




**SOFT3302/3602
Software Quality Assurance**

Test Design


Michaelmas 2007 SOFT3302 1



Structured Testing Review

The Unbearable Hardness of Testing
A definition of testing
Structured testing
Quality requirement factors
Black-box testing
Functional hierarchies

Michaelmas 2007 SOFT3302 2



Lecture Outline

Models
Black box test design techniques

- Equivalence Partitions*
- Boundary Testing*
- Cause-effect Graphs*
- State-based Testing*
- Syntax Testing*
- Use Cases*
- Story-based Testing*

Michaelmas 2007 SOFT3302 3

Test Design

Cannot test without knowing what IUT must do
Complexity of software requires models
Test designer must use, revise and augment
Additional models may be needed

Michaelmas 2007 SOFT3302 4

Test Models

Testing is a search problem
Find the few input and state combinations
which will trigger failures and expose faults

- Most combinations will not*

Brute force is ineffective
Poking around leads to unwarranted
confidence
Testing must be systematic, focussed and
automated

Michaelmas 2007 SOFT3302 5

What Is a Model?

Capture essential relationships and features
Easier to develop and analyse

- Cf. *physical models*

Models have four main components

- Subject*
- Point of view/theory*
- Representation*
- Technique*

Michaelmas 2007 SOFT3302 6

The Rôle of Models

Model validation

- Attempts to establish confidence in the sufficiency of a formal abstract component with respect to its informal behavioural requirements

Verification

- Attempts to show that an implementation is correct with respect to its representation without executing it

Consistency checking

- Evaluated a representation as an instance of its meta-model

Michaelmas 2007 SOFT3302 7

The Rôle of Models

Responsibility-based testing

- Evaluates whether observable behaviour conforms to the representation

Implementation-based testing

- Evaluates observable behaviour with respect to a test model derived from an implementation

Product validation

- Evaluates conformance of observable behaviour to requirements

Michaelmas 2007 SOFT3302 8

Consequences

Test Model Characteristic	Effect of Informal Approach	Effect of Formal Approach
Unambiguous	Obstacle	Enabling
Complete		
No omissions	Neutral/Enabling	Neutral
Minimal	Neutral/Enabling	Neutral
Problem oriented	Neutral/Enabling	Neutral
Verifiable	Obstacle/Enabling	Enabling
Feasible	Obstacle/Enabling	Neutral/Enabling
Consistent		
Absence of conflicts	Obstacle	Enabling
Structural errors	Obstacle	Enabling
Behavioural errors	Obstacle	Enabling
Protocol errors	Obstacle	Enabling
Modifiable	Neutral/Obstacle	Enabling/Obstacle
Traceable	Obstacle/Enabling	Neutral/Enabling
Usable later	Enabling	Neutral/Enabling

Michaelmas 2007 SOFT3302 9

Test Design

A test design is not a test plan

- *Test plans contain scheduling, resourcing, &c.*

Test designs specify tests

Test designs map test cases onto components

Michaelmas 2007 SOFT3302 10

IEEE Test Design Specification

Identifier

Features to be tested

Approach refinements

Test identification

Feature pass/fail criteria

Michaelmas 2007 SOFT3302 11

Test Identification

A test plan may include all tests cases

- *Depends on size and complexity of components*

Test plans may give only lowest level test objectives

Test cases kept separately in a test-case database

All test cases must refer back to the test objective they implement

Michaelmas 2007 SOFT3302 12

Features To Be Tested

List of components
 For each component define test objectives

Michaelmas 2007 SOFT3302 13

Approach Refinements

The test plan identifies the test activity
 The test activity determines the approach

- A regression testing activity needs a regression testing approach*

Approach needs to be implemented using test design techniques

- Regression testing may require Functional Analysis, Cause-effect Graphing and Use Cases*

Michaelmas 2007 SOFT3302 14

Feature Pass/Fail Criteria

Determining test outcome may be very specific or complex

Generally this is something like 'a feature has been tested if all the tests have been run'

Some features may have multiple acceptable test outcomes

- Only a subset of the tests may need to pass*

Michaelmas 2007 SOFT3302 15

Black Box Test Design Techniques

- Equivalence Partitions
- Boundary Testing
- Cause-effect Graphs
- State-based Testing
- Syntax Testing
- Use Cases
- Story-based Testing

Michaelmas 2007 SOFT3302 16

Equivalence Partitions

Group input and outputs of a component into classes which can be treated similarly

- Assumed to be treated equally by component
- A value in a partition is taken to be representative all the entire partition

If the test passes for the representative value it's assumed that all other values in the group will also pass

Michaelmas 2007 SOFT3302 17

Example Partitions

Values between boundaries

- Numeric values <0, =0, >0
- Dates <2000, =2000, >2000; leap/non-leap years
- Months of various lengths
- Dates/times: week-days/-ends, in and out of hours

