ATAC

ITEA 2
INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Model-based testing in modern agile software development – How to integrate it into the development process?

Project number:	ITEA 2 10037
Edited by:	Matti Vuori, TUT
Date:	2014/03/18
Document version no.:	*1.0*

Table of Contents

•

1. Introduction

As we try to bring model-based testing (MBT) to the mainstream, we need to more address the issues of integrating the tasks and practices of MBT into the real-life systems development practices. Here, we try to provide some rough descriptions and principles for it. The goal is to share an overview of the issues so that tool and process developers can take the right things into consideration.

This report mostly includes process-related issues and for example, competence-related issues are left out for now.

Nowadays it is often assessed that testing in agile development is quite developer-centric and low level. There is a need to emphasise system level testing more. That itself brings changes to the approaches of testing. Some of the things that we document here are caused by the fact that MBT is usually “heavy” system level testing. But above and besides that, it has some unique practices that need extra consideration. Some of them may have equivalents in traditional test automation, but even then we need to map them into the processes using their concrete names.

Note that the descriptions of the practices are somewhat simplified and do not cover all variations – they just aim to point out the essential principles.

As attachment we have included mind-maps that present many attractions and obstacles to MBT. Most of them are generic and apply to many development paradigms, but some relate closely to agile. In the report we will look into some of those.

2. Role of MBT in development of various systems

MBT doesn’t need to be a do-it-all methodology. For various domains it can have a targeted purpose, which of course doesn’t need to be definitive and restrictive. But it gives guidance about where to start and where to emphasise MBT. We like to see testing as a rich activity where there are many (but not too many) approaches that complement each other. That means that even when MBT is heavily used, there is room for other kinds of test automation, not to mention manual testing.

In Table 1. we present some rough ideas about the role of MBT in various domains.

Table 1. Role of MBT in various domains

	Machine control systems	Industrial automation	Communication infrastructure	Mobile device app suites	Business information systems	Web applications
Description	Control systems such as in work machines, tractors.	Automation for e.g. process plants.	Network systems, protocols	Suites and single apps for phones and tablets.	Serious information systems used in management, governance.	Individual web applications.
Essential	Safety critical, high robustness. Software and hardware under test.	Safety critical, high robustness. Software and hardware under test.	Reliability critical, high robustness. Software and hardware under test.	Priorities vary highly. App interaction.	Business critical. System integration.	(Varies)
Low level design	Systematic. Interaction diagrams, state machines.	Systematic. Interaction diagrams, state machines.	Systematic. Interaction diagrams, state machines. Standards and specifications based.	Adhoc. No modeling.	Adhoc. No modeling.	Adhoc, platform templates. No modeling.
High level design	Requirements-based. Architecture-driven. V-model.	Requirements-based. Architecture-driven. V-model.	Requirements-based. Architecture-driven.	UX-driven. Agile.	Business process, business requirements driven.	Agile. UX-driven.
V&V	Standards important. Many levels (overall system, hw, electronics, sw). Formal verification has a role.	Standards important. Many levels (overall system, hw, electronics, sw). Formal verification has a role.	Standards important. Many levels (overall system, hw, electronics, sw). Formal verification has a role.	Internal standards. Platform requirements. Traditional sw testing.	Traditional sw testing. High test levels emphasised and systematic (system, acceptance).	Traditional sw testing.
Existing modeling	System architecture. Component state machines. Interactivity diagrams. Fault trees. FMEA. Use cases.	System architecture. Logic diagrams. Interactivity diagrams. FMEA. Use cases.	System architecture. Protocol stacks. State machines. Interactivity diagrams. FMEA. Use cases.	User stories.	Business process models (flows). Use cases.	User stories.
Existing modeling skills	Engineers (good).	Engineers (good).	Engineers (good).	Sw developers (varies)	Architects (good)	Sw developers (varies)
Mandatory reqs for modeling	Safety standards. Revieability.	Safety standards. Revieability.	?	(None)	(None)	(None)

