ATAC

ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

User-centered model-based testing tool design

– Design principles for testing tools that provide best possible tester experience

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

Table of Contents

1. Introduction 3

2. The main principles – a manifesto 3

3. A generic view to factors affecting the tester's performance 4

4. Support for tester's mental models 4

5. Support for "good work" 5

6. The main usability factors of MBT tasks and workflows 7

7. Usability heuristics 9

8. Testing tool UI as an interface to testability 11

9. Cultural and organizational factors 12

10. Attraction of MBT – how to make people love it 13

11. The competence issues 13

12. References 14

APPENDIX 1: Four MBT tools – all for different cultures 15

1. Introduction

This is a report that deals with a very critical issue: How can we develop model-based testing solutions that provide a nicest possible user experience and usability for the tester. It is clear that for a tester to be productive in today's rapid working environments and agile way of action, just technical excellence of the tools is not enough. The tools must be designed for the user – in this case the tester.

This report presents some principles and ideas for the testing tool developers which we hope will improve the consideration of these issues in future testing tools, in the ATAC consortium and elsewhere in Europe, hopefully in aiding the introduction of tools to a wider user community.

As an appendix we have also included a general test automation system quality model mind map and a test automation system checklist, because the user-centred qualities always exist in the context of overall tool quality. Those will be developed further separately.

2. The main principles – a manifesto

- Model-based testing (MBT) should be for everyone – not just experienced gurus or scientists.
- In the development of MBT systems we need to consider the skill and knowledge levels of all actors in the system – including management.
- Testing is always done in organizations. We must understand that all testing activities are in some form teamwork and we must support shared activities and the viewpoints of all interest groups and occupational groups.
- The true value of MBT systems results from their use. Therefore we must make them as easy to use and to take into use as possible.
- We need to practice solid usability and user experience principles in all elements of the test system and its processes.
- We must favour the practical applicability of the system more than its technical elegance.
- The culture of the organization greatly affects the suitability or tools. Some cultures prefer simple Unix-like small tools whereas some may find complex and big tools preferable.

There are many viewpoints to the user experience of test automation tools. Some of those are listed in ***Figure 1***.

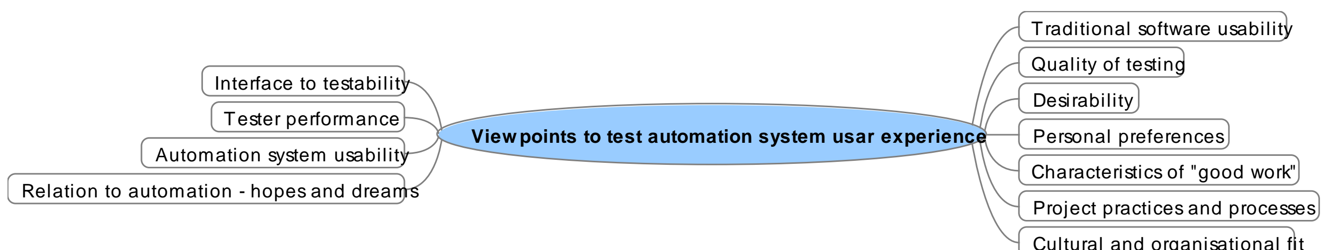

Figure 1. Viewpoints to test automation tool user experience.

3. A generic view to factors affecting the tester's performance

If we simplify things a little, there is just one purpose of a testing tool: to make a tester perform better – to be more efficient and effective, to find defects better, to provide better information. Therefore, we need to look into the tools in a performance context – what are all the things that affect a tester's performance? An overview to those is shown in ***Figure 2***.

