December, 2009

testing experience

The Magazine for Professional Testers

Standards – What about it?

printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

ISTQB® Certified Tester
Advanced Level Test Manager

ACCREDITED ISTQB TRAINING PROVIDER

ISTQB International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Dear readers,

Standards are there to be used, but as Erik van Veenendaal said, with common sense. The drug industry, for example, is forced (damned) to follow them, other parts of industry are less forced and don't do it.

As you can follow in this issue, there are many standards and also as many opinions about them. Go in and get your own opinion.

Graham Bath told me about the idea of having a "readers' opinions corner" in the magazine. I think that this is great idea and that's why we decided to have it. We want to encourage you to send us your opinion about the articles.

Let us know, if you agree with them or if you are very disappointed. A far as you don't insult anyone, we will publish all your comments.

As you know, we recently organized the Agile Testing Days in Berlin. It was great. We had very good talks, good mood, an excellent party and, last but not least, a great audience. They twitted all the time, and you could follow it under the hash tag #agiletd. A great experience. We plan the next event for October 4-7, 2010 and hope to see you there. Check the call for papers please.

The next Testing & Finance will take place in June in Frankfurt again. Save the date!

Based on the success of the conferences, we plan testing conferences around the world. Check please our website to see which of our testing conferences is near to you.

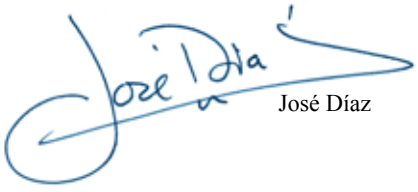
The testing experience magazine now has two sisters: Security Acts, the magazine for IT-Security, and Agile Record, the magazine for Agile Development and Testing. Don't hesitate please to contact me, if you are interested in writing an article.

We would like to also have great success with these magazines and ask you for your support.

We decided to stay in Berlin over the Xmas-holidays. Although it would be quite warm in Gran Canaria and the sea water is fantastic, and although I will miss my family, we thought that a little bit of snow as a contrast to sunny 28°C at the beach is good for the kids. White Xmas is fine. They just know Xmas at the beach with 28°C! Life is hard.

I wish you the best for the time and hope you will jump happily, healthy and loved into the New Year 2010.

Felices Fiestas!



José Díaz

Contents

IEEE829:2008 A major change of focus

by Bernard Homès

The test plan is the basis for all future test activities on a software application. It is a contract between the test and development teams and the management.

IEEE 829:1998 was focused on documentation and this documentation detailed the strategy to be used for the testing activities for:

- Management to understand the added value of the tests and the remaining risks, the costs and requirements as well as the time frame,
- Development team to focus on the areas that are most critical or where a required level of quality has been set,
- Test team to develop and execute the test campaign on time and within budget,
- Test project manager to have a reference document to keep the project on course and on budget.

In 2008, the IEEE Standards Organization published a revised version of IEEE829, expanding it from software to also cover systems.

This article describes the different types of document used in the IEEE829:2008 standard and their relation to one another. Building on this information, it details the different steps and aspects of the test plan as described in the standard and the relations between the different chapters, how they complement each other in order to provide a complete solution

IEEE829:2008 introduces a number of changes, one of which is the change of focus from a document-centric standard to a process-centric standard.

This article maps changes from the old version with the new version, and suggests ways to implement the new version of the standard in your context.

Directly from one of the members of the IEEE829:2008 team, learn how to change your focus from documentation to testing tasks.

Note: This article was originally presented at CONQUEST09, but has since been improved

Expose

1 Key points

The following key points will be highlighted in this article:

1. Synthesis of the IEEE829:1998 and IEEE829:2008 standards
2. Relationship between the documents
3. How to be compliant to the 2008 version and how to migrate from 1998 to 2008 version
4. Advantages and drawbacks of the IEEE829:2008 version

1.1 Intended audience

The intended audience includes:

- Test and development managers as well as project managers and any testers, analysts or consultants intending to deploy the IEEE 829 standard.
- Other suggested audiences include business, line and purchase managers, in order to understand what is required in the context of testing their systems, so that they are able to evaluate proposed testing strategies.

2 Novelty

IEEE 829 is a well-known standard for software test documentation, that is not specifically new. The standard has been widely commented, sometimes attacked, by the different schools of testing, and is a bone of contention between the Agile testing school – which consider documentation to have very limited added value – and the systematic school, where the strict adherence to established processes and standards is considered mandatory.

Since 2003, a group of international experts from the different schools of testing have gathered under the auspices of the IEEE, in order to improve the standard and make it ready to face