	Machine control systems	Industrial automation	Communication infrastructure	Mobile device app suites	Business information systems	Web applications
Mandatory reqs for testing	Safety standards. Revieability. Traceability. Documentation.	Safety standards. Revieability. Traceability. Documentation.	Meeting standards' requirements.	Platform owner / store requirements.	(None)	(None)
Good features for modeling	Management of complexity.	Management of complexity. Reuseability with templating.	Completeness, detail. Automatic test generation from protocols.	UX-driven. Fast. Reuseability.	Business process - driven.	UX-driven. Fast.
MBT SWOT						
Strengths	Cultural compatibility. Perceived benefits. Mature culture for QA.	Cultural compatibility. Perceived benefits. May be mandatory. Mature culture for QA.	Cultural compatibility. Perceived benefits. Mature culture for QA.	Perceived befefits to testing interactions.	?	Versatile culture, open for new ideas.
Weaknesses	Tool adaptation still underway.	Tool adaptation still underway.	Done already.	Fast development, product changes.	No clear point of utilisation. Needs research and tool development.	No clear need for it. No good examples. Needs resources. Needs systematic action.
Threats	Need compatibility with dev. tools	Need compatibility with dev. tools	(None)		Must have integration in development environments.	Even traditional testing weak.
Opportunities	Simple tools for divided testing tasks.	Simple tools for divided testing tasks.	?	Long term testing. User story based modeling.	Test creationg from business process models. Use case testing.	Automatic UI testing.
Key MBT selling point	Use MBT to manage risks	Use MBT to optimize realiability	Use MBT to optimize realiability	Use MBT to validate that your app suite really works after changes	Use MBT to validate reliability of transactions	Use MBT to test user stories automatically

3. Obstacles and attractions of MBT

In *Figure 1* and *Figure 2* we present mind-maps that present many attractions and obstacles to MBT.

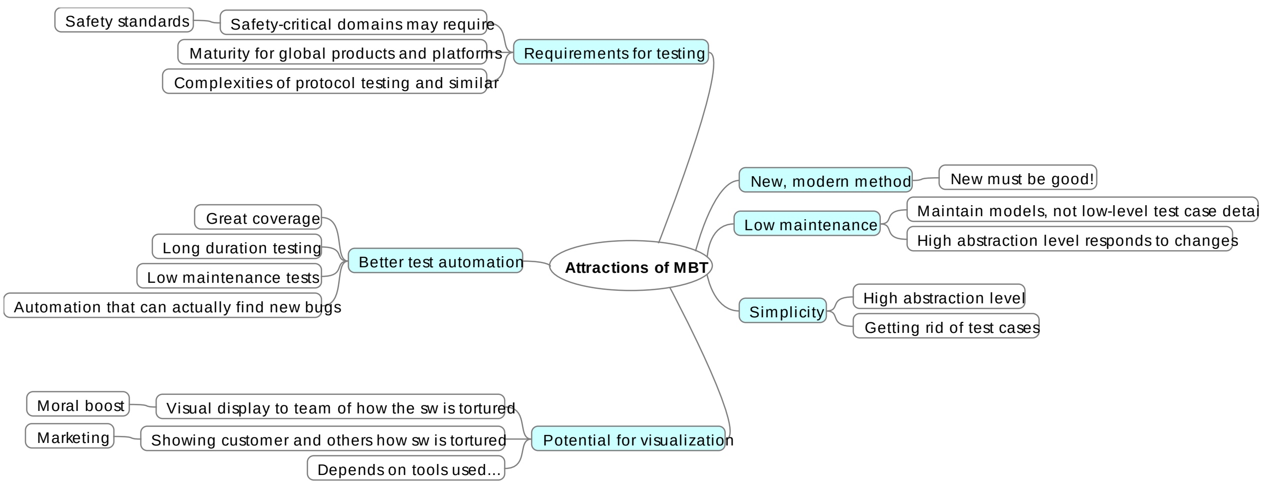
Attractions of MBT
Requirements for testing
Safety-critical domains may require
Safety standards
Maturity for global products and platforms
Complexities of protocol testing and similar
New, modern method
New must be good!
Low maintenance
Maintain models, not low-level test case detai
High abstraction level responds to changes
Simplicity
High abstraction level
Getting rid of test cases
Better test automation
Great coverage
Long duration testing
Low maintenance tests
Automation that can actually find new bugs
Potential for visualization
Visual display to team of how the sw is tortured
Moral boost
Showing customer and others how sw is tortured
Marketing
Depends on tools used...

