

Focus on UML Testing Strategies

Isabel Evans - IE Testing Consultancy Ltd
Richard Warden - Software Futures Ltd

As new software development methods are introduced, we must examine how we carry out software testing to understand whether our well-tried testing methods are still appropriate, whether we need a new testing strategy for this new development method, or whether a mix of new and old will be best. An example of a new approach over recent years has been the introduction of UML (Unified Modelling Language) often as an adjunct to OO developments. In this series of articles we will look at UML testing strategies, and we will provide some of the answers to two frequent questions:

- How does the use of UML affect testing?
- How can testers use UML?

Let us start with three important points:

1. UML is relatively new.
2. It provides developers with great flexibility in how they use it.
3. There are no off-the-shelf testing strategies.

For these reasons, our experience is that you have to work out a testing strategy specific to a project using past experience and best practice. How will you do this?

In this first article, we will show you how to set about designing your test strategy. We will assume that you intend to develop a test strategy and a document following the IEE 829 Test Documentation Standard Test Plan; in this article we will assume that we will only discuss those parts of the test strategy where we have found specific lessons for UML Testing Strategies. In other respects you will follow a normal IEE 829 test plan structure.

Setting goals for the test strategy

With any test strategy, you will find it useful to use a goal-based approach. Determine the goals of the test strategy and have them agreed with the project manager, developers and users before the project gets underway. These goals must complement those of the other project staff; goal setting is a good way of finding out whether other project staff really know what they are expected to achieve. Working on goals with the other project members helps in many ways. One advantage it has is to help contain the wonderful natural human optimism that a new tool or method will solve all our project problems; in this case that by picking up a new development language it will solve past problems.

Note: UML will not turn poor analysts and programmers into good ones; but used properly it can make good ones better.

DO THIS!

*Focus on the goals for the project and for testing, and test they are SMART goals –
Specific, Measurable, Achievable, Realistic, Time-bound*

Doesn't UML provide a strategy?

UML is a strategy free language. It is a set of models used for requirements engineering through to design, coding, packaging and deployment. It does not describe processes or lifecycle models – they have to be developed separately. The design and architectural models are object-oriented; the higher-level requirement models can be but are often not.

Note: UML is relevant to Non-OO as well as OO projects.

It is quite feasible to develop requirements using UML and implement them without using OO.

UML says nothing about testing.

The de facto lifecycle model with UML is the Rational Unified Process or RUP. Of US origin it describes testing as the design and execution of tests after the Inception phase. There are no testability inspections or reviews by testers of models during Inception or later – a significant weakness.

Identify UML usage

Your first task is to identify the extent to which UML is being used on a project.

One risk, especially on an early UML project, is common to the adoption of any new language, method or tool; that having learnt the language we only use a sub-set of it, and that not very well. A great plus of UML is that it is a rich language with many static and dynamic models that, when used properly, can give very good behavioural descriptions of a system. These models complement each other and enable much crosschecking to find errors at the review stages, where the economic leverage is greatest. The project will have greatest benefit if the team uses UML models comprehensively.

DO THIS!

Ask - how much is UML/OO and how much traditional development?

Identify the UML and OO models that testers are going to have to work with.

Identify the models the developers are going to use

If they propose not to use certain models that may be useful to testers, for example by allowing cross check reviews, ask why.

The next task is to capture the proposed development lifecycle and determine who is doing what testing. Identify the test processes you want to implement and want built into the project lifecycle. Using the V model, i.e. test early and test often, will be economical, and its spirit can be used in prototyping and evolutionary lifecycles as well as in traditional lifecycles. Here, you will have a great emphasis on reviewing **requirements models**, which in UML are **Use Case models** and associated artefacts such as **statechart diagrams**, **scenarios** and **activity diagrams**.

DO THIS!

Build time and resource for these into your planning and your strategy

Carry out a risk analysis on UML and application knowledge

Next, you should perform a risk analysis focusing on two key questions:

1. *To what extent is the application understood?* In other words, is this application new to the developers and/or the business, with all the risks associated with that scenario?
2. *What level of maturity/capability have the developers achieved in using UML?* In other words, how likely is it that mistakes will be made?

Of course, there are even greater pressures if the team is using a new language to build a new application type – double trouble! At minimum testers need a clear expectation of how developers plan to use UML. Some process review mechanism is needed because in practice the planned approach could break down. This is a risk to document in the test strategy, the mitigation being to have the processes reviewed.

DO THIS!

If UML is new to the team, assume that the development processes will be insecure and quite error-prone and that part of the tester's remit is to identify error-prone processes as early as possible.

Include Requirements Reviews in the Strategy

Especially if the team has low UML capability it is essential that testers be allowed to review/inspect requirements models as they are generated.

For example, on one project we reviewed the first 20 Use Cases of 190 and found a high error rate; on extrapolation we anticipated over 2000 errors in the full Use Case model. Feeding this info back caused changes to the way the developers worked. Imagine the outcome if so many errors went downstream; they would be built into the system's design and coding and would be a nightmare for testers.

This type of results can only be achieved by applying a strong review process on requirements models as early as possible. You need very probing rules and heuristics to drive the reviews and identify the design errors.

The test strategy and your planning need to allow time and resource for these reviews.