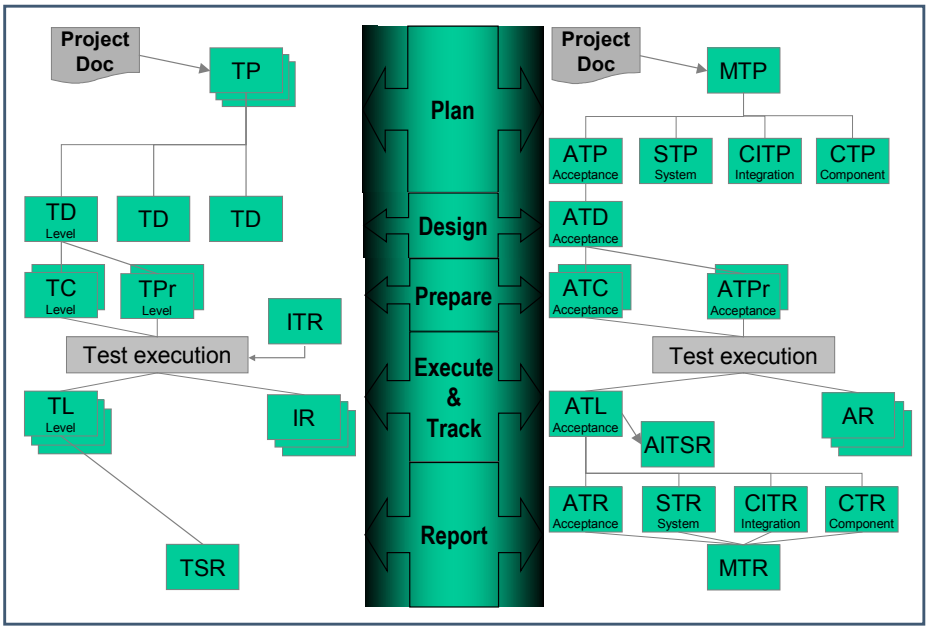


Figure 1: IEEE829:1998 vs IEEE829:2008 structure

the new challenges, adapting it to the evolved and more complex world facing us.

This endeavour reached its climax in July 2008 with the official publication of the IEEE829:2008 version of the standard, which now addresses both software and system test documentation. This expansion of scope from "software" to "systems" make this standard a leading item when one takes into account the increased complexity of systems and the emergence of systems-of-systems.

This paper describes the highlights of the new standard, its possible implementation in your or-ganization, and the tasks needed for you to become compliant with the new version of the standard.

2.1 Organization

The organization of the documentation suite of the new version of the standard, compared to the 1998 version is as described in the diagram (Figure 1). Shown on the left side is the IEEE829:1998 structure of documents, and on the right side the new 2008 version.

(See Figure 1)

Legend for IEEE829:1998 is as follows: TP: Test Plan; TD: Test Design; TC: Test Case; TPr: Test Procedure; ITR: Item Transmittal Report; TL: Test Log; TSR: Test Summary Report.

Legend for IEEE829:2008 is as follows: ATP: Acceptance Test Plan; STP: System Test Plan; CITP: Component Integration Test Plan; CTP: Component Test Plan; ATD: Acceptance Test Design; ATC: Acceptance Test Case; ATPr: Acceptance Test Procedure; ATL: Acceptance Test Log; AITR: Acceptance Interim Test Status Report; AR: Anomaly Report; ATR: Acceptance Test Report; STR: System Test Report; CITR: Component Integration Test Report; CTR: Component Test Report; MTR: Master Test Report.

3 Benefits

Compliance with a standard means that the person / organization intends to ensure that its processes and documentation fit a specific framework. This usually involves complying with mandatory (or regulatory) requirements and is based on the principle that an organization can improve its deliverables if the development processes are repeatable and measurable. In IEEE829:1998, the only items that were mandatory were the title of the documents and their chapters. This was sometimes considered as too burdensome and at other times too light-weight to fit the type of development undertaken.

With the advent of different SDLCs[1] and the increased complexity of some systems, it became clear that a new, updated and upgraded standard was necessary. This standard would be aligned to the ISO/IEC 12207 standard and adaptable to the criticality of the software and systems being developed.

1 SDLC : Software Design Life Cycle, described in ISO/IEC 12207

The inclusion of persons like James Bach and Cem Kaner from the Agile / context-driven school of testing ensured that the standard would focus on tasks that added value to the system and that it is adaptable to different contexts.

3.1 Applicability

Software becomes more and more pervasive nowadays. It is no longer limited to applications running on mainframes, minis or micros, linked or not by networks. It is now present in almost any commercial appliance, from fridges to cars and from telephones to personal assistants. The hardware verification aspect is now as important as the software-specific aspect. The new version of the IEEE829:2008 standard addresses this.

The new applicability of the standard is reflected in the decision to use the terms "software" and "system" interchangeably to reflect an "identified component or collection of components that include but are not restricted to software […] that comprise a software-based system".

This generalization to include hardware components implies that the future test managers and test analysts in charge of defining test strategies, test plans (Master Test Plans and Level Test Plans), will be required to have an understanding of the environment the software will be running on.

In terms of contexts, this standard can be applied in multiple contexts and environments, from aerospace to automotive industry, but also to other industries such as banking, insurance, and de-velopment of any software.

3.2 Tasks vs. documents

In IEEE829:1998, only a limited set of documents was suggested (see figure 1, left side), and testing tasks were defined based on the deliverables. In the new 2008 version of the IEEE829 standard, a number of tasks are provided per process, and each task has associated inputs and outputs, and can be associated with specific levels defined in ISO/IEC12207:2008.

This enables authors of test documentation to define the tasks required, and to use the standard as a checklist from which to pick and choose the different tasks according to their own context, ensuring that all tasks selected provide added value.

It is important to note the linkage between IEEE829:2008 and ISO/IEC12207. This allows mapping of testing tasks to other tasks not directly linked to systems development, such as acquisition.