Figure 1. Attractions of MBT.

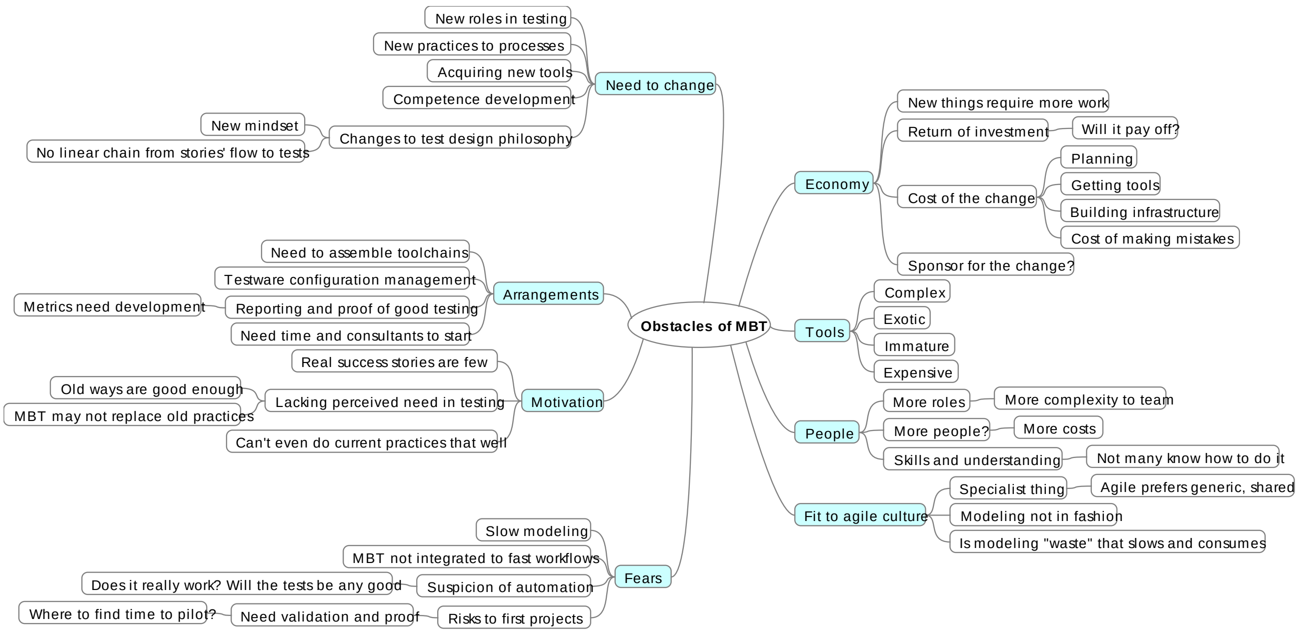
Obstacles of MBT
Need to change
New roles in testing
New practices to processes
Acquiring new tools
Competence development
Changes to test design philosophy
New mindset
No linear chain from stories' flow to tests
Arrangements
Need to assemble toolchains
Testware configuration management
Reporting and proof of good testing
Metrics need development
Need time and consultants to start
Motivation
Real success stories are few
Lacking perceived need in testing
Old ways are good enough
MBT may not replace old practices
Can't even do current practices that well
Fears
Slow modeling
MBT not integrated to fast workflows
Suspicion of automation
Does it really work? Will the tests be any good
Risks to first projects
Need validation and proof
Where to find time to pilot?
Economy
New things require more work
Return of investment
Will it pay off?
Cost of the change
Planning
Getting tools
Building infrastructure
Cost of making mistakes
Sponsor for the change?
Tools
Complex
Exotic
Immature
Expensive
People
More roles
More complexity to team
More people?
More costs
Skills and understanding
Not many know how to do it
Fit to agile culture
Specialist thing
Agile prefers generic, shared
Modeling not in fashion
Is modeling "waste" that slows and consumes

