

First given at the BCS SIGIST Conference

11 February 2003

Challenges of Testing UML Based Systems

Richard Warden

Software Futures Ltd
2 Waterloo Way
Bredon
Tewkesbury
Glos.
GL20 7NA
GB

p +44 (0) 1684 772 691
f +44 (0) 1684 772 639
e info@softwarefutures.ltd.uk
w www.softwarefutures.ltd.uk

Aim and background



- Feedback from industrial experience.
- Working with UML since 1997.
- Major test projects.
- Development and delivery of test methods with Sema4.
- Training development and delivery with Isabel Evans, Unicom and Electromind Ltd.

UML Testing Challenges



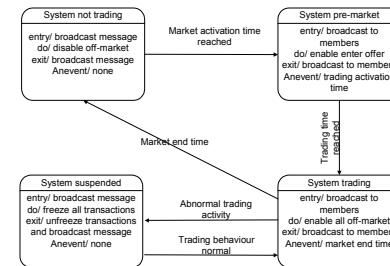
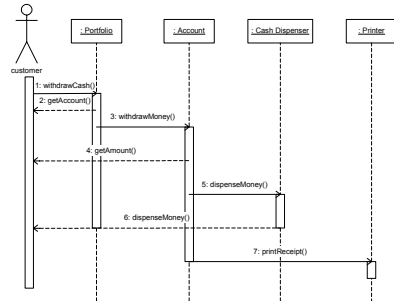
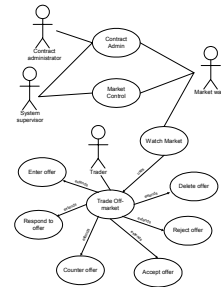
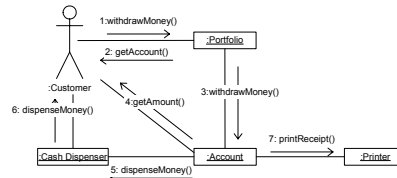
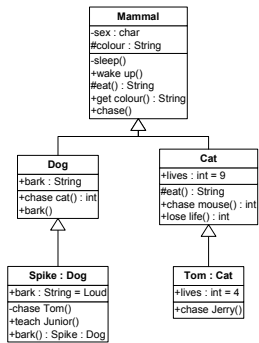
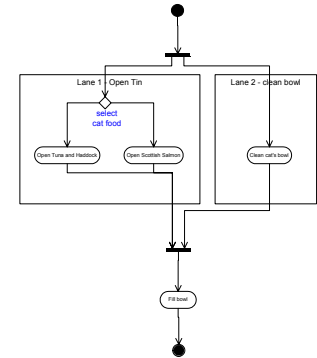
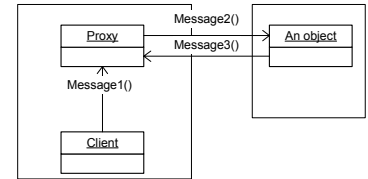
- Read and understand it.
- Identify good from bad UML.
- Use it to design effective tests.
- Judgements are the real challenge.

Unified Modelling Language



- The UML 1.4 specification is over 550 pages long.
- Four layer metamodelling architecture.
- Judgements:

- Which models do we use and where?
- What rules and standards do we need?
- Where do we get the wisdom from?