As can be understood, this shift of focus from a document-oriented standard to an activity-focused standard enables the users to map their processes to the standard and check for any deficiencies. It is thus easier for users to implement this new version of the standard than the previous version.

Another aspect that should not be forgotten is the possibility to use this standard even when using agile methods and exploratory testing techniques that de-emphasize the use of documentation. As these methods and techniques are task-focused instead of being document-focused, compliance to the standard can be attained.

3.3 Integrity levels

Failure of some software and/or systems may lead to serious or even catastrophic consequences that cannot be solved by just rebooting the system.

Some examples include the Osprey V22 systems and the multiple developmental problems it suffered, killing in the process a number of test pilots. Other similar issues abound, so it was deemed important that the IEEE829:2008 standard addressed this in detail, showing what tasks were recommended at each level. These levels were called "integrity levels" and implement in IEEE829:2008 the ideas that were defined in DO178B/ED12B, ECSS and other standards.

The standard proposes a four-level integrity scheme, numbered 1 to 4 (4 is the highest integrity) described as follows:

- Negligible : "Software must execute correctly or intended function will not be realized causing negligible consequences. Mitigation not required."

- Marginal : "Software must execute correctly or an intended function will not be realized causing minor consequences. Complete mitigation possible."

- Critical : "Software must execute correctly or the intended use (mission) of system/software will not be realized causing serious consequences (permanent injury, major system degradation, environmental damage, economic or social impact). Partial-to-complete mitigation is possible."

- Catastrophic : "Software must execute correctly or grave consequences (loss of life, loss of system, environmental damage, economic or social loss) will occur. No mitigation is possible."

This concept of integrity level trickles down to other aspects of testing, such as the level of mandatory documentation according to the integrity level, the recommended minimum tasks per level and their intensity, and level of tester independence (see section 3.3.3). Integrity levels can also be defined based on the results of a risks analysis.

Even though the IEEE829:2008 standard does not mandate the use of integrity levels, it is noted that the use of an integrity level is "a recommended best practice that facilitates the selection of the most appropriate activities and tasks".

Incidentally, the implementation of integrity levels is also an incentive for defensive programming and other similar reliability enhancement techniques for software development.

3.3.1 *Mandatory documentation per integrity level*

The IEEE829:2008 standard proposes (table 2, page 16) a list of test documents associated to the integrity level of the systems or software being tested. An extract of this table (for integrity levels 4 and 1) is shown right:

A quick comparison of the two columns shows that for a software (or system) with "negligible" integrity level, the types of documents, number of documents and level of detail of the documents varies widely.

It is to be noted that this list of documents is not limitative, and that other documents may be needed in testing projects, such as risk analysis reports, etc. These other documents may also be tailored according to integrity level.

3.3.2 *Testing tasks & intensity per integrity level*

As with IEEE829:1998, the different testing tasks required are those needed to produce the docu-mentation, and any intermediate data that is needed. IEEE829:2008 proposes a table (table 3 pages 23-29) listing the tasks to be executed according to the integrity level. This table covers all processes specified in ISO/IEC12207:2008 (Acquisition, Supply, Development, Operation and Maintenance). An example is given below:

Integrity level 4 : Catastrophic	Integrity level 1 : Negligible
Master Test Plan	-- intentionally left blank, nothing mandatory --
Level Test Plan (Component, Component Integration, System, Acceptance)	Level Test Plan (Component Integration, System)
Level Test Design (Component, Component Integration, System, Acceptance)	Level Test Design (Component Integration, System)
Level Test Case (Component, Component Integration, System, Acceptance)	Level Test Case (Component Integration, System)
Level Test Procedure (Component, Component Integration, System, Acceptance)	Level Test Procedure (Component Integration, System)
Level Test Log (Component, Component Integration, System, Acceptance)	Level Test Log (Component Integration, System)
Anomaly Report	Anomaly Report (Component Integration, System)
Level Interim Test Status Report (Component, Component Integration, System, Acceptance)	-- intentionally left blank, nothing mandatory --
Level Test Report (Component, Component Integration, System, Acceptance)	Level Test Report (Component Integration, System)
Master Test Report	-- intentionally left blank, nothing mandatory --

Test Activities	Life Cycle Processes																																			
	Acquisition (5.2)				Supply (5.3)				Development (5.4)																				Operation (5.5)				Maintenance (5.6)			
	Acquisition Support Test (5.2.1)				Planning (5.3.1)				Concept (5.4.1)				Requirements (5.4.2)				Design (5.4.3)				Implementation (5.4.4)				Test (5.4.5)				Installation/Checkout (5.4.6)				Operation Test (5.5.1)				Maintenance Test (5.6.1)			
	Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level				Integrity Level			
	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1
System Test Plan generation													x	x	x	x																								
System Test Procedure generation																					x	x	x	x																
System Test Report generation																									x	x	x	x												
System test execution																									x	x	x	x												
Task iteration																																					x	x	x	x
Test Readiness Review participation																					x	x			x	x														
Master Test Report generation																													x	x	x	x					x	x		
Test Traceability Matrix generation									x	x	x		x	x	x		x	x	x		x	x	x		x	x	x													

Additional testing tasks that should be envisaged are also proposed by IEEE829:2008, per life cycle phase, and described in more detail in the standard (Annex D, pages 90-96). Among those additional tasks, we have [non-limitative list]: audits, branch coverage, database analysis, independent risk assessment, peer reviews, performance monitoring, regression analysis, certification, documentation evaluation, …

3.3.3 Tester independence

Tester independence is an important aspect that has been promoted in the ISTQB suite of syllabi, and which is a fact of life when studying the offshore testing markets.