Figure 2. Obstacles of MBT.

Most of the attractions and obstacles are generic and apply to many development paradigms, but some relate closely to agile.

Some very interesting are the general approaches in agile:

- There is a tendency in agile to use generic methods that everyone can do. Just think about the lack of proper architecture and usability design in agile. MBT is not quite that, but with simple tools it can get closer.
- There is a negative relation towards modeling. After the demise of waterfall and RUP, people still have a “hangover” from modeling. It can be seen as “waste”, documentation that doesn’t provide value. Of course in MBT it is the very thing that provides value. But this is something that needs to be remembered.

- Still, agile aims to be fast and efficient and there are still too many stories of modeling having taken days. For that we need to emphasize incremental modeling that doesn't need to be waited for and good, fast tools.

4. Mapping MBT activities to agile development

4.1. Stereotypical agile process

There are many project models, but usually they are based on some cyclic process flow in which new versions of the system are created periodically. In this dissertation we want to use a specific process model such as Scrum as basis of this analysis, as specific models tie our analysis unnecessarily to their highly specialised details – and because in any case the specific process should preferably always be tailored to specific circumstances anyway.

Our simplified model (see Figure 3) has the following elements:

- Process flow.
- Startup activities. Pre-development tasks that are done before the increments – concepting work, etc.
- A series of increments, adding more features or otherwise more value. (Note: in some processes these recurring process phases are referred to as "iteration". We use the term increment here to avoid confusion. Inside the increments there will be plenty of iteration, as designs evolve through their analysis and we must not confuse that with the process phases.)
- A rhythmic series of releases, at the end of one or more increments. Not all increments need to produce releases and thus the need for results of the increments to be safe, or validated to be safe, varies.
- The releasing increments produce releases directly to the customer or production, or release candidates for additional internal processing (to be passed through required, for example, product management processes).
- Closure activities. Post-development tasks that are done after the increments.
- On-going activities, such as testing and safety tasks that are carried outside the increment model.
- Management and control processes that are outside the series of increments.
- Practices, related to, for example, integration and testing.