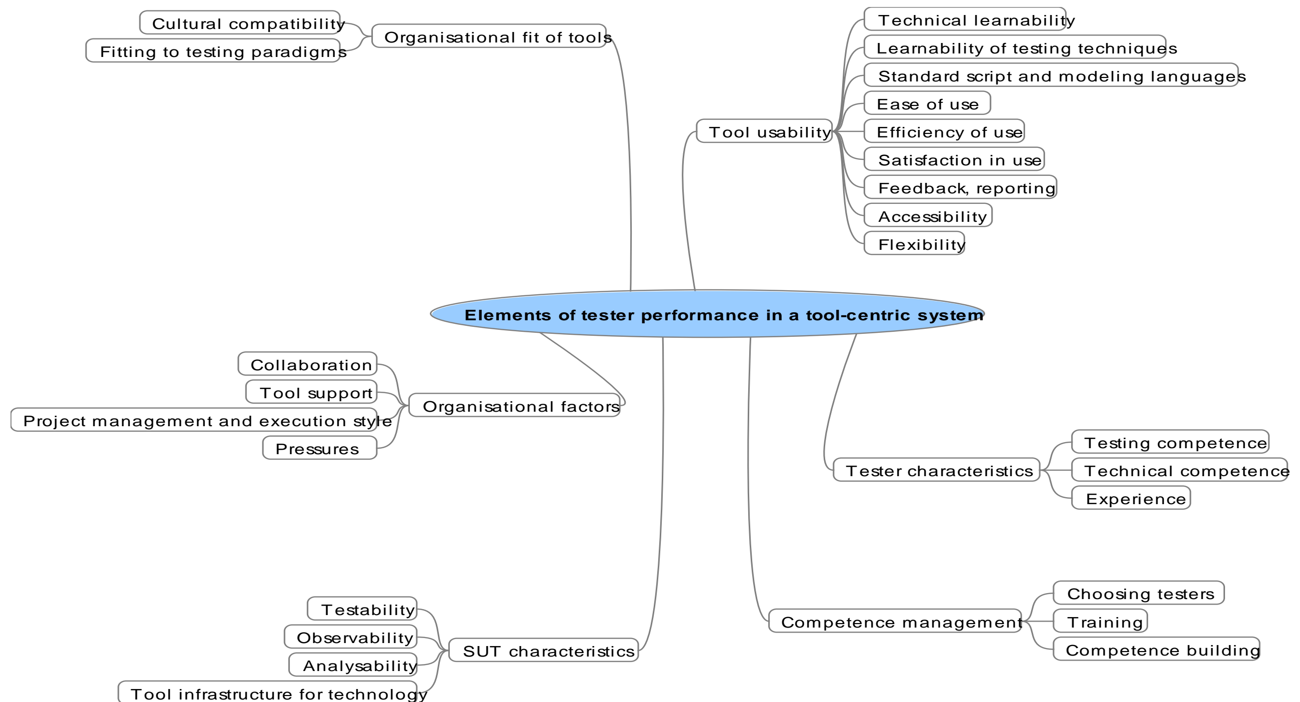

Figure 2. Elements of tester performance in a tool-centric system.

The figure is only illustrative in nature and lacks many important issues, but serves its purpose here: putting tools into context.

Of course, this is just one of the contexts that we need to look into. The other one is the traditional context of use, which is a narrower one and that is the reason why we look more into that one next.

4. Support for tester's mental models

Systems should support correct mental models – if the tester has "good" mental models, the system should support those, but in other cases, the system should "educate" the tester about how to approach things and do things and how things "are".

