Object-Oriented Integration Testing

Paul C. Jorgensen and Carl Erickson
Structure vs behavior

Structure
Needed for development
Only moderately useful for testing

Testing
Behavioral
Traditional software:

- Written in imperative language
- Described by functional decomposition
- Developed in waterfall life cycle
- Separated in three levels of testing
Imperative languages:

Description as a directed graph

Graph theory testing constructs
Event-driven nature of OO systems forces a “declarative spirit” on testing

Not evident at unit level (most OO programming languages are imperative)

Pronounced at integration and system levels
Functional decomposition

Deep implications for testing:

Emphasizes levels of abstraction

Creates questions of integration order

Stresses structure over behavior
Waterfall model

Described as a “V” where the development phases are at levels corresponding to the three testing levels

Sequential nature of model predisposes a bottom-up testing approach

Integration portion is driven by the functional decomposition tree

Preference of structure over behavior as the goal of integration testing
Traditional: Three levels of testing

- Unit
- Integration
- System
Unit:

Often not clearly defined

Unit Testing:

Unit is tested “by itself”

Adjacent software replaced by stubs and drivers

Goal: Verify that unit functions correctly
Integration testing:

Least well understood of the levels

Several views

Stress structure and interfaces, rather than behavior
System testing:

Conducted in terms of inputs/outputs that are visible at the port boundary of system
OO systems:

Much of the descriptive power of graph theory-based structural testing techniques will not be applicable

Individual methods within an object remain imperative
No clearly defined integration structure

Seen as an advantage: No longer natural to focus on structural testing orders
Shift to composition:

Objects may be correct by themselves, but errors may occur when they are composed
Five distinct levels of OO testing:

- Method
- Message quiescence
- Event quiescence
- Thread testing
- Thread interaction testing
Method → Unit level

Thread & Thread interaction → system level
Two remaining levels

Observation: Method executions are linked by messages, and quiescence provides natural endpoints (fig 1)
Definition: A Method/Message path (MM-Path) is a sequence of method executions linked by messages.

An MM-Path starts with a method and ends when it reaches a method which does not issue any messages of its own.
Definition: An Atomic System Function (ASF) is an input port event, followed by a set of MM-Paths, and terminated by an output port event
Example: ATM machine

Identifying MM-paths:

Digit entry (fig 3): A minimal MM-path

Pin entry (fig 4): An ASF, composed of six MM-Paths, an input port event and several port events

Simple transaction: A thread
Example of errors discovered by integration testing:

Lack of invocation of the superclass init method in the NumKeypad init method
Observations:

New constructs defined in article result in a unified view of OO testing, with fairly seamless transitions across the five levels
Figure 6:

- An object may be involved in many threads, and threads entail many objects

- An object may be involved in many ASFs, and an ASF may entail many objects

These two mappings guarantee that objects are integrated, and furthermore, the integration is grounded in behavioral rather than structural considerations
• The shortest MM-Path consists of two methods linked by one message

• The shortest ASF consists of one input event, a minimal MM-Path, and an output event

• An MM-Path is maximal within an ASF
The five levels of testing result in distinct, useful testing goals and bottom up testing order
The new constructs are directly usable as the basis for test coverage metrics, and they work well with composition