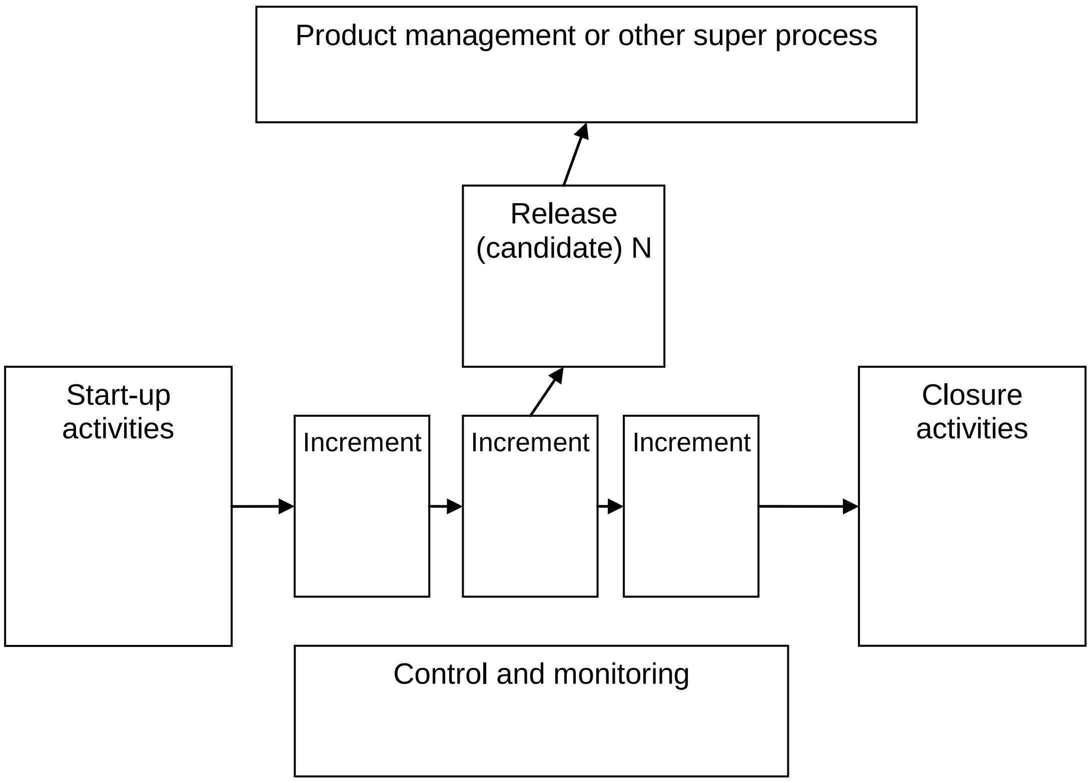

Figure 3. The basic agile process.

4.2. Overview of testing in agile development process

Overall flow of testing of a new feature / story:

- The implementation in planned and necessary testing planned .
- The developer creates the implementation and does unit testing.
- During integration (usually with continuous integration) the new implementation is put though to additional tests. This is mostly low-level integration testing, but can include running automated UI level tests.
- A tester does “system” testing often through UI. This is mostly functional testing.
- Automated tests are created mostly for regression testing of the feature later.

Testing of the whole:

- Manual exploratory testing.
- Automated tests during integration.
- Performance tests at various points of the project, sometimes automated for some builds.
- Usability assessments for the whole application, so the new UI’s are in context and in interaction with others.
- Maturity testing when aiming for releases.

4.3. Allocation of MBT work inside and outside team

There have been some work allocation models where modelling and even test execution has been provided as external service. If we aim for integration of MBT into the agile process and team, most of the work needs to be done inside the team, as it requires communication, collaboration and iteration.

There may be are some tasks related to the release process where tasks can be carried out outside the development team, such as testing related to high level system integration, long term maturity tests, safety validation tests and similar.

4.4. Main principles of agile test modelling

Ambler (2002, p. 27) gives these core principles for agile modelling of software in agile development.

- Software is your primary goal.
- Enabling the next effort is your secondary goal.
- Travel light.
- Assume simplicity.
- Embrace change.
- Incremental change.
- Model with a purpose.
- Multiple models.
- Quality work.
- Rapid feedback.
- Maximize stakeholder investment.

Here, we reflect on those from test modelling and derive from them some hypothetical principles for agile test modelling.

- Good testing is your primary goal. The goal is not to make models, but create good testing.
- Model for today.. The modeling should be targeted for the current implementation, not much as a platform for other. One must be able to make changes to any direction.
- Travel light. This means creating just as much of models that you can get by. Enough is the optimum.
- Go for simplicity. Just as for the application, simplicity is a virtue for the test models too. The simplest a model can be, the easier it is to understand, to maintain.
- Embrace change. The MBT tester must be positive towards change. The application will change and so will (should) the models as we learn more of the evolving system.
- Develop and change models incrementally. Don’t change everything at once. Start with simple models and keep the models working when you change them.
- Model with a purpose. Each test model should have a clear purpose in testing. Model for that. Generic “all-purpose modeling” has low value.
- Multiple models. Models look into the system at some perspective. The more perspectives we have, the better we can find defects.
- Quality work. Make the model professionally. Make models that are well crafted, robust and sufficiently documented and managed.
- Rapid feedback to developers. Testing in agile development needs to provide rapid feedback.