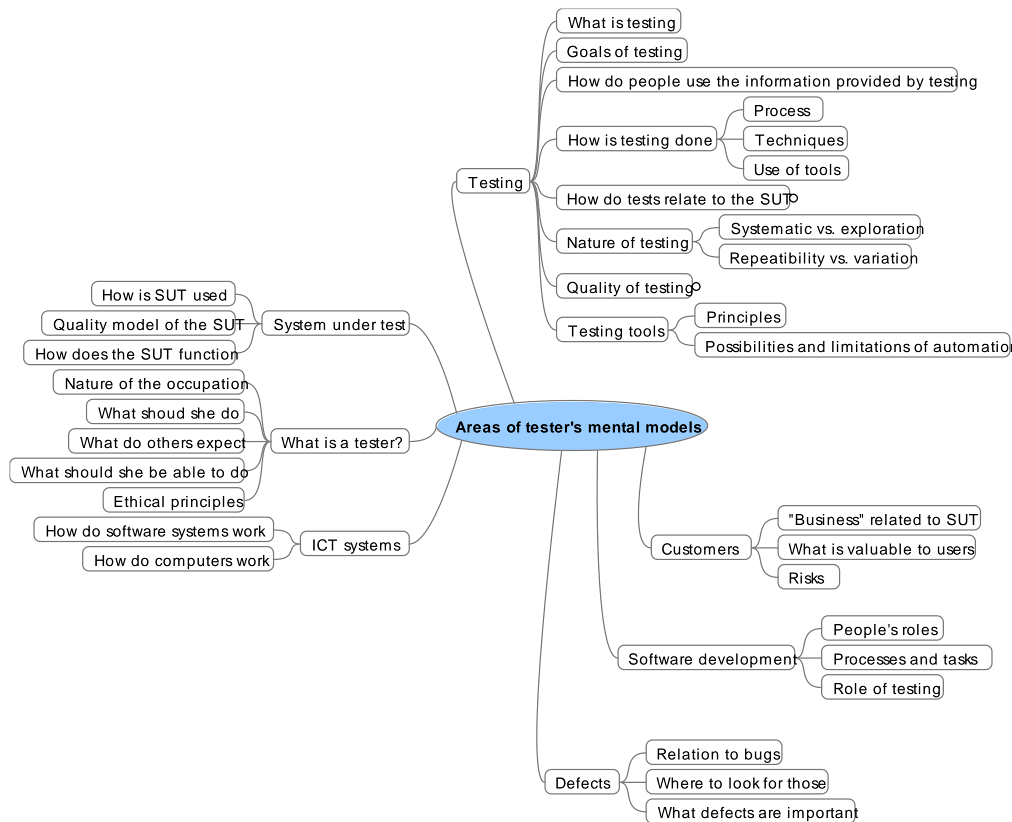

Figure 3. Areas of tester's mental models.

Notes on a couple of those:

- Some MBT systems have an inherent thinking pattern of execution state transitions that "are allowed" in some situation. That is a "positive execution" approach and leads to positive testing. But testers should try the things that are not allowed, in order to check that the system is robust. Testing tools and modelling techniques should emphasize that.
- This relates to the tester's understanding about bugs and quality of testing. Automation can lead to "making the moves", getting the test engines running, but that does not mean that testing is good.
- A tester has an internal model of how the SUT works. That model may be abstract – such as based on modes and states, or very physical – code lines and functions being executed. -- or many or those simultaneously. A good tool should provide an appropriate view to the SUT that reflects that.

5. Support for "good work"

Testing tools are just tools and as such they need also to be considered in the context of a person's work. Here we have two viewpoints into that:

- The Nordic understanding about desired characteristics of work.
- The overall activity system in the work context.

The characteristics of good work are something that have been tried to describe for decades, in order to make the work profiles or workers suitable for humans. This has been an essential approach in the manufacturing industry but also needs to be considered for ICT workers. The following characteristics are a personal condensation of factors presented in literature and are not supposed to present any complete picture of the issues.

- Suitable autonomy. The tester has possibilities to control the workflow and decide the order of activities.
- Good utilisation of human skills. Doing what humans are good at. Those include the ability to detect new patterns and unexpected things.
- Avoiding mechanistic routines. Let the technology, test automation do the routine tasks.