The new IEEE829:2008 standard suggests a degree of independence between testers and developers according to the integrity level of the software / system being tested. This can be used as a guideline when decisions must be taken to outsource some testing activities.

Independence can be defined as (Te) Technical independence, (Ma) Managerial independence or (Fi) Financial independence. The test organizations proposed by IEEE829:2008 are :

- Embedded: where the tester is embedded in the development organization, such as a developer testing his/her own code
- Internal: where the tester is considered as inside the development team, but with a specific testing activity
- Integrated: where the testing organization is integrated in the same organization as de-velopment, but as a separate sub-organization
- Modified: where the testing organization is in the same overall organization, but not under the same direct management as the development team.
- Classical: where the testing organization is fully independent from the development or-ganization, either as a sub-contracted organization, or with the development being sub-contracted.

Alternative	Technical	Manage-ment	Financial
Classical	R	R	R
Modified	R	C	R
Integrated	C	R	R
Internal	C	C	C
Embedded	M	M	M

NOTE - R = Rigorous independence; C = Conditional independence; and M = Minimal independence.

The degree of independence varies between (R) Rigorous independence, (C) Conditional independence, and (M) Marginal independence as described in table F1 of IEEE829:2008.

One can compare this suggested level of independence with the one proposed by the ISTQB® in its Foundation Level syllabus. There are no major differences between the two.

In terms of the types of independence (Technical, Managerial or Financial), the reader will be able to evaluate the impact of each type of independence on the actual ratio of independence between developers and testers.

3.3.4 Test Level

Organization of the test documentation is always a problem. Should one group/segregate work according to the types of tests (static vs. dynamic), or according to the testing techniques (EQP, BVA, …), according to ISO9126 characteristics (functional, maintenance, etc), or according to the different testing levels (Component/Unit, Component Integration, System and Acceptance)? This new standard allows any type of organization, but it suggests one: Horizontal levels according to a standard SDLC.

IEEE829:2008 proposes to associate the documentation to the different test levels of the software / system being developed and tested. This implementation, which can be tailored to fit multiple integration levels, is detailed and ways to implement it according to the integrity level are also provided.

IEEE829:2008 suggests horizontal levels, going up from Component, to Component Integration, to System and up to Acceptance. This fits with the V-Model SDLC, and is fully in line with the ISTQB-proposed organization.

The Component test level, as well as the Component Integration test level have a different scope depending on the granularity of the component. If components are low-level, such as DLLs, the next level up is integration of the DLLs. If components are parts of a larger frame of reference, such as within a System-of-Systems, the components can be complete systems developed by suppliers, and the integration should be seen as "system integration" at the customer level.

Similarly, any one level suggested by the standard can be translated in more than one level for the organization, such as Acceptance being split in "Supplier System Acceptance" and "User Acceptance".

Your software and system documentation organization is fully customizable, and the standard can be adapted to your own context and organization.

The documentation suite for each test level comprises:

- The Level Test Plan (LTP) describing the scope of the test level, resources and methods
- The Level Test Design (LTD) providing more details/updates for the test methods
- The Level Test Case (LTC) with inputs and outputs, and
- The Level Test Procedures (LTPr) defining the test setups and execution instructions.

This implies that a full documentation suite will include Level Test Plans, Level Test Design documents, Level Test Cases and Level Test Procedures for all defined levels, resulting in a large documentation set.

IEEE829:2008 allows users of the standard to add, combine, or eliminate "whole documents and/or documentation content topics based on the needs (and integrity level) of their individual systems". This allows the test documentation to be fully tailored to the user context (see also section 3.3.6 below), which was not allowed in the previous (1998) version of the standard. This results in the user being allowed to provide reference to information present elsewhere, eliminate content topics provided in the process or covered by automation tools, combine or eliminate documents.

3.3.5 Customization & compliance

A number of organizations have already implemented IEEE829:1998 or some other type of docu-mentation suite. The adaptation of their existing documentation suite to fit the new standard should not be seen as too large a task, otherwise the changeover to the new standard will not occur.

Contrary to the 1998 version, the new version of the IEEE829 standard is highly adaptable, including in terms of the templates for the documents proposed in the standard. One will remember that IEEE829:1998 considered even the different sections of the documents mandatory. The 2008 version does not consider these sections to be mandatory. This does not mean that anyone can do anything and still be compliant to the standard. It means that the decision to depart from the suggested template must be justified and must be traceable.

As mentioned in a previous section, IEEE829:2008 allows the combination and/or elimination of documents, and suggests that this be done depending on the integrity level. The following table proposes a set of testing documents, depending on the Integrity Level:

The "L", representing "Level" in the titles of the documents should be replaced respectively with "A" for "Acceptance", "S" for "System", "CI" for "Component Integration" and "C" for "Component". The documents can be expanded, combined or eliminated depending on various factors such as duration of the development life-cycle, complexity of the system, higher level of integrity, number of test levels, outsourcing and geographically dispersed organization, personnel turnaround, etc.