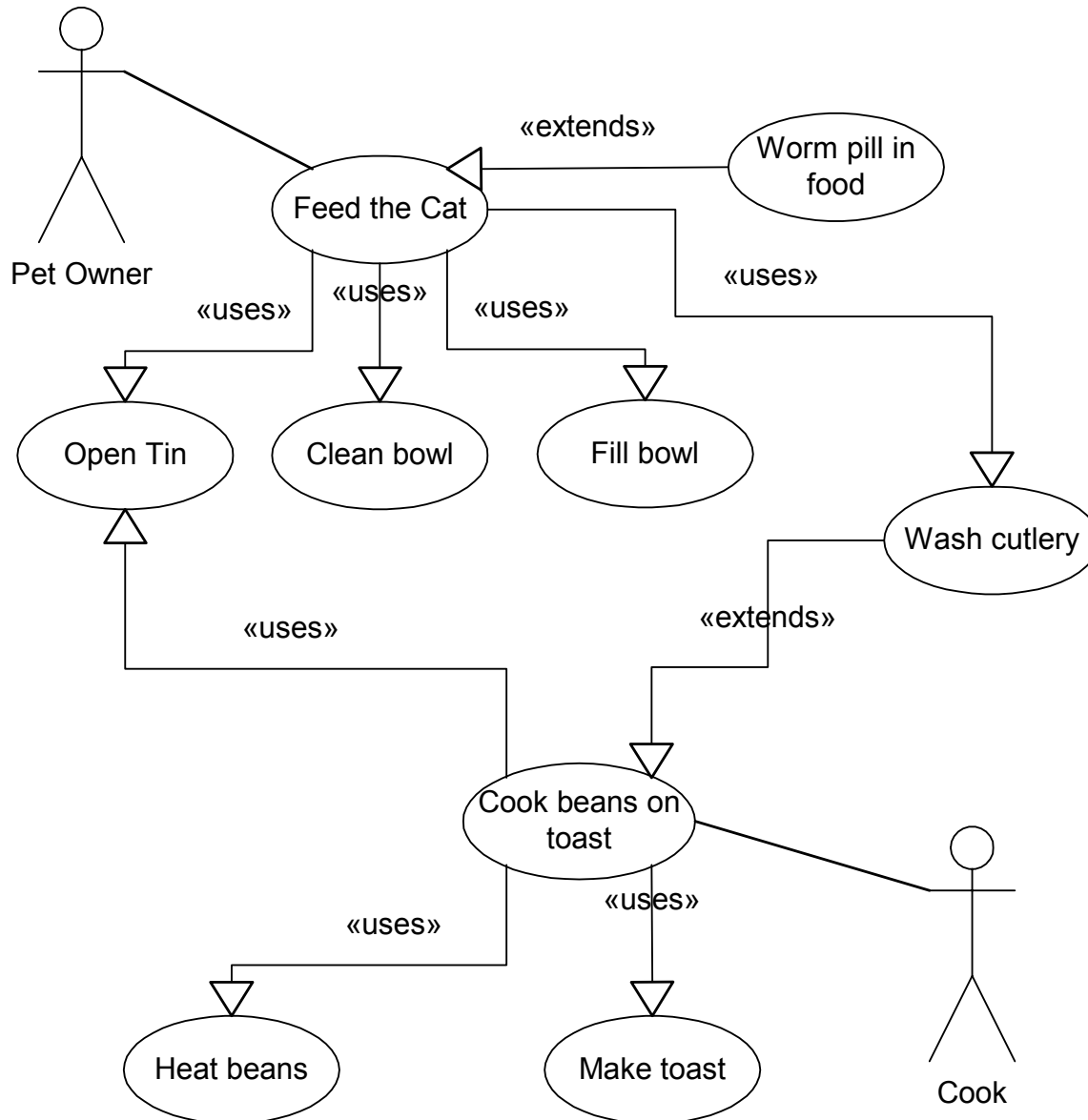


- For functional behaviour.
 - *Use Case model – user requirements and analysis.*
 - *Class diagram – class specifications & relationships.*
 - *Sequence diagram – time-based communication.*
 - *Collaboration diagram – messaging between objects.*
 - *Statechart diagram – states and events of objects.*
 - *Activity diagram – actions, links and control flow.*
 - *Component diagram – software components for building.*
 - *Deployment/package diagrams – distribution architecture.*
- Performance, security, usability, reliability, testability.....?
- UML includes OO models, but UML-based systems do not have to have OO implementation.
- Systems and acceptance testing may not require OO knowledge.