- Fruitful collaboration with colleagues. Parts of the collaboration are the workflows that produce things to test and expectations for testing and all the communication related to those.
- A clear role in the working community, team, and a sense of belonging.
- Sense of identity. The tasks and the tools build on those. A tester is understood from what she does and what tools she uses. Therefore, good testing tools should have a clear identity and be easy to explain, if they are not obvious.
- Security and safety regarding organisation and working environment. The tools must be safe, so the tester must not all the time be afraid of doing something that might do harm.
- Clarity of expectations. What is the tester expected to produce? How is she expected to work and act? Tools do not a large role here.
- Clear rules. This is a management / leadership and cultural issue. Tools have a smaller role in this, except that the rules regarding tool use emphasise organisational rules – there may be unwritten rules about who uses some tools, for example.
- Possibility to work individually when needed. Some tasks require deep thinking and some form of social insulation. That means that tools should not be tied to a location, but should be able to be accessed from anywhere. This emphasises network based tools (instead of workstation-specific tools).
- Possibility to participate in decision making in things that affect the tester. This includes choices in testing processes, methods, techniques and tools.
- The tester can decide the speed of working. An individual tester should never be a blocking tied-up part of a workflow. Viewpoints to that: test automation, multiple testers who share the workflow. Some buffering – not all builds are tested in similar ways.
- Good, reliable tools. Good test tools should be reliable – the testing tools themselves must be tested properly and be robust for technical issues and user errors.
- Getting feedback. The tools should give immediate feedback about how the testing tasks have succeeded and what the test results are, and the status of test runs.

One interesting aspect to this is how do people see their work in test automation? All automation implies automated systems that have the human as “operator” who just defines the task and rules for the automat. On the other hand, for good testing, the human should be more like a detective, a creative person who gathers information, places traps for the bugs and so on. Good test modelling is exactly like that. Still, the “operator” role can sometimes be appropriate – for starting traditional regression tests during building and similar, but for other tasks, it can be harmful.

When designing the test systems we need to understand the roles and give support for the role that should be dominant at each testing task. This way, we can make the tools more effective aids for the tester and also make them trigger any expected transition between the tester mind-sets (yes, the tester can have states and transitions too!).

6. The main usability factors of MBT tasks and workflows

6.1. General

This chapter presents the main use cases in the lifecycle of an MBT system and the essential usability factors associated with each.

Generally, the usability of an MBT system does not consist of just generic UI issues (although they are important), but of the quality of the system in many regards, such as:

- Suitability for work processes and to be used with any other tool or technical system,
- Suitability for the actual tasks and nature of the project and its characteristics and pressures.
- Suitability for the actual user. Users notoriously vary and in some context there may be similarities within a group of MBT tool users that may cause some tool characteristics to suit or not suit that group better or worse than some other group.
- Support for the internal mental models and “natural” working styles of the tester. Attempt to minimise the tool and to let the tester concentrate on the task.
- Suitability for use in a team and in an organisation.
- Ability of the tool’s to deliver information about the testing.
- Cultural fit.

6.2. Choosing test automation tools

Choosing the best tool to any automation task is the critical phase. If the tool chosen is not appropriate to the task, its features are not of use and its usability will not help the testers do their work.

Usability factors:

- The tool’s purpose is clear: what is it designed to test, in what environment and conditions.
- The tool’s feature set is clearly documented and one can assess based on the descriptions what the tool can do and how it will relate to and interact with other tools.
- Demonstrability. One needs to be able to try the tool and to demonstrate it to others.

6.3. Choosing the test strategy and automation strategy for a project

In a development project, one must select a good strategy for the whole testing. For automation, one must decide what will be automated and what will not be automated.

Usability factors: None.

6.4. Tool configuration and preparation

In today’s projects time is of essence. Projects need to get started rapidly and testing needs to be started as soon as possible. There will not be much resources for configuring and preparation of tools.

Usability factors:

- How simple the configuration and preparation is.
- Number of installation tasks and tools used in installation and configuration.
- Understandability of configuration options.
- Architecture of configuration system and its files.
- Portability, including possibility to install the tools without admin rights on the computer.
- Ability to reuse existing configurations.

- Ability to automate the configuring and to do it remotely.
- Support for virtualisation, including using ready-made virtual machines.

6.5. Test modelling

Test modelling is traditionally a very challenging task, requiring special skills.
Usability factors:

- The type of modelling language – visual or code-based.
- Complexity of modelling language – number of abstraction levels needed, complexity of data structures.
- Readability. User-friendly design of keywords and terms, clarity of diagrams.
- Ease of use of modelling tool. Traditional software / UI usability issues. Compatibility with conventions in other tools used. Integration with IDEs and similar.
- Handling of large models. Ease of splitting and combining models.
- Ease of reusing models for new products, product variants and projects.
- Version and configuration management of models.