Integrity Level	Selected documents
4	MTP, LTP, LTD, LTC, LTPr, LTL, AR, LITSR, LTR, MTR
3	MTP, LTP, LTD, LTC, LTPr, LTL, AR, LITSR, LTR, MTR
2	LTP, LTD, LTC, LTPr, LTR, LTL, AR, LITSR, LTR
1	LTP, LTD, LTC, LTPr, LTL, AR, LTR

IEEE829:2008 (chapter 6) proposes a number of ways to customize the standard to the user's needs.

Compliance to the standard can be claimed relatively easily, as the templates in the standard are not mandatory. It is thus relatively simple for the user to define a mapping between the user's documentation set and the templates. It is, however, necessary for the different topics to be "ad-dressed". IEEE829:2008 defines "addressed" as "*making a decision as to whether a documentation content topic is to be documented prior to the test execution (in a tool or not in a tool), documented post-test execution, not documented (addressed by the process), or not included. "Included" means that either the information is present or there is a reference to where it exists.*". It is thus necessary to document – for traceability and auditing purposes – the decision that led to a topic being "addressed" (or not) in the documentation.

Compliance to ISO/IEC12207 is attained by way of a traceability matrix describing the relationship between IEEE829:2008 and the ISO/IEC standard (Annex H of IEEE829).

3.3.6 Adaptability to context

The implementation of a standard in an organization should not entail major changes in the organ-ization's own processes. This standard provides suggestions on how to adapt the standard docu-mentation set to the organization's specific context. This enables the standard to be adaptable in terms of scope, depth and breadth.

The list of suggested testing tasks and processes, per integrity level, present in IEEE829:2008 enables compliance to be attained quickly, both in terms of process compliance and in terms of definition of the necessary tasks to attain compliance.

Process Compliance: In term of processes, the different processes are fully explained in the different sections of the standard, and these processes are fully in line with the ISO/IEC12207 software life cycle processes and with ISO/IEC15288 system life cycle processes. Standard processes in ISO/IEC12207 include : Management, Acquisition, Supply, Development, Operation and Maintenance.

- *Management processes*, where the MTP and LTP are generated and the test activities are monitored and evaluated continuously. Management processes include review activities, interfaces with organizational and supporting processes, and identification of process improvement activities.
- *Acquisition processes*, where the need to acquire a system, software product or service is defined, refined through the creation of a Request for Proposal (RfP) and the selection of a supplier. This process includes supplier management, from contract execution to acceptance, including testing of the supplied product or service (incl. associated do-cumentation).
- *Supply process*, from the decision to respond to a RfP, all the way to signing the contract, determining the required resources, preparation, execution and delivery of the software-based system. This process uses the contract requirement and overall schedules to revise and update the test interfaces planning between supplier and acquirer.
- *Development process*, consists of the activities and tasks of the development group, and includes analysis, design, code, component integration, testing, installation and acceptance related to the software or software-based product. It would be in this process that the applicable integrity level is evaluated and the traceability matrix generated.
- *Operations process*, covers the operation of the software product and operational support to the different users. This process performs operational testing, system operation, and user support. In this process we will also find risk identification and management (mitigation or avoidance), as well as identification of security issues.
- *Maintenance process*, which is related to any modification to the code or associated documentation, caused by identified problems, required improvements or adaptations. Usually any modification will be treated as a development process and verified as any of the other processes.

Tasks associated to processes: The tasks associated to the different testing processes are described in the standard (Annex C), including their required inputs and the produced outputs. It is thus easy to describe the required tasks and define the scope of activities that need to be executed (subcontracted, off-shored or done internally) and verified in terms of compliance to a service level agreement.

The user is thus guided in the implementation of the standard, with the necessary adaptation to integrity levels and to present (or perceived) risks.

Claiming compliance from IEEE829:1998 is possible by mapping the old (1998) version of the standard to the new (2008) version. This is relatively straightforward both in terms of number and scope of documents and in terms of content of the different sections of the documents. As the templates of ISO829:2008 are not mandatory (i.e. normative), compliance can be obtained relatively easily. Some differences appear in that the ITR (Item Transmittal Report) is not present in the new version of the standard, and in that there is a lack of "Approval" sections in the different documents.

Both of these can be implemented in your environment via customization of the standard.

Formal compliance can be claimed if all the topics in the standard have been "addressed" (see 3.3.5 above) and can be documented as having been covered.

3.4 Other changes

Beyond the changes from a document-centric to a process-centric standard, a number of more subtle changes have occurred in IEEE829:2008; some of these are listed below (non-exhaustive list):

- "Incident Report"is renamed to "Anomaly Report"
- "Item Transmittal Report" has been removed from the IEEE829:2008 documentation set. This does not mean that an item should be transmitted for testing any way you want, but that the formalism is no longer required
- "Level Interim Test Status Reports" are now available to provide interim reports on the test campaigns for each level. One is not limited to a single test report.
- Metrics are emphasized for Test Management purposes, and a specific "Master Test Plan Metrics Report" template is provided. Metrics can be selected from such standards as ISO/IEC9126-2:2003 External Metrics, ISO/IEC9126-3:2003 Internal Metrics and/or IEO/IEC9126-4:2004 Quality in Use Metrics.