By design we do not mean the application's design, but the design of the requirements model itself. The requirements model has to be designed and if analysts break those design rules they may create major errors that end up in the architectural and systems design. At this point you are reviewing the requirements model for **Testability**, and not second-guessing the analysts' models. If testability is poor then there are three outcomes:

1. The developers will embed these errors into their systems design and on into the classes and code and will not find them because they will not test for them.
2. The testers will have inputs for test analysis, design and execution that are lacking in detail and are ambiguous, so it will be far harder for them to find these errors.
3. The downstream transmission of requirements/testability errors can jeopardise the project.

A major testing goal in any UML project concerns risk assessment, prevent or amelioration around the requirements model.

DO THIS!

Include requirements reviews in the strategy

Common issues to address in your strategy

A major issue in many projects concerns requirements capture and subsequent engineering. Richard has worked on projects where requirements were initially captured using non-UML models. Then the Use Case model was developed from them. This may lead to a number of problems:

- It is difficult to ensure traceability between initial capture and the Use Case model.
- It is difficult to match the use of the high level requirements and the UML requirements (Use Case) model at different levels of testing
- UML does not lay down a specific standard for use

Without traceability between artefacts it is difficult to identify when requirements change, and if those changes have been reflected in the artefacts.

DO THIS!

Include traceability rules in the strategy including traceability between UML and non-UML artefacts with related information

An example is that for acceptance and user testing the high level requirements are the authority and the Use Case model is tested against them – this is **validation** (Have we built the correct system?). For this we have found that standard test techniques such as business flow and transaction processing work well. However, we have also found that testers can use UML models to develop acceptance tests. For systems testing, the Use Case model is the authority – this is **verification** (Are we building the system correctly?). Here, standard techniques such as equivalence partitioning, boundary analysis and cause-effect are very useful.

The point here is that testers need a good working knowledge of UML not just to be able to understand what the analysts and developers are producing, but also to review and critique their models, and to use UML to model tests. Developing this knowledge is another strategy element.

DO THIS!

In the “Staffing and Training Needs” section of the strategy / test plan (IEEE 829) note that UML knowledge is required or must be acquired

Another issue is that developers have great freedom on how they use UML, which raises the question of standards. A project needs standards on how UML will be used to ensure a consistent approach. If standards are lacking then each analyst and developer is likely to develop their own personal standard/process; we end up in a CMM level 1 situation. In the absence of any off-the-shelf standards, another strategy consideration is how they will be developed and agreed across the project groups.

Some specific examples are with requirements models:

Much of the content of a Use Case is narrative language description of processing flows. This presents the same problems as a “traditional” requirements document. It can be written just as badly. Faced with a poor Use Case, you may have to convert narrative to flowcharts to tease out problems. Use Cases do not guarantee better results unless used properly with other UML models.

If the Use Case model is badly designed then all important pre and post conditions will be vague and ambiguous. This creates a big problem for testers, as they cannot test for them properly. Remember that UML can describe various dynamic behaviours. Use Case models are one view. UML contains statechart diagrams as another dynamic representation but the general advice in the UML community is that developers only use them when there is complex state behaviour. In short this is a licence not to use them. Our experience is that any non-trivial Use Case(s) point to *state behaviour* that we need to test. Developers need to be made aware of this. For acceptance testing, state behaviour is not described at the individual Use Case level but across many Use Cases that form a specific part of the application. Analysts need to be aware that they must model this behaviour. Producing statechart diagrams has one good benefit. If the Use Case model is badly designed the analysts will have great trouble modelling the state behaviour because they will not be able to specifically exactly all the conditions for entry, execution and exit of a state – and why? Because poor Use Case design meant that pre and post conditions were never fully defined.

DO THIS!

Set standards in the project for the use of static and dynamic representations of the requirements, so that use cases are supported by acceptance level and program level state diagrams

At this point another strategy consideration is that of education of the testers and the analysts. They have to learn what testers want from UML and how testers will use their models. One way to achieve this is to send both analysts and testers on UML testing training. We presented a course with such an audience. The analysts found it very helpful and it aided communication between two groups that are often at loggerheads; the “my group is OK and yours is not” syndrome.

DO THIS!

Include analysts and developers in test training programmes to ensure a common message

Conclusion

Because UML is simply a set of models to describe a system’s behaviour any project using it must already be able to:

1. Develop strategies, set goals and determine key metrics.
2. Define its lifecycle and processes for development and testing and have staff trained and experienced.
3. Perform a risk analysis; UML is both a threat and an opportunity.

If these elements are not in place UML will not solve the problem.

In our next article, we will discuss developing a robust Testability review process for UML requirements models to gain the leverage of testing very early in the lifecycle.

The authors:

Richard Warden is an independent consultant and founder of Software Futures Ltd. In 27 years experience he has worked in development, support, quality management and test in both technical and managerial roles. Since starting his company in 1991 he has added extensive experience in areas of job design and work motivation, process analysis and development, and goal setting and metrics design. His UML experience includes nearly two years as a testing consultant for the Swiss Stock Exchange working on UML-based trading systems. Subsequently he spent another 15 months as principal test consultant for Sema4 Europe, an object-technology company. In conjunction with Isabel Evans they developed one of the first testing training courses on Systems and Acceptance Testing of UML-based Systems. More information can be found at www.softwarefutures.ltd.uk.

Isabel Evans is Principal Consultant at IE Testing Consultancy Ltd. She has nearly 20 years in software quality and testing, as a practitioner, consultant and trainer. She is an active member of the BCS SIGIST Standards Working Party, particularly working on the usability, installability and conversion techniques. More information can be found at www.ietesting.co.uk.