For that, modeling must be fast and tests need to get running fast. For that, start with simple models of most essential issues, test with those, and continue with more modeling and other executing strategies after that.
- Maximize stakeholder investment. This applies to all testing. With all work that you do, give maximum value to stakeholders. Provide by testing the most important information that the stakeholders need now.

4.5. MBT in process phases of agile development

4.5.1. Start-up activities

When the development process starts, some preparation is required:

- Selection of tools – if there are alternatives.
- Planning of the usage of MBT in the process. Just getting the shared idea for what is expected. Concise Master Test Plan is a good idea. Make everyone understand what is tested manually, what with traditional test automation tools and what with the MBT tools.
- Discuss testability. MBT (and other testing too) requires good testability. Make sure that everyone understands the implications for, for example, UI techniques.
- Plan the adaptation. Find and select an existing adapter or start building one. In the latter case, the adapter can be built incrementally as the project proceeds. Make sure there are resources for that.

4.5.2. MBT in testing of a new development item (feature, story)

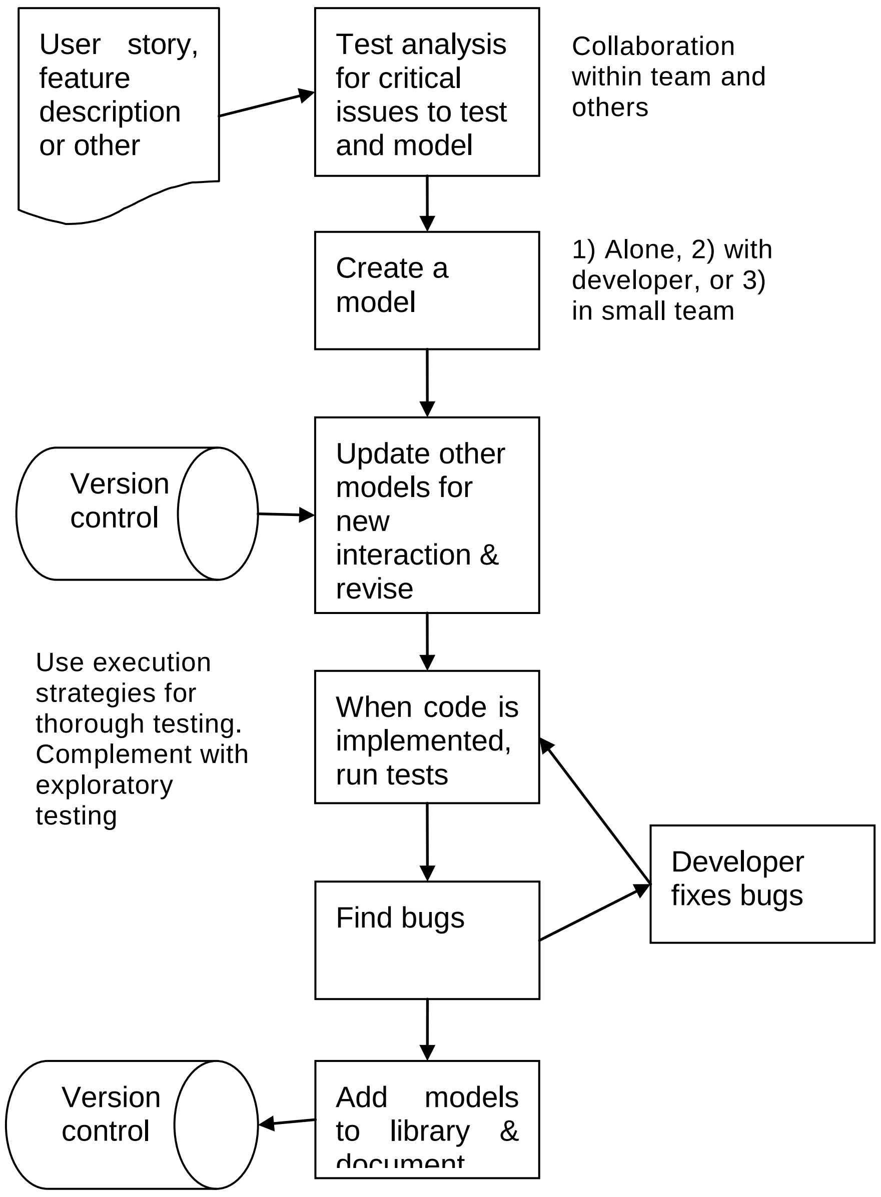
User story, feature description or other
Test analysis for critical issues to test and model
Collaboration within team and others
Create a model
1) Alone, 2) with developer, or 3) in small team
Version control
Update other models for new interaction & revise
Use execution strategies for thorough testing. Complement with exploratory testing
When code is implemented, run tests
Developer fixes bugs
Find bugs
Version control
Add models to library & document