- Black box testing.
- Use Case model and associated diagrams.
 - *Use Cases (mandatory).*
 - *Scenarios.*
 - *Statechart diagrams.*
 - *Activity diagrams.*
 - *Sequence diagrams.*
- Verification – testing implementation of the system.
- Validation – user/acceptance testing with the model.

Simple Use Case model



But how well is it written?



- Use Case model.
 - **Decomposition** into Use Cases.
 - Size, complexity and detail.
 - **Relationships** between Use Cases.
 - Uses, extends and includes.
 - Control flow complexity.
 - Consistency of pre and post conditions.
- Poorly designed Use Case models are a major threat.
 - *Poor functional cohesion.*
 - *Overly complex relationships.*
 - *Contains design/implementation detail.*

Testability review criteria



- Is the diagram understandable with syntactic and semantic conformance to UML?
- Does the diagram represent all the behaviours and properties in the user requirements? Are other diagrams needed?
- Does the Use Case model contain any properties that make testing unnecessarily difficult or impossible? The model should adhere to design heuristics and exceptions should be documented. Non-functional objectives should not be overlooked.
- Is the Use Case model logically consistent with related artefacts, e.g. activity, statechart and sequence diagrams?
- The Use Cases may be packaged for implementation and the interactions between packages should be small and obvious so that incremental testing can be achieved.
- The 3Cs of Completeness, Correctness and Consistency are useful to consider.



- Motive or Goal.
- Trigger.
- Summary.
- Requirements met.
- Assumptions.
- Actors.
- Pre-conditions.
- Constraints.
- Post-conditions.
- Basic processing flow.
- Additional flows.

- Use Case heuristics.
 - *are the requirements met in the Use Case flows?*
 - *to what actors do triggers relate?*
 - *are the actors real?*
 - *are the assumptions valid?*
 - *are the pre and post conditions testable?*
 - *are the constraints met?*
 - *are the flows complete, correct and consistent?*

Which models are needed?



- Supporting artefacts.
 - are the business functions complex, would scenarios help?
 - do actors show state behaviour?
 - do use cases show state behaviour?
 - are their complex flows, difficult narrative descriptions?
 - are activity diagrams needed to expand flows and show synchronous behaviour?
 - are sequence diagrams needed to show temporal behaviour?