6.6. Test execution and test debugging

Testing is the phase that happens over and over again. Obviously, it must be efficient and easy. But for any new modelling there will be teething issues – there are errors in the models and problems in the test tool configuration. Solving these will cause difficult problems if the tool doesn't provide good support.
Usability factors:

- Efficiency of test execution – number of steps, need for manual interaction.
- Observability of test execution.
- Ability to locate test execution problems and errors.
- Ability to debug and simulate models with stepwise execution and observation of model states.
- Linking of test results with run history events.
- Readability and understandability of test logs.

6.7. Test reporting

Reporting is where we get output from the tool for others – information about errors and defects, information about the success of test runs and about the general state of the system under test. So, information ergonomics is essential here and making the tool generate such information that various parties can use in their tasks and their decision making.
Usability factors:

- Understandability of test results and reports.
- Clarity of meaning of test metrics.
- Tailorability of test results.
- Ability to integrate tool with other reporting systems.

7. Usability heuristics

7.1. General

Heuristics are “rule of the thumb design rules” for the usability. Here we look into a couple of heuristics set (the ones included in the Heuristic evaluation Wikipedia page) to see what specific guidance they might give to the design of testing tools.

7.2. The “original” heuristics by Nielsen

Let’s first look into the “original” heuristics by Nielsen (1994):

“Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.”

- Reflection regarding MBT: For MBT tools this means that the user should be able to see how testing proceeds, what elements have been covered and what errors have been found. This implies using both logs and visualizations.

“Match between system and the real world: The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.”

- Reflection regarding MBT: There are various aspects to this: 1) Speaking “testing language” – any modelling specific or mathematical concepts should be translated to terms used in testing. For white box testing at unit level, the testing conventions should be targeted for programmers. 2) Speaking the context language, using domain concepts. Context specific languages and high abstraction level keyword based testing excel in this.

“User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.”

- Reflection regarding MBT: Ability to cancel and interrupt test runs. Ability to restore the test environment state.

“Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.”

- Reflection regarding MBT: Testing tools should follow the conventions of other tools used in the platform of in the domain. Sometimes testing tools can be like prototypes with their own UI style, perhaps due to lack of proper UI design. Modelling techniques should be as standardised as possible.

“Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.”

- Reflection regarding MBT: Static analysis and simulation of models before execution, especially if real executions requires a special environment.

“Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.”

- Reflection regarding MBT: Visual modelling with contextual menus is excellent in this regard.

“Flexibility and efficiency of use: Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.”

- Reflection regarding MBT: Often used tasks in testing and test preparation should be made as efficient as possible. Macros and small utilities are valuable.

“Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.”

- Reflection regarding MBT: Testing tools should be made simple. This is usually the trend nowadays (2013).

“Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.”

- Reflection regarding MBT: Besides standard emphasise to error messages in the tools, any messages and related information from the SUT must be presented as clearly as possible.

“Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.”

- Reflection regarding MBT: Just like the user interface, testing tools are sometimes lacking in this regard.

7.3. Gerhardt-Powals’ cognitive engineering principles

These heuristics by Gerhardt-Powals (1996) take a more holistic approach to evaluation and aim to provide cognitive principles.

“Automate unwanted workload: Free cognitive resources for high-level tasks. Eliminate mental calculations, estimations, comparisons, and unnecessary thinking.”

- Reflection regarding MBT: Automate the easy things.

“Reduce uncertainty: Display data in a manner that is clear and obvious.“

- Reflection regarding MBT: Pay emphasis on reports, logs and metrics.

“Fuse data: Reduce cognitive load by bringing together lower level data into a higher-level summation.”

- Reflection regarding MBT: MBT needs good metrics. There is a still lot of work in this area.

“Present new information with meaningful aids to interpretation: Use a familiar framework, making it easier to absorb. Use everyday terms, metaphors, etc.”