In order to be complete, it is necessary to add that the IEEE829:2008 standard provides explanations for a number of key concepts that are emphasized in the standard:

- Integrity Levels (see also 3.3 above)
- Recommended minimum testing tasks for each integrity level (see also 3.3.2 above)
- Intensity and rigor applied to testing tasks
- Detailed criteria for testing tasks
- Systems viewpoint
- Selection of test documentation
- Compliance with International and IEEE standards.

4 Conclusion.

To the international testing community, the IEEE829:2008 is an important deliverable that is the result of a joint effort by different schools of testing to come up with a coherent and consistent set of documentation that can fit multiple contexts of software and system test.

As with any human endeavour, some points can still be improved. This article attempts to list the positive and less positive aspects and provides a number of pointers on how to avoid the negative points.

Mapping the old and new IEEE829 standards is straightforward and should not pose any problem to users who will be able to implement the new standard and convert existing documentation to fit the new IEEE829:2008 standard. This will enable the audience to ensure consistency of their do-cumentation across different projects, and across different versions of their software, also when outsourc-

ing either in NearShore or OffShore, and irrespective of whether working on simple systems or complex systems-of-systems.

The shift of focus from a document-oriented standard to a tasks-oriented standard, and its compliance with other standards, such as ISO/IEC12207:2008 and ISO/IEC15288:2008, ensure that the IEEE829:2008 standard will become as successful as its previous version.

References

[IEEE829:1998] IEEE Standard for Software Test Documentation, © IEEE, 1998

[IEEE829:2008] IEEE Standard for Software and System Test Documentation, © IEEE, 2008

[ISO12207:2008] Systems and software engineering — Software life cycle processes, © ISO/IEC, 2008

[ISO15288:2008] Systems and software engineering — System life cycle processes, © ISO/IEC, 2008

[DO178B/ED12B] Software Considerations in Airborne Systems and Equipment Certification, © RTCA/EUROCAE 2000

Biography

Founder and principal consultant for TESSCO Technologies Inc., Bernard Homès is a former member of the board of IEEE (France) and active in various software testing associations and workgroups.
After 25+ years in software development and testing with different consultancies, he set up the TESSCO group consultancies and provides training and consultancy services worldwide.
The fields of operation of TESSCO Technologies Inc. includes :

- Training testing teams & setting up software test centers,
- Specialization in mission-critical systems (banking, health care, telecoms, space & airborne systems),
- Qualification of Systems of Systems (large & complex critical systems) for international customers (such as Orange, Alcatel Alenia Space, Eurocopter).

Bernard Homès is also President of the French Software Testing Board (www.cftl.fr) and represents France at the ISTQB.
A long-time speaker at different international conferences and universities, Bernard has acquired a number of software testing certifications, and chairs the ISTQB Advanced Level syllabus working party.

He can be contacted at bhomes@tesscogroup.com

Standards in Medical IT

by Dr. Anne Kramer & Georg Götz

On November 2nd, 2009 Heidelberg University Hospital officially inaugurated the first German ion beam therapy centre. The news has again raised the public interest in development of new medical devices. In this context, a paradox can be observed: while most people trust the technical equipment of a hospital, few people are satisfied with their treatment and service. The hospital workflows are perceived as being inefficient with long waiting times and redundant examinations. We believe that this is mainly due to the hospitals' suboptimal IT infrastructure that slows down the transmission of examination results.

The discrepancy between confidence on one hand and scepticism on the other hand is even more surprising, since both clinical workflows and the development of medical devices are supported by international standards. However, the significance of these standards varies. The development process of new medical devices is subject to strict regulations. ISO standards such as the Risk Management Standard DIN EN ISO 14971 are virtually law-like. Disregarding these laws and standards for processes can have severe consequences for the manufacturer, whereas the standards for interoperability of clinical IT and standards to support clinical workflows are not compulsory. Also, they are hard to implement, since hospitals usually have a grown infrastructure of IT applications with legacy systems.

In this article we give an overview of the different standards that are relevant for medical devices and medical IT. The first section deals with standards that apply to development processes – with particular focus on quality assurance. The second part is dedicated to technical standards on the interoperability of medical IT.

Process Standards for Medical IT Development

As mentioned in the introduction, various national and international regulations apply to the development of medical devices. An overview is given in table 1.

MDD 93/42/EEC	Medical Devices Directive European directive for medical device manufactures
ISO 13485	Quality Management System for design and manufacturing of medical devices (related to ISO 9001)
ISO 14971	Risk Management System for medical devices
ISO 62304	Life Cycle Management for medical device software
ISO 60601-1	Medical Electrical Equipment Part 1: General Requirements for Safety
FDA 21 CFR Part 820	Quality System Regulation
FDA 21 CFR 11	Electronic Records and Electronic Signature

Table 1: Most important process regulations

The code of federal regulations title 21 of the American Food & Drug Administration (FDA) is of pre-eminent importance. The FDA is the public authority that certifies new medical devices and by this approval decides whether the product may be sold on the US market or not. Therefore, the requirements of the FDA are an absolute must for every globally acting manufacturer. Findings during one of the regular controls might lead to heavy sanctions, and - unlike in other cases - the assumption of innocence does not apply. It is up to the manufacturer to prove that all necessary processes have been implemented and that they have been followed correctly. Comprehensive documentation must be provided for every plan, result and decision. Since the FDA regulations are rather concise, additional guidance is given with recommendations for process and documentation. A list of required documentation for the "Pre-market Submission for Software Contained in Medical Devices" is given in reference [1]. "General Principles of Software Validation" are given in [2].