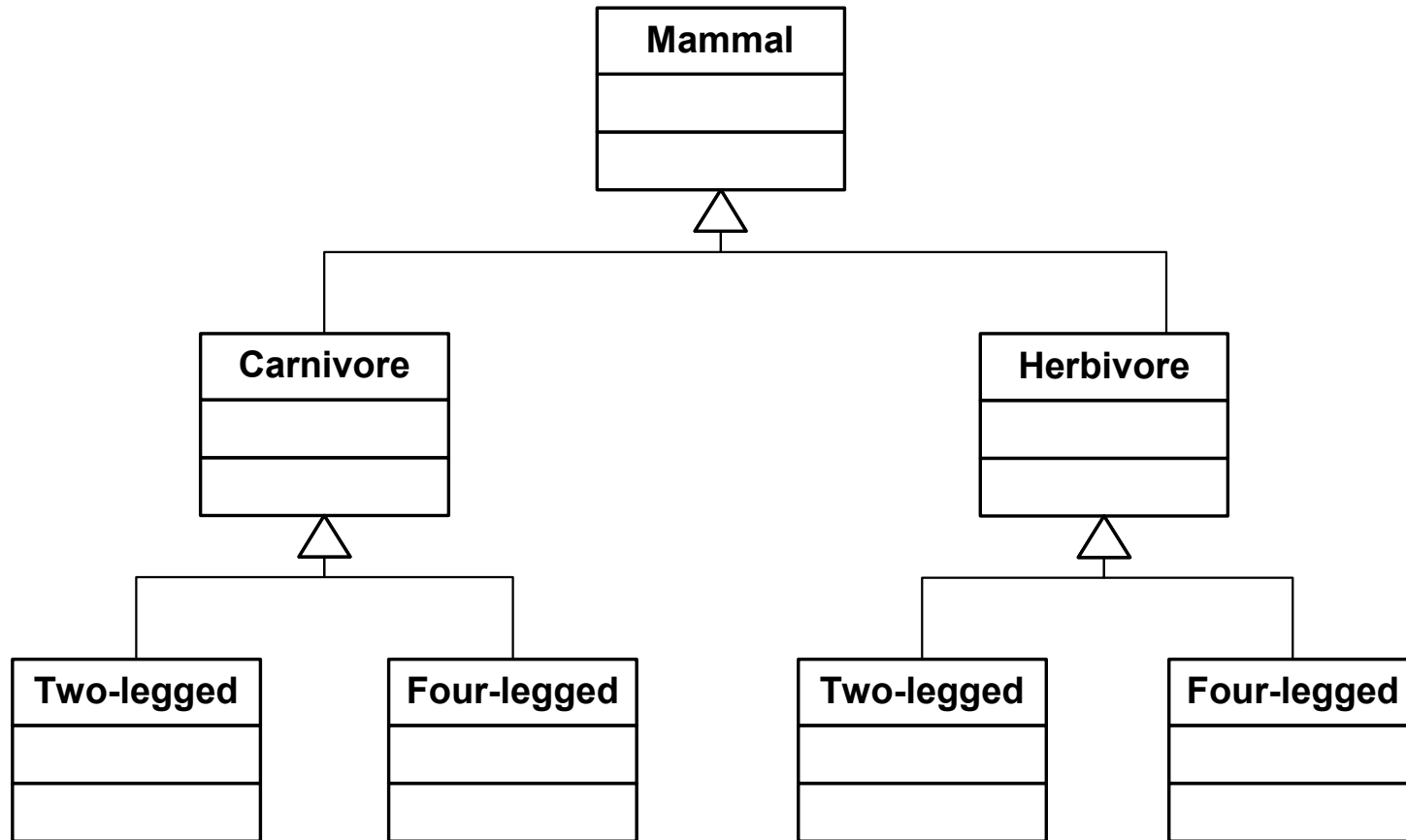
- Review OO designs.
 - Understand error types.
 - Develop checks for them.
- Develop test strategy for structural tests.
 - Understand test design issues.
 - Cope with incremental development.
 - Difficulty of testing frameworks.