- Reflection regarding MBT: Attaching metrics and observations to common things like displays or requirements provides a common ground for others to understand the issues. “States” can be too abstract. Metaphors, however can be dangerous.

“Use names that are conceptually related to function: Context-dependent. Attempt to improve recall and recognition. Group data in consistently meaningful ways to decrease search time.”

- Reflection regarding MBT: The first requirements were assessed with Nielsen’s heuristics. Grouping data means for example splitting models (grouping state information) and presenting logs grouping by some attribute.

“Limit data-driven tasks: Reduce the time spent assimilating raw data. Make appropriate use of colour and graphics.”

- Reflection regarding MBT: For MBT, logs are the biggest type of raw data. Presentation and analysis of those is very important.

“Include in the displays only that information needed by the user at a given time.”

- Reflection regarding MBT: One particular issue is filtering the models and selecting what to view at any time. That can be very important for any non-trivial model.

“Provide multiple coding of data when appropriate.”

- Reflection regarding MBT: Using code and visualisation is one implementation of this.

“Practice judicious redundancy.”

- Reflection regarding MBT: Providing multiple views to elements in the test system in one thing to look into. Software architecture and behaviour always benefit from multiple, partly redundant views to the system at various abstraction levels.

8. Testing tool UI as an interface to testability

One view to look into any testing tool is to see it as an interface to the testability of the system under test. Traditionally, testability characteristics include things like controllability, observability, explorability, measurability and others. A good user interface should be a “transparent” aid for these, but at the same time do any transformations to the information that aids the tester or the testing process. In Table 1 some testability features and possible testing tool UI aids to support them are presented. Note that the testability features are somewhat interdependent – for example explorability is strongly related to controllability and observability.

Table 1. Some testability features and possible testing tool UI aids to support them.

Testability features (some)	Possible UI aids
Controllability	
• Managing of states	Visualisation of states. Stepping thorough states. Returning to previous states.
• Execution scope management	Control means for activating subsystems and components. Dynamic model contraction. Control of interfaces (e.g. firewalls).
• Management of the execution sequence	Selectable starting and end points for test execution. Breakpoints.
Observability	
• Observing internal states	Viewing state information.
• Observing external states	Viewing state information of external systems, including distributed subsystems, data stores, devices. Alarms for state variable changes (system change, data change).
• Making things visible	Slowing the process so that fast activity can be seen. Stepping through sequences.

Testability features (some)	Possible UI aids
Analysability	
• Seeing relations between actions / activities and the resulting changes	Visualisation or relations between system elements. Tracking of actions to their potential effects.
• Structural visibility	Ability to show and follow structures and their relations (e.g. opening an interface to show its code and further, the called external code and its associated data).
• Error tracing (tracing the beginning of certain chain of events)	Display of execution chain. Backtracking the system states / calls to one that started event chain that resulted in the examined condition.
Explorability	
• View the status of exploration	Show what has been done / happened in system elements / states. Show states / elements not accessed. Dynamic change of variable values.
• Access to everything	Provide dynamic testing interfaces that the tester can use to “do anything”. Full access to the object model (or similar) of the system.
• Exploration is safe	Indicators of dangerous states / corruption etc. (depending on testing environment).
• Fast ad-hoc changes to test actions and approaches	Non-modal UI, having all actions available all the time.
Measurability	
• Activity measurement. Measuring of actions (in broad sense) by type	Ease of (or automatic) instrumentation. Views to logs by filtering (less pre-selection).
• Time measurement	— “ —
Configurability	
• Ease of configuration of SUT and test environment	Configuration tools. Dynamic configuration. Saving and restoring of the system state.
• Ease of configuration test environment	Configuration tools. Dynamic configuration. Saving and restoring of the system state.

9. Cultural and organizational factors

Tools are always used in an organisation where they must fit the technical and social environment. One part of the culture is what kind of tools the development and testing organisations are used to – some teams collectively prefer small Unix/Linux -type tools whereas others like to see a more holistic system with more integrated functionality.

Usability factors:

- “Style” of tool matches the collective general preferences (size, amount of functionality, user interfaces) of the organisation.