Figure 4. MBT in testing of a new development item (feature, story).

4.5.3. Tests for the whole system

Basic low level integration testing

This is the traditional low-level integration of system components, such as classes and similar – and in particular, integration of code written by and changed by several developers. In that, the main tests are traditional unit tests written using a unit test framework. MBT has usually no role in that, except for fast (system level) smoke tests run at the same time.

Separate system tests that span various functionalities

This is a level of testing above the feature / story tests. Here we collect many test models to run a quite comprehensive suite of tests. This is usually done within the team, by a test engineer. Tests will bring out defects in the models and in the adaptation, so the person executing the tests should ideally be able to make the corrections or have someone in the team as a pair to do that.

Regression tests at least daily (depending on the volume of development)

This usually should be done inside the team. The models used should be created during the incremental development of test models, as well as any specifications for the test runs and the configuring of the models.

Performance testing

Performance tests should be run as early as possible. That means that the system will be quite immature at that point. Because of that, carefully written linear test cases actually make sense. Model-based tests can cause problems to the test runs! Still. carefully selected model-based tests might give more realistic performance information. That requires test models tailored for performance testing and carefully selected test execution strategy.

Longer term maturity tests

These may be carried out by a separate team or in the team during a stabilization sprint. MBT is very suitable for this kind of testing.

Things to note:

- The QA team should need to do minimal modeling or model management.
- Regression test suite can be a good candidate – the models are suitably simple.
- A suitable test execution strategy (coverage criteria) needs to be selected.

Customer acceptance tests

There are two types of those. In the agile culture simple acceptance tests are often used in which it is checked that the new functionality works the way the customer or product owned expects. Manual testing or traditional test automation really is sufficient for that, or rapid modelling can be used. Still, this is about communication and any means for that need to be as simple as possible.

But the problem here is that those really are not proper acceptance tests. There are no real users involved, just people representing them. Good acceptance tests need to be done by the customers in their own environment. Test automation and especially MBT should not have much role in that. They belong to the system testing level.

4.6. MBT as a tool to aid team’s understanding of the system

MBT is a great paradigm for visualising how the system “works”. Usually the developers do not use any modelling tools but use linear “coding” as means of implementation. Of course, the user stories and similar provide high-level descriptions of the desired behaviour and value to the user or customer, but there is still room for more viewpoints. And the views that MBT provides can be such.

To make the collective learning potential materialise, obviously collaboration is required. That means creating the models together with some developer (who is implementing the feature in question), or in a small team of people.