Classification problems

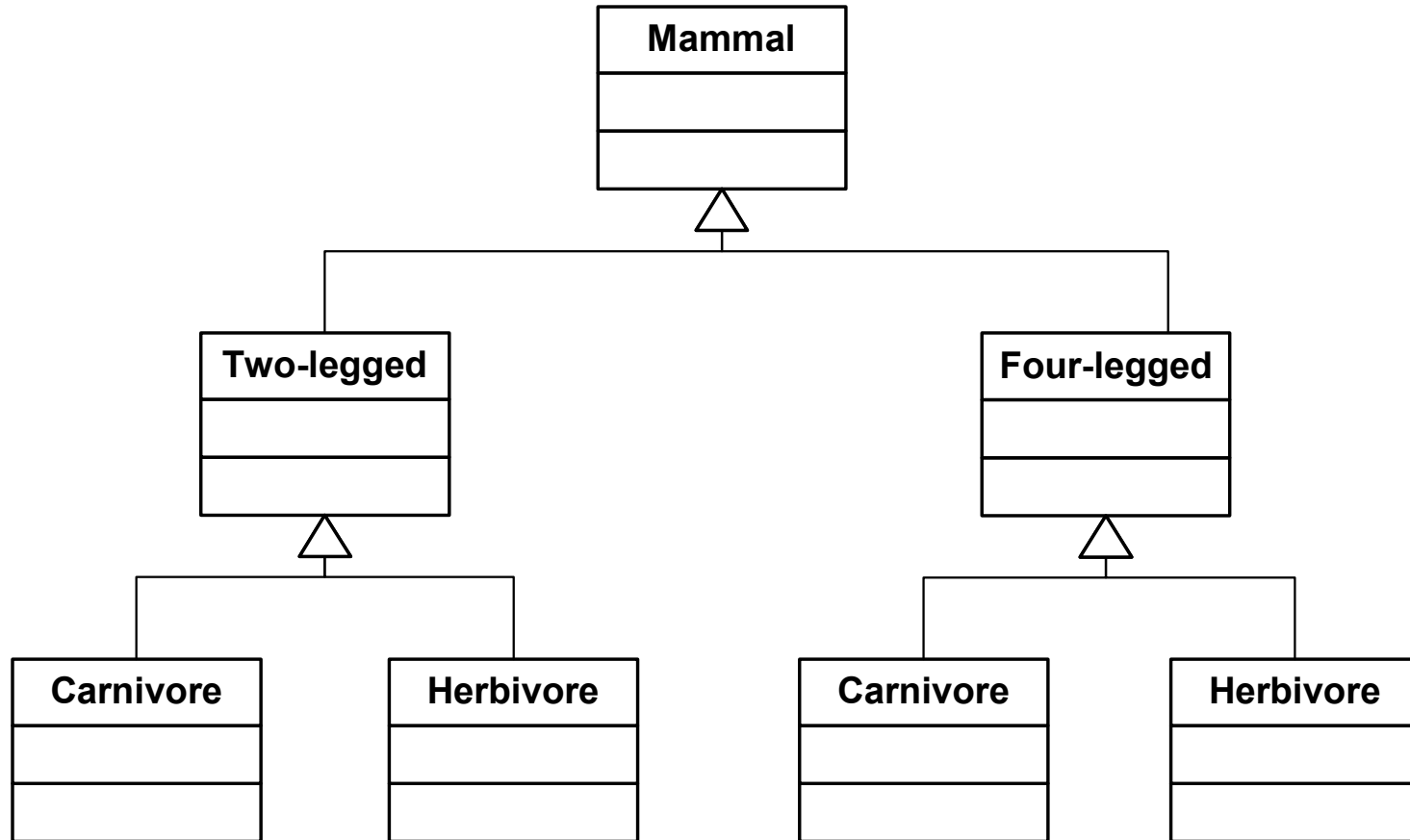


- Exercise:
 - Draw a class structure showing the parent class Mammal.
 - Child classes describing Herbivores and Carnivores.
 - Child classes that can be Two-legged or Four-legged.

Solution A



Solution B



Solution C - multiple inheritance



Work it out for yourself!

- Not strict inheritance – rule breaking.
- Generalisation and specialisation are confused.
- Potential for inheriting non-applicable methods.
- What about pre and post conditions?



- Reviewing class models and associated dynamic models.
 - *Class design heuristics (Riel).*
 - *Class relationships.*
 - *Inheritance structures, e.g. single or multiple.*
 - *Methods and interfaces.*
 - *Collaboration between objects - messaging.*
 - *State behaviour of objects.*
 - *Sequencing – timing relationships.*

- Visibility:
 - *Hidden methods and data, but need to be visible to testers.*
 - *Methods to return internal state of objects.*
 - *Develop a parallel test architecture.*
- Inheritance and polymorphism:
 - *Hierarchical inheritance tables.*
 - *Orthogonal arrays (Copeland).*
 - *Test patterns (Binder).*

Where to model non-functional behaviour?



- ISO 9126 Quality Characteristics
- Reliability
 - *Maturity*
 - *Fault tolerance*
 - *Recoverability*
- Usability
 - *Understandability*
 - *Learnability*
 - *Operability*
 - *Attractiveness*
- Efficiency
 - *Time behaviour*
 - *Resource utilisation*
- Maintainability.
 - *Analysability.*
 - *Changeability.*
 - *Stability.*
 - *Testability.*
- Portability.
 - *Adaptability.*
 - *Installability.*
 - *Co-existence.*
 - *Replaceability.*

See BCS SIGIST standards working party on non functional testing
www.testingstandards.co.uk.

Implications for testing - general



- UML models a rich variety of behaviour at many levels of abstraction.
- It can remove many traditional sources of error.
- Creates its own errors – may be too rich!
- Different models give testers different ways to test.
- Prioritise UML models for review and test:
 - *Use Case + Activity + State.*
 - *Class + Collaboration + State + (Sequence).*
 - *Clusters of Classes.*
 - *Components.*
- Implement UML in phases that map to your priorities.

Implications for testing - reviews



- Effective providing well done.
- Planned and managed.
- Use and develop UML heuristics and checklists.
- Involve developers, analysts and users as well as testers.
- Review as early as practical.
- Consider use of an Oracle.
- Review incrementally.
- Phase in starting with key documents.
- Select appropriate review technique and formality/ceremony.
- Avoid the A-Z trap; running out of time halfway through.

Implications for testing - OO



- Objects and classes encourage high functional cohesion. Avoids traditional error sources such as temporal and logical cohesion.
- Encapsulation leads to good coupling; implementation is hidden and behaviour invoked by protocols and messaging. Prevents access to private data and implementation code; no common or blank data.
- However hidden attributes and methods are more difficult to test.
- Functional tests can be designed separately and run before structural implementation tests.
- Interfaces are a traditional source of errors. Objects with many methods sending/receiving many messages increases the interface complexity.
- Methods are usually smaller than traditional procedural code, reducing code complexity.
- Changes to a class can be inherited by other classes; minimising code changes and chances of error. But any errors may be inherited requiring greater regression testing to detect misinheritance.
- Polymorphism requires careful testing to ensure new, changed or over-riding methods do not produce unexpected results.



- The basic unit to test is the Class.
 - However class interactions may make this difficult.
 - May need to test clusters of classes or develop a substantial test harness.
- New risk areas for testers:
 - Frameworks can be very difficult to test.
 - OO does not make poor designers good ones.
 - Designers and programmers who have a traditional mindset leading to misuse of OO. E.g. God classes and container classes that do not reflect domain classes.
 - Violation of OO principles; classes not related by the Is A rule, public access to private data and confusion of associations.
 - Lack of understanding of good OO design; e.g. too many over-riding methods in child classes.

- Bashir I & Goel A.L., Testing Object-Oriented Software; Life Cycle Solutions, Springer-Verlag, 1999.
- Binder R.V., Testing Object-Oriented Systems; Models, Patterns and Tools, Addison-Wesley, 2000.
- Booch G, Rumbaugh J & Jacobson I, The Unified Modeling Language User Guide, Addison-Wesley, 1999.
- Fowler M & Scott K, UML Distilled: Applying the standard object modeling language, Addison-Wesley, 1997.
- Jung D.C., Hsia P & Gao J, Testing Object Oriented Software, IEEE Computer Society, 1998.
- McGregor J.D. & Sykes D.A., A practical Guide to Testing Object-Oriented Software.
- Muller P-A, Instant UML, Wrox Press Ltd, 1997.
- Riel A J, Object-Oriented Design Heuristics, Addison-Wesley, 1996.
- Schneider G & Winters J.P., Applying Use Cases: a Practical guide, Addison-Wesley, 1998.
- Siegel, S., Object Oriented Software Testing, John Wiley & Sons, 1996.
- Taylor D.A., Object Technology: A Manager's Guide, Addison-Wesley, 1998.
- Whitmire S.A., Object Oriented Design Measurement, John Wiley & Sons, 1997.
- UML 1.4 .pdf download from www.omg.org.
- See also Cetus web site www.cetus-links.org.



For more information about our UML testing services please visit our web site

www.softwarefutures.ltd.uk