1) Wrong PIN entered once; corrected PIN entered. Display menu. (2) PIN OK; customer's bank not online. Display "Try later." (3) PIN OK; customer's accounts are closed. Display "Call your bank." (4) Stolen card inserted; valid PIN entered. Retain card.
Cash Withdrawal	Bank customer	(1) Requests $50; account open; balance $1,234.56; $50 dispensed. (2) Requests $100; account closed. (3) Requests $155.39; account open; $150 dispensed.
Cash Replenishment	ATM operator with armed guard	(1) ATM opened; cash dispenser is empty; $15,000 is added. (2) ATM opened; cash dispenser is full.

Identifying Operation Variables

TABLE 14.2 Operational Relation for the Establish Session Use Case

	Operational Variables				Expected Result	
Variant	Card PIN	Entered PIN	Customer Bank Response	Customer Account Status	Message	Card Action
1	Invalid	DC	DC	DC	Insert an ATM card	Eject
2	Valid	Matches card PIN	Bank acknowledges	Closed	Contact your bank	Eject
3	Valid	Matches card PIN	Bank acknowledges	Open	Select a transaction	None
4	Valid	Matches card PIN	Bank does not acknowledge	DC	Please try later	Eject
5	Valid	Doesn't match	DC	DC	Reenter PIN	None
6	Revoked	DC	Bank acknowledges	DC	Card invalid	Retain
7	Revoked	DC	Bank does not acknowledge	DC	Card invalid	Eject

From Op. Vars to Tests (1)

TABLE 14.3 Minimal Test Suite for the Establish Session Use Case

	Operational Variables				Expected Result	
Variant	Card PIN	Entered PIN	Customer Bank Response	Customer Account Status	Message	Card Action
1	Invalid	DC	DC	DC	Insert an ATM card	Eject
1T	%*@#				Insert an ATM card	Eject
1F	Any *true* test for variants 2–7.					
2	Valid	Matches card PIN	Bank acknowledges	Closed	Contact your bank	Eject
2T	1234	1234	ack	CLSD	Contact your bank	
2F	Any *true* test for variants 1, 3–7.					
3	Valid	Matches card PIN	Bank acknowledges	Open	Select a transaction	None
3T	1234	1234	ack	OPEN	Select a transaction	None
3F	Any *true* test for variants 1, 2, and 4–7.					

From Op. Vars to Tests (2)

4	Valid	Matches card PIN	Bank does not acknowledge	DC	Please try later	Eject
4T	1234	1234	nack		Please try later	Eject
4F	Any *true* test for variants 2–7.					
5	Valid	Doesn't match	DC	DC	Reenter PIN	None
5T	1234	1134				
5F	Any *true* test for variants 1–4, 6, 7.					
6	Revoked	DC	Bank acknowledges	DC	Card invalid	Retain
6T	5555		ack		Card invalid	Retain
6F	Any *true* test for variants 1–6, and 7.					
7	Revoked	DC	Bank does not acknowledge	DC	Card invalid	Eject
7T	5555		nack		Card invalid	Eject
7F	Any *true* test for variants 1–6.					

Use Case Traceability Matrix

To wrap up the pattern:

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	. . .	Test 9999
Use Case 1				✓					
Use Case 2		✓			✓				
Use Case 3	✓					✓			✓
Use Case 4					✓		✓		
Use Case 5									
. . .									
Use Case 999		✓							✓

FIGURE 14.1 Use case test case traceability matrix.

eUCs: Typical Expected Faults

- domain faults: usually on boundary of conditions
  - » Card has expired
- logic faults: logic of specification is incorrectly coded
  - » Allowing a negative balance
- incorrect handling of don't cares
  - » In a high-interest savings account, there's a charge for a withdrawal only if your balance is less than 10K. In fact, this charge should be for everyone in this type of account.
- incorrect or missing dependency on pre-conditions
  - » a UC behaves correctly despite a violated pre-condition...
    - Expired card works…
- undesirable feature interactions (or is it scenario interactions)
  - » e.g., ATM shut downs while user is doing a transaction!
- incorrect output (e.g., wrong balance)
- abnormal termination (e.g., ATM eats your card…)
- omissions and surprises
  - » e.g., PIN does not get validated, all your accounts are zeroed…

eUCs: Other Fields

- Oracle: expected results by human intuition
- Automation: no… finding opvars is not necessarily trivial
- Entry criteria:
  - extended UCs must complete, consistent, verifiable (how to check?)
  - no *execution* of test cases at system level before its components have been tested (i.e., bottom up test execution…)
- Exit criteria: (as a % of completeness of req coverage)
  - XUVC = $\frac{\text{(\# of implemented UCs)}}{\text{(\# of required UCs)}} * \frac{\text{(Total \# variants tested)}}{\text{(Total \# of variants)}} * 100$
- Consequences:
  - Leaves out performance, fault tolerance, etc.
  - extended UC reduces to a decision table
  - Given no one agrees on level of abstraction of a UC, this pattern may be very dififcult to apply!

Pattern 2: Covered in CRUD

- **Intent: Verifies that all basic operations are exercised for each class in the system under test…**
- **Strategy:**
  - **Build a use case/class coverage table matrix**

	Class 1				Class 2				Class 3			
	C	R	U	D	C	R	U	D	C	R	U	D
UC1	√	√		√		√		√				
UC2					√	√			√	√	√	√

  - **C: creation; R: read, U: update, D: delete**
- **Exit criterion:**
  - **All basic operations of each class have been exercised at least once…**

Pattern 3: Allocate Tests by Profile

- **Intent: Allocate the overall testing budget to each use case in proportion to its relative frequency.**
- **Context: any time you use Pattern 1, especially in the presence of a combinatorial explosion of possible paths.**
- **Strategy: you must somehow (!) obtain an operational profile from the potential users. Then you merely sort.**
- **My comment:**
  - **frequency alone may not be sufficient: priority or importance (a la Boehm) must also be considered!**

Profiling

TABLE 14.5 Test Case Allocation for ATM

Use Case	Probability	Number of Tests
Establish Session	0.37	308
Cash Withdrawal	0.18	150
Checking Deposit	0.15	125
Savings Deposit	0.12	100
Funds Transfer	0.08	67
Balance Inquiry	0.06	50
Restock	0.02	17
Collect Deposits	0.02	17
Total	1.00	834

Implementation Specific System Tests

- **Several issues are typically downplayed if not ignored through use cases:**
  - **Configuration (wrt versions of s/w and h/w)**
  - **Compatibility**
  - **Setup/shutdown**
  - **Performance (see next slide)**
- **For Human Computer Interaction:**
  - **Usability, security, documentation, operator procedure testing**
- **Beyond system testing?**
  - **Alpha and beta testing (by independent volunteers), acceptance testing (by real customer), compliance testing (wrt standards and regulations)**

About Performance

- **We need quantitative formulations of performance reqs:**
  - **Throughput: number of tasks completed per unit of time**
  - **Response time: we need average and worst-case**
  - **Utilization: how busy is the system**
- **Other issues:**
  - **We need a worst case analysis**
  - **Performance modeling initially requires lots of magic numbers**
  - **Load testing considers how the system responds to increases in input events**
  - **Concurrency testing: load testing with concurrent events**
  - **Stress testing: rate of inputs exceeds design limits**
  - **Recovery Testing: testing recovery from a failure mode**
- **For real-time systems we must distinguish 3 types of events:**
  - **Repeating: must be accepted within a certain interval**
  - **Intermittent critical: aperiodic input with response within a fixed interval of time**
  - **Repeating critical: combination of 2 previous ones**