The degree of documentation and testing depends on the risk that an injury may be caused by the device (level of concern). Minimum documentation covers the rationale leading to the level of concern, the device hazard analysis, a concise description of the software including functional requirements, a traceability analysis, a functional test plan with pass/fail criteria and results and the revision level history. For a higher level of concern additional documentation on requirements, architecture, design, development environment, verification and validation activities and bugs is required [1].

Providing traceability among requirements, specifications, hazards and tests (verification and validation) is one of the core topics. Another one is the proof that the degree of testing has been sufficient. This can be challenging if the number of tests is large or if product variants have to be tested. Usually, not all tests are executed for all possible configurations. Instead, a rationale is given why tests have been limited. This approach saves effort in test execution, but increases the time spent for test management. Here, a test management tool is helpful. By using a tool, it becomes easier to prove the completeness of tests and to provide objective evidence that test cases have been left out for a good reason – and not just forgotten. While we strongly recommend the use of a tool, we also recommend evaluating the specific needs of your company (or project) and the required interfaces to existing applications first. Specific attention should be paid to the support of product variants, configuration

management and test result reports. To comply with FDA regulations, it should be possible to generate a report for all tests related to hazards etc.

Figure 1 shows the most popular process framework in medical IT development, the so-called V-model. The required design activities are shown to the left, the verification and validation activities to the right. Traceability must be established both vertically and horizontally. Figure 1 also shows the gap between the vendor and the clinical environment. In the clinical domain, the components sold by the vendor have to be integrated into a workflow-based solution. While standardized workflows exist (and are described below), reality teaches us that the clinical usage varies significantly from one clinical environment to another and usually deviates from the standards. Typically, the communication gap between vendors and users is significant, leading to problems for system integration and system quality. Thus, to perform efficient quality assurance, advanced testing and test design methods have to be applied.

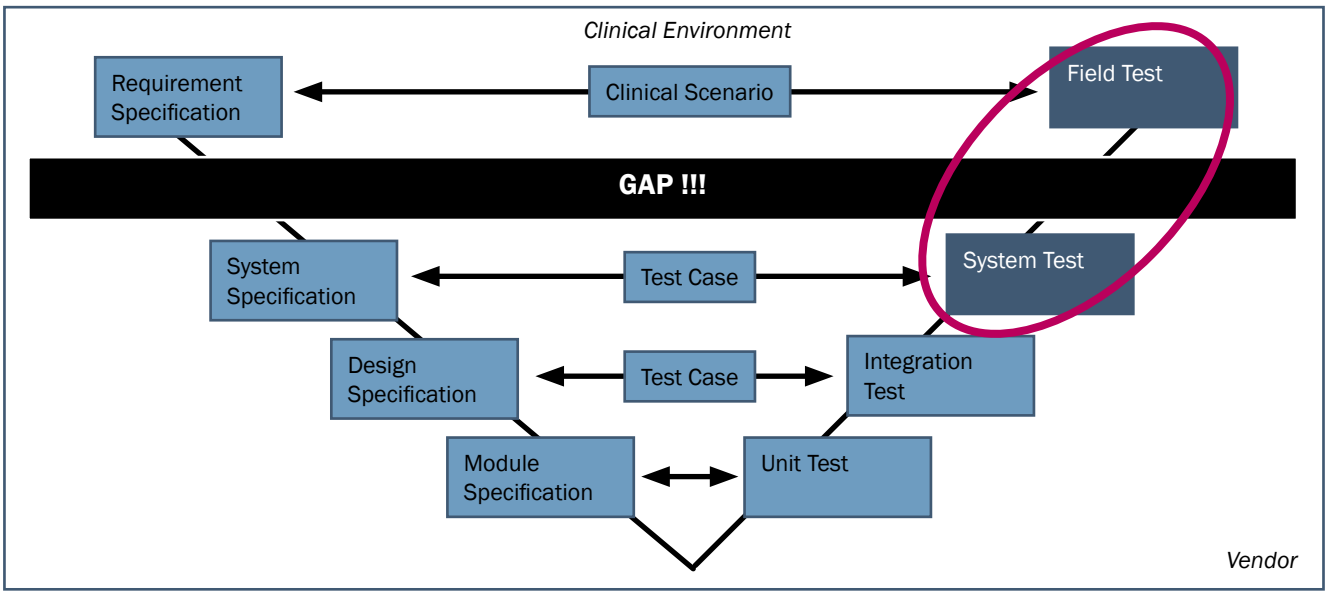


Figure 1: Gap between vendor and clinical environment

Advanced QS Methods for Medical IT Development

Using tools helps to manage test cases, but not to define the right ones. Also, the tool will not provide a rationale why certain test cases can be left out. The concept of model-centric testing proved to be a good way of designing and documenting tests [3]. With this method the behavior of the system under test is described with models (e.g. activity or state diagrams), taking the tester's mindset into account. Each path through the model corresponds to a test case. The model focuses on system usage rather than on system specification. Unlike in model-based testing, the test design model contains all the information relevant for the test. This includes test management information as well as test data. Figure 3 illustrates how model-centric testing is integrated in the process and tool chain. Note that the process chain depends on the specific project requirements.