Objects with different state

- Empty/non-empty strings/lists/containers
- Files with different permissions
- Objects/references which exist/don't exist

Objects of different type

- String vs list; HDD vs FDD

Valid vs invalid input and output values

Michaelmas 2007 SOFT3302 18

Design of Test Cases

Test cases must exercise all partitions
 Each test case lists

- Inputs*
- Partition(s) to be exercised*
- Expected outcome of the test case*

Two approaches

- Separate test cases for each partition*
- Minimal set of test cases to cover all partitions*

Michaelmas 2007 SOFT3302 19

One-to-one versus Minimalist

One-to-one

- More time consuming as many more cases*
- More detail in the case of test failure*

Minimalist

- Less time consuming to prepare cases*
- Less detail on failure*
 - Testing still effective
 - May be hard to determine cause of failure
 - Makes debugging more difficult

Michaelmas 2007 SOFT3302 20

Boundary Testing

Tests around edges of partitions

- Multiple tests per partition on boundaries*

Retains assuming that similar values (values from the same partition) are treated similarly

Catches developer errors in choosing boundary values and conditions

Can be used to test multiplicities from UML diagrams

Michaelmas 2007 SOFT3302 21

Example Boundaries

The limits of the equivalence classes

- Zero
- Monday/Sunday, December/January
- MININT/MAXINT, MINFLOAT/MAXFLOAT
- Co-ordinate boundaries
- Wrap-around values: min/max, '99 vs '00
- Zero, one, multi-element strings/sequences

Michaelmas 2007 SOFT3302 22

Boundary Value Tests

Test on and next to boundaries

- Move value by smallest possible value
 - ±1 for ints, ± single char for strings, ±ε for doubles
- Non-scalar values only have =/≠

Boundaries may be open/closed

- Complete vs minimal testing

Michaelmas 2007 SOFT3302 23

Boundary Matrices

Vary values/conditions around one boundary

- Test boundary

Hold other values/conditions at typical values

- Choose values from equivalence class

Michaelmas 2007 SOFT3302 24

Cause-effect Graphs

Behavioural model of a component based on logical relationships between cause and effect

Caused expressed as boolean conditions on inputs

Effects expressed as boolean expressions on outcomes

Michaelmas 2007 SOFT3302 25

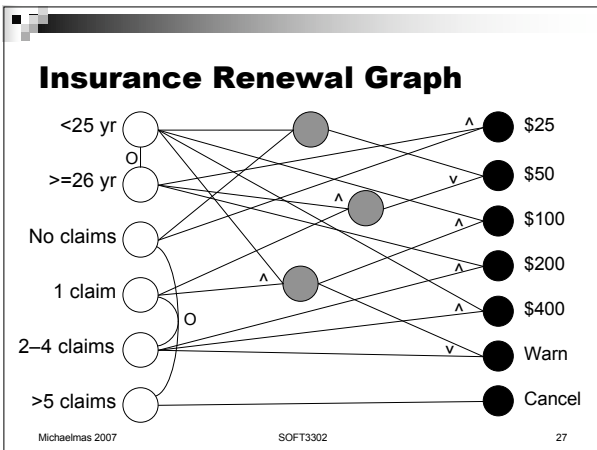
Cause-effect Graphs

Model of the internal condition processing of the SUT

Graph relating causes and effects using AND, OR, NOT, NAND, NOR relationships

Test cases produced to exercise each possible unique combination of inputs to the components being tested

Michaelmas 2007 SOFT3302 26



Notation

Boolean combinations

- AND \wedge
- OR \vee
- NOT $\sim \neg$
- NAND ∇
- NOR \sphericalangle

Constraints

- Exclusive, Inclusive, Only One, Require
- Masks

Michaelmas 2007 SOFT3302 28

Decision Tables

Need to cover all possible input variants
 In the insurance case that's $2^6 = 64$ cases
 Not all cases need to be tested

- In the insurance case we actually have 2 x 4 cases because of the constraints*

For each remaining test case construct a table showing causes and effects with one row per test case

Michaelmas 2007 SOFT3302 29

State-based Testing

Well suited to object-oriented software
 State-based testing is complex

- 8 2-state machines
- ~80 3-state machines
- ~2700 4-state machines
- ~275 000 5-state machines
- ~100 000 000 6-state machines

Michaelmas 2007 SOFT3302 30

State-based Testing

State models needed

- *Set of states*
- *Set of transitions*
 - From state to state, cause by an event, may resulting in an output action
- *The events which cause transitions*
- *The actions which may result from transitions*