That way, people can:

- Discuss, what is important for the value of the system.
- Discuss, how the critical things are supposed to behave.
- Create simple models and reflect each other’s understanding of the behavior.

And after some testing, look together at the findings and learn from all that.

4.7. Definition of "done" and metrics

"Done" means that a new implementation is also tested. That is very clear to anyone and sufficient for team's use. One piece at a time becomes "done" and the coverage of testing integrates to the coverage of implementation. But there is more to "done" than that: Special issues for MBT include these:

- A story [or other] is modelled. The model is sufficiently documented and stored in version control to a model library or other testware storage.
- Any related models (other stories, features, applications) are updated.
- Adapter changes have been made.
- Tests have been run and passed [at chosen target quality level] and defects reported. There are no critical level or highest priority defects.
- Any small MBT tests to run automatically during builds are added to the continuous integration engine's scripts.
- Regression testing suites have been updated.

Still, other metrics are required for assessing the whole system. Mostly, in MBT, they are concerned with coverage:

- State and transition coverage.
- Use case coverage.
- Application component / functionality coverage.

Of course, the testing can also report code coverage. When the testing is done in the development team, it is easy to create builds that contain instrumentation for code coverage measurement. But this also means that the versions under test will not be versions targeted for production and thus the code coverage measurements need to be used with care. The possible side effects also depend on the system under test.

All these need to be combined from all test types – other test automation and manual testing (including exploratory testing). The goal must be monitoring how well the system has been tested, not how much any particular testing method has tested it.

4.8. MBT tool considerations

Because agile is a team effort, it is advisable to have testing technologies reasonably generic so that more than one person can participate in any effort without much training. That applies for example any languages used by the tools – rare, specialised testing languages (such as TTCN-3) should be avoided and preference should be given to the languages used in the implementation of the application, or generic, often used languages, such as Python.

4.9. Testware management

Version control system has a very important role.
MBT artefacts need to be stored in standard version control too:

- Often in a special directory “tests” etc.
- Scripts should be well packaged so that anyone can use them when needed.

All this means that testers need to embrace version control and shared repositories as one of their main tools.

4.10. Testware quality management and quality assurance

In traditional testing, there might be reviews for test models and adapter code, but in the agile culture it is hard to see that happening very much. Formal reviews are replaced with collaboration, team and pair work.

Still, reviews for models are an important thing and can be valuable and the more critical the system under test is, the more they should be recommended. One idea could be cross-reviews between testers of different projects. That would improve tester collaboration and aid in sharing testing know-how inside the organisation – which is a problem in any “developer team centric” way of working.

4.11. Making it visual and fun

Agile “likes” simple visual things: backlogs, dashboards etc. It is good to have something visual that gives an overview to what is happening. Logs are obviously boring to watch, but MBT could use a visual display that shows how the system is tortured by going through states with every possible combination of values and in every possible sequence. Seeing that would be good for bonding the team and showing it to the customer and other interest groups would be good for marketing the team’s competence and will of making really robust software. It would be interesting, beneficial, exciting and fun! But current tools don’t yet support that kind of monitoring.

4.12. Considerations for robot-assisted testing in agile

A robot has traditionally been something that belongs to a laboratory. But if we use them for testing in agile development that thinking must be revised.

- For daily testing in the team of features and applications the robot should be located with the team. Otherwise it will not get used. Of course, in that case it also must not disturb anyone – it must be silent and not cause visual attractions (motions, blinking lights…).
- For “QA” testing, including maturity testing, in can be located elsewhere.

Of course, when a team of developers have a robot in the room, it will be used for things other than testing. If this is not controlled, soon it may not be in working order. Perhaps in those cases two robots should be purchased – one for testing, one for play.

4.13. New competences for tester

This is an essential issue, but omitted from this document. This paragraph is just to remind of that!

5. References

Ambler, Scott. 2002. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley Computer Publishing. 384 p.