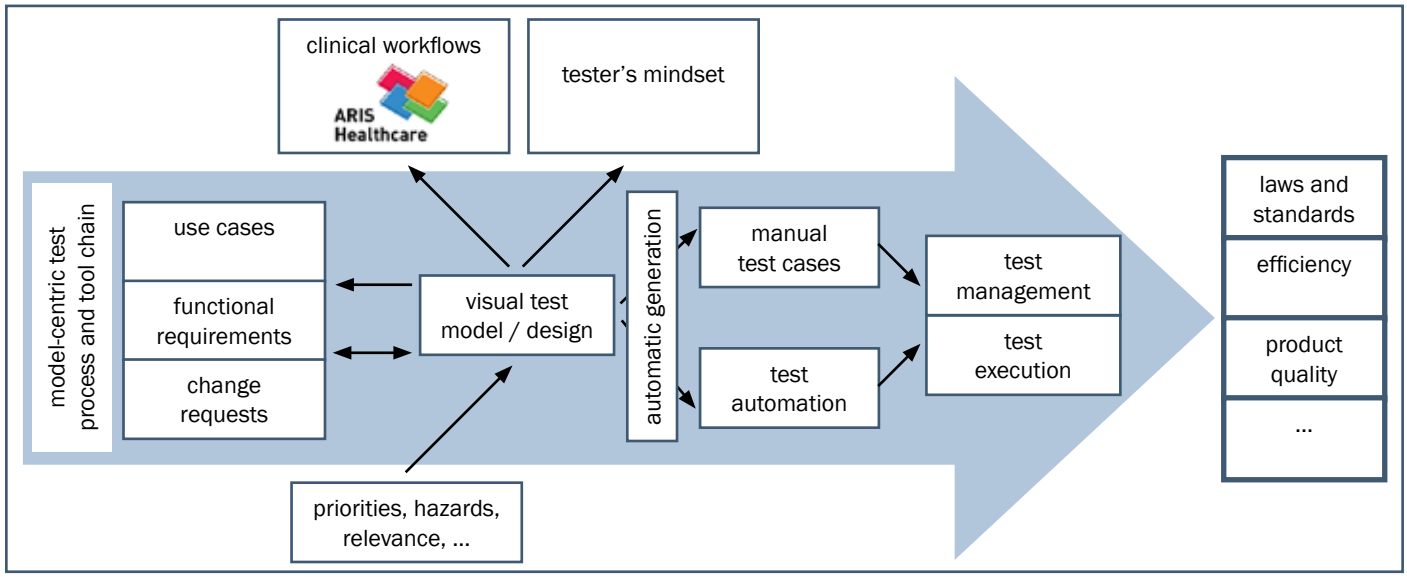


Figure 2: Process and tool chain with test design model as central repository

The model constitutes the basis for all discussions and is an essential part of the documentation. This helps in closing the communication gap between vendor and clinical environments, and thus increases both system quality and efficiency of system integration. The model establishes the traceability to requirements and hazards, which are linked to the corresponding states or activities. Thus, it becomes literally "visible" which test cases are hazard-relevant. The tester's mindset, i.e. the reasoning why tests have been designed the way they have, is documented and becomes comprehensible for others – in particular for the auditor. Finally, the use of models is also economically interesting. Less time is spent during review, because pictures are much easier to understand than text. Also, tools exist on the market to generate manual or automated test cases automatically from a model. Thus supported by appropriate tools and methods, the testing process can be efficiently integrated into the development process as required by regulations and laws (see also fig. 3).

Standards and Interoperability in Medical IT Infrastructures

In the last decades IT has revolutionized the processes and workflows in the medical domain, especially in clinical environments. Cost pressures and service requirements by both staff and patients force medical institutions to adapt their technical infrastructure.

Clinics are moving away from isolated, task-specific solutions and paper records. The reasons are often technical advancements like CT scanners with their immense amounts of data produced, the need to store data digitally and make it available outside of the scanner system, e.g. on diagnostic workstations. For this purpose, dedicated systems were developed and networked. For example, scanners deliver their images to a storage system called Picture Archiving and Communication System (PACS). Radiology information systems (RIS) handle work lists and scheduling.

This trend continued with the extension of these radiological networks to other departments. Planning and scheduling tasks had to become more efficient. Thus, sharing of information between different departments became necessary. Hospital Information Systems (HIS) were introduced as the core of hospital IT infrastructure. With the radiological networks and HIS in place, the current trend is to integrate further departments with their up-to-now isolated systems.

Interoperability Standards in the Medical Domain

The first communication protocols used for connecting IT systems were vendor-specific, proprietary standards. Due to the increasingly networked infrastructure including systems from different vendors and generally a heterogeneous environment, the development of common standards became necessary.

Beginning in the 1980s, the Digital Imaging and Communications in Medicine (DICOM) standard was defined for transmitting, handling, storing and printing imaging information. DICOM defines the data to be exchanged and its format as well as services that DICOM-enabled devices or systems should provide. