Testing UML Based Systems

Richard Warden
Testing UML Based Systems

Richard Warden
+44 (0)1684 772 691
warden@globalnet.co.uk
www.softwarefutures.ltd.uk
Introduction

- A practitioner’s view of UML
- Based on work performed with Sema4 Europe Ltd
  - Object Technology specialists
  - Integrate testing with development
  - Quality and process driven
- Three years experience:
  - Testing financial trading systems
  - Design of UML testing training
  - Development of methods and techniques
Aims

• To share experiences
• Topics
  – Status of UML in relation to testing
  – Systems testing
  – Acceptance testing
  – Summary comments on UML for testers
• Not covered
  – UML testing lifecycle and test processes
  – Explanation of UML
  – Class testing
Status of UML

- UML has been evolving over the last few years
- Developers are learning how to use UML
  - What is a good Use Case model?
  - How do they decide which diagrams to use?
- Testers are running to catch up:
  - Little to be found in the literature
    - 1000+ books on OO dev, 6 on OO testing
  - No established/institutionalised best practice
  - Knowledge being accumulated by individuals
  - Very little training on UML testing
What is UML?

- A set of models to describe both static and dynamic functional behaviour
- Non-functional behaviour?
- Scope is:
  - Requirements engineering
  - Analysis
  - Design
  - Implementation
  - Packaging
UML and object orientation

- UML includes OO models
- OO does not cover requirements engineering
- UML requirements models do not have to be implemented in OO
  - Hybrid systems
- Requirements model does not have to use OO concepts
Requirements testing – introduction

- Systems testing
  - Testing against the Use Case model
  - Are we building the system correctly?
- User/acceptance testing
  - Testing against the requirements definition
  - Uses the Use Case model
  - Have we built the right system?
Systems testing – first lessons

- Use Cases from external developers
- What to do?
- Review before test design
  - Based on 3Cs (complete, correct, consistent)
  - Past experience of reviewing documents
  - Some understanding of analysis modelling
- The “Useless Cases”
  - Use case model design was lacking
  - Use case process flows dense and impenetrable
  - The same problems as seen with traditional documents
A Use Case model
Use Case structure

- Motive – the reason why
- Trigger – a business event or another Use Case?
- Summary – overview of business function
- Requirements met – traceability, requirements elsewhere
- Assumptions – assumed state of the system
- Actors – those who trigger behaviour
- Pre-conditions – conditions to start the Use Case, origins?
- Constraints – Business (?) limitations
- Post-conditions – conditions to exit the Use Case, relayed?
- Basic flow – main processing flow
- Additional flows – optional flows
- Relationships to other Use Cases – extends and uses
**Initial problems**

- Written in narrative English
- A form of pseudo code
- Difficult to understand
  - Flowcharting to understand what was happening
- Difficult to cross check elements of the Use Case
- Lack of supporting artefacts to explain the processing
Heuristics needed

• Heuristics provide the rules for reviews and inspections
• Where rules are broken there must be a valid reason
• Rule breaking increases complexity and/or reduces understandability of the system
• **Ignoring rules reduces testability**
Heuristics development

- Adapt the rules we know
- Develop a structure
  - Semantics and syntax
  - Traceability
  - Design heuristics
  - Consistency between related artefacts (see later)
  - 3Cs
Use Case heuristics

Problems to look for:
- Monolithic Use Cases
- Use Cases that are verbs
- Too many Use Cases
- Controller Use Cases
- Imbalance of abstraction
- Too much detail
- Attribute-based Use Cases
- Covariance in Use Cases
- Incorrect relationships
- Unstable, non-extensible
- Actors that are passive

Observations:
- too much behaviour in one place
- single piece of behaviour
- lack of abstraction
- all behaviour is extended
- analysis/design cross-overs
- a scenario not a Use Case
- not function-based
- must be justified
- uses and extends confused
- ask What if? questions
- do not trigger behaviour
Related artefacts?

- In addition to Use Cases we may have
  - Scenarios
  - Statechart diagrams
  - Sequence diagrams
  - Activity diagrams
- State behaviour commonly overlooked
State behaviour example

- Four trader states
- Four trading system states
- Four contract states
- 64 combinations of states
  - Which combinations are permitted, need testing?
  - What depth of testing?
Trading system statechart

- **System not trading**
  - Entry: Broadcast message
  - Do: Disable off-market
  - Exit: Broadcast message
  - Event: None

- **System pre-market**
  - Entry: Broadcast to members
  - Do: Enable enter offer
  - Exit: Broadcast to members
  - Event: Market end time

- **System suspended**
  - Entry: Broadcast message
  - Do: Freeze all transactions
  - Exit: Unfreeze transactions and broadcast message
  - Event: None

- **System trading**
  - Entry: Broadcast to members
  - Do: Enable all off-market
  - Exit: Broadcast to members
  - Event: Market end time

- **Market activation time reached**
- **Market end time**
- **Abnormal trading activity**
- **Trading behaviour normal**
Statechart heuristics

Problems to look for:
- Are events realisable?
- Are all events in the Use Cases?
- Are guards in the Use Cases?
- Are transition actions in the U-C?
- Are entry/exits actions in the U-C?
- Are activities in the Use Cases?
- Is state generalisation needed?
- Is state aggregation needed?

Observations:
- → how do we reproduce them?
- → missing, overlooked events
- → missing, overlooked guards
- → missing, overlooked actions
- → missing, overlooked actions
- → missing, overlooked activities
- → high number of combinatorial states
- → simultaneous state activation
Systems test design

- With a good Use Case model
  - Good decomposition for test design
  - Test by Use Case, relationships?
- Large variation in Use Case sizes
  - Suspend/unsuspend trader - 15 lines of flow
  - Contract admin – many hundred lines of flow
- How to approach naturally complex Use Cases
  - Test selection, e.g. equivalence classes
- Data modelling?
- Non-functional tests?
Acceptance testing

- **UML does not capture requirements**
  - Business/high order concepts
  - Business processing
  - Requirements definition
- UML describes how the system functions
- UML does not describe why it functions that way
- Acceptance testing requires a different approach
  - What are the business patterns of behaviour
  - How can they be modelled?
Acceptance test patterns

- Patterns of trading
  - Trader to trader open Repo
  - Trader to trader closed Repo
  - Trader to trader standard general collateral Repo
  - Trader to trader non-standard general collateral Repo
  - Trader to market open Repo
  - Trader to market closed Repo
  - Trader to market standard general collateral Repo
  - Trader to market non-standard general collateral Repo
Instance of a pattern

Start pattern 1
Trader A creates offer using contract type 23
Trader B creates counter-offer changing Rate
Trader A creates counter-offer changing Period
Trade cleared through SEGA
Trader A creates trade
Trader B accepts offer
End pattern 1
Activity diagram for test pattern

```
Trader A
  Create offer
  Accept offer

Trader B
  Create CO
  Accept offer
  Reject offer
  Create offer

Trader C
  Create CO
  Cease trading

```

© Software Futures Ltd 2000
UML for testers

- UML needs support in three key areas:
  - Acceptance testing
  - Data modelling
  - Non-functional requirements
- Good UML helps test analysis and design:
  - Range of behaviour described
  - Diagrams are superior to words
  - Checking inter-related behaviour
Summary

• Analysts need good training and mentoring
• Testers need to develop inspection guides for all UML models, in conjunction with analysts
• Testers can develop their own techniques using UML diagrams, e.g.
  – Test scenarios for system testing
  – Activity diagrams for testing patterns
  – Statecharts for high level dynamic behaviour
Testing UML Based Systems
Richard Warden

Richard Warden has about 25 years IT experience ranging from development analyst, designer and programmer to test, project and quality management. He has worked on mainframe defence support systems, mini and micro based business systems, CAD systems and more recently client-server systems in financial applications. For the last nine years he has acted as an independent consultant, Software Futures Ltd, performing testing and other assignments for clients in UK and mainland Europe.