Michaelmas 2007 SOFT3302 31

Basic State Machines

Process by which events are generated or queued is not a subject of the model

Events are recognised and transitions fire singly

No events other than those defined are recognised

The machine is only in one state at a time

Current state cannot be changed except *via* a defined transition

The model is static: no state, event, action or transition is added/removed as a result of execution

The details of the process by which an action is computed is not a subject of the model

Transitions do not take place over an interval

Michaelmas 2007 SOFT3302 32

Notation

Traditional

Structured Analysis

Unified Modeling Language

Michaelmas 2007 SOFT3302 33

Test Design

Test cases designed to exercise transitions
For each case specify

- Initial state*
- Input(s)*
- Output(s)*
- Expected resulting state*

Michaelmas 2007 SOFT3302 34

Switch Coverage

0-switch coverage covers single transitions
1-switch coverage looks at the results of 2 transitions
N-switch coverage looks at the results of N+1 transitions

Michaelmas 2007 SOFT3302 35

Invalid Transitions

Switch coverage only tests valid transitions
Also want to attempt invalid transitions
Need a state table

- Explicitly shows invalid transitions*

Any invalid transition which can be induced represents a failure

Michaelmas 2007 SOFT3302 36

α -states and ω -states

The α -state represents an object before its construction

The ω -state is reached after an object is destroyed

Useful for end-to-end testing of objects including constructors and destructors

These states are not interchangeable with the state diagrams initial and final states

Michaelmas 2007 SOFT3302 37

Syntax Testing

For systems which involve parsing structured documents we have the option of syntax testing

Use a model of the formally defined syntax of the inputs

- EBNF, DTD and so forth

Check that only correct formats are accepted and all invalid formats are rejected

Michaelmas 2007 SOFT3302 38

Test Design

Need to be able to generate documents

- Derive a syntax tree

Use the tree to derive valid and invalid documents

For valid documents

- Proceed through the tree generating possible options at each level

For invalid documents

- Proceed through the tree introducing single possible errors at each level to produce possible invalid documents

Michaelmas 2007 SOFT3302 39

Use Cases

Tests derived from use cases help uncover problems arising from the normal flow of use of a system

- May show it's not actually possible to move from one part of a case to another*

Use cases also help uncover integration errors

Michaelmas 2007 SOFT3302 40

Use Case Descriptions

Use cases have

- Preconditions*
- Postconditions*
- Flow of events*
 - A flow graph is developed from a sequence diagram for the use case
 - Typical flows
 - Atypical flows

Michaelmas 2007 SOFT3302 41

Test Design

Document paths which traverse the flow graph

Choose typical paths

- Use cases are usually documented to make these obvious*

Choose alternative atypical paths

For each case the path is identified along with expected inputs and outputs

All paths through the use case need to be explored

Michaelmas 2007 SOFT3302 42

Test Design

Single path can give rise to many test cases

- Other black box test design techniques may come in useful such as boundary testing or equivalence partitioning*

An overly large number of tests cases can arise

- Judgement needed*
- Risk (likelihood x severity) may need to be taken into account*

Michaelmas 2007 SOFT3302 43

Negative Tests

Attempted input not listed in use case

- Includes attempts to violate preconditions*

Attempted transitions/flows not lists in use case flow diagram

- Trying unexpected responses*
- Aborting a sequence*

Michaelmas 2007 SOFT3302 44

Use Case Revision

Test design may uncover problems with use cases

- May not be able to abort with entering some or all information*

Clear use cases help with test design

Bad/incomplete use case will be uncovered during test design

Michaelmas 2007 SOFT3302 45

Story-based Testing

More complex scenarios designed to exercise system

Attempts to move away from knowledge of implementation

Also called Soap Opera Testing

Michaelmas 2007 SOFT3302 46

Story Sources

Development team

Non-technical sources with domain knowledge

- Business sources*

From media

- Press, television, web*

From books, papers

Michaelmas 2007 SOFT3302 47

Test Design

Develop story variants

Add detail to add complexity

- Keep story realistic*

Identify use of the system in the story

- Define the interactions*
- Formalise the action sequences*

Follow practices as for use cases

Michaelmas 2007 SOFT3302 48

Final Comments

Test design is the formal documented process of developing test cases from models

Different techniques may be applied

- *Being thorough will likely mean multiple techniques*

Test design is a creative part of development

Many more tests than can be used will be developed

- *Subsets will need to be chosen judiciously*

Cannot be done in isolation without the development team and possibly the client

Michaelmas 2007 SOFT3302 49

References

[Bin99] Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns and Tools*, Addison-Wesley, 1999.

[Gal04] Daniel Galin, *Software Quality Assurance: From theory to implementation*, Pearson Addison-Wesley, 2004.

Some material adapted from 2004 SOFT3103 course.

Michaelmas 2007 SOFT3302 50