10. Attraction of MBT – how to make people love it

Related to the cultural factors is the question of how to make people love the tool. When we are introducing a new tool to an organisation everyone should have urgency to start using it. People should really desire the tool. These are no more just usability factors although it is nowadays understood that people work better with tools that they like – when using such tools they are more relaxed, more creative and innovative.

Usability factors:

- Maximum cultural fit to organisation.
- Clear fit to some parts of existing test / development processes or their phases / tasks.
- Maximum support for key users' tasks.
- Clarity of proposed results – everyone can see and like what the results will be.
- Good appearance and visual design that matches that of the environment.

In addition to these "rational usability" factors there are other important user experience factors:

- The tester must be able to feel proud of the tool. It is an extension to her identity. Therefore, it must present the values of the tester and make those values visible. An example of those is general professionalism, or some testers, creativity.
- While testing is serious professional work, sometimes tools and methods should be somewhat like games. In explorative work that is clearly possible (exploring a SUT could be compared with exploring a game space). There could be possibilities to this, that we should keep an open mind towards.

11. The competence issues

Another big part of the performance issue is of course the testers' competence. The tools and the tasks form a unity where the tester uses his/her competence to do what is required. Tools are just tools. In an advanced system such as an MBT activity system there are various competence-critical factors where we need to assess the possibilities to improve the process and the tools and also the possibilities to improve the competencies of the testers.

But first we need to look into the elements of competence in this kind of setting. The approach here tries to be practical and will be reflected to more detailed competence models in other works.

Elements of competence include:

- Testing skills.
- Context knowledge.
- SUT related skills.
- Project competencies.
- Communication skills.

Let's take a brief look into each on some of those – in the context of MBT.

Testing skills is obviously the main factor here. If we see testing as intellectual work that tools only support, utilising and supporting that is the beef of the whole MBT issue. The testers need to be able to design and execute good tests. For that they need to have traditional testing know-how, but also other competences come into play. Key issue in model-based testing is modelling. The tester must understand what to model and how. To be able to extract the essence-to-be-modelled of the system under test, the tester requires context knowledge and also technical understanding of the SUT, its behaviour and architecture. That is not a trivial issue and modelling tools can only help in that to a level. Generally, visual tools are beneficial as they show the behaviour more than in just code fragments. Still, it depends on many factors whether the model-creation should be done using visual tools or whether it is sufficient to have to the tool draw the visualisation (as is the case in Intel's fMBT tool).

For the basic model creation automatic tools have been suggested that would either create a model fully automatically by exploring the SUT or by recording only selected user interactions. This applies to UI-level modelling only. Those technologies would clearly give benefits as there is a big difference between being able to create a model from scratch and being able to edit an existing model.

12. References

Nielsen, Jakob (1994). Usability Engineering. San Diego: Academic Press. pp. 115–148. ISBN 0-12-518406-9.

Gerhardt-Powals, Jill (1996). "Cognitive engineering principles for enhancing human - computer performance". International Journal of Human-Computer Interaction 8 (2): 189–211.

Heuristic_evaluation. Wikipedia. http://en.wikipedia.org/wiki/Heuristic_evaluation (checked 2013-01-29)

APPENDIX 1: Four MBT tools – all for different cultures

This appendix briefly analyses the approaches of four MBT tools. It is based on presentations given in the INFORTE seminar Model Based Testing (http://inforte.jyu.fi/events/MBT). The basic idea of this text is that all the tools mentioned are fine examples of MBT, but vary on their suitability for testers and the organisational culture where the testers work. It is something that is often neglected, but should be considered in the design of tools. **Note that this text is not intended to be a review of the tools (and may contain inaccuracies). It just aims to present their differences in approach and thus their differences in suitability to different organisations and cultures.**

TEMA (http://tema.cs.tut.fi/) from Tampere University of Technology, presented by Antti Jääskeläinen and Mika Katara, is a tool that was developed especially for the testing of user interfaces of mobile devices, and remains quite unique in that. It utilises visual modelling and a layered architecture that separates the behaviour of the system and its implementation, making the tool suitable for testing product lines an helped by it in being SUT platform and UI technology independent (though the toolset itself requires Linux to run). The approach reflects the goals of the mobile industry of going into a more abstract specification style in development. TEMA is implemented in a quite complex architecture, representing a total system with planned roles for all professionals participating in its use. Even its use as a service – internal or externalised – has been considered, as separate testing teams are often used in the companies that are likely to use TEMA. TEMA supports online MBT only and has recently been included in robot-based test systems by OptoFidelity.

In a stark contrast to TEMA, fMBT from Intel (https://01.org/fmbt/), is a tool with no visual editor, though it can visualise its models. This tool is particularly interesting, as its main developer Antti Kervinen was also a major force behind TEMA. fMBT was developed to meet some practical goals, most importantly to be an easy tool for any developer – many of who live in an "Unix-like" tool culture, where command line has a preference over visual tools. Also, fMBT needed to work on API level and support other kinds of development tasks than TEMA. Where TEMA is mostly concerned with testing interactions between concurrently executing mobile applications, fMBT had a domain in the testing of device configurations, various system components and subsystems and similar. So in its development was integration to the developers' culture and infrastructure crucial and that shows clearly in its command-line tools, configuration files and model definition syntax and support for many programming languages, making it a tool that can be rapidly integrated in most any technical development project. Kervinen acknowledges all this – different purposes need different tools. It is also understood that personal and team's preferences highly influence the design and usage patterns of the tool. fMBT supports many languages (from Linux shell scripts to JavaScript), which makes it a versatile generic testing tool for many development teams. It should be noted that while fMBT does not use graphical tools for creating models, it can create such, thus allowing for graphs to be used in communication. As fMBT is still a new tool and has recently been released as open source, it will be interesting to see the paths of its future development.

OSMO Tester from VTT (http://code.google.com/p/osmo/) is another code-based tool that aims for simplicity. It is based on the premise that modelling should be done using a general programming language, preferably the same language that the system is implemented with. That way the tool can be integrated into any existing of future testing and development tool infrastructure (just like for example JUnit, the most well-known unit testing tool for Java developers) and the tool can utilise existing assets, testing libraries and most importantly – existing skills. That approach at the same time may somewhat focus the tool's users to developers and "developers in test "-- a role common in many companies such as Google and Microsoft and has been found in many telecom sector companies. One thing that is lost is the expressiveness on well-drawn visual models. There have traditionally been big problems with incomprehensible visual test models, but the work done by researchers have provided some tools for helping in that, so the problems with diagrams sometimes reflect badly done models and not the state of the art (or craft). Of course we do not live in an ideal world. Still, the OSMO approach provides a very effective test design tool for Java-skilled test designers. OSMO supports both online and offline MBT and is mostly restricted to testing a single application only. Support for any system events (like an incoming call or similar) is by test environment only.

MATERA from Åbo Academy (http://research.it.abo.fi/research/embedded-systems-laboratory-old/projects/previous-projects/d-mint/matera) again shows a unique approach. The idea was to create a framework which is independent of the tool chain used in test generation and execution and to base the modelling to UML (SysML), making it as generic as possible. The approach should make it possible to utilize many different tools, such as Conformic QTronic, which was used during the development in a collaborative

research project. A case study in the telecom sector may have greatly influenced MATERA's design, as it emphasizes requirement based testing and tracing between requirements and tests, which is an essential element in the development of telecom systems. In that regards, MATERA shows clear benefits to companies and product types that have that as a requirement. With other tools, this support needs to be built otherwise, if necessary. Due to the design, MATERA can support online and offline testing in any platform that the test generation and execution components support. That also means that many elements of it are unknown – such as the algorithms used in proprietary third party tools.

Overall, it is good to see various approaches to MBT tool design and also to see that the choices have roots in each target context and are not made just by preference. Still, some tools might have a more versatile feature set, if the developments were not hindered by the (natural) lack of development resources in research institutes and in the in-house tool development. That is often a problem in the development of internal tools and tools developed in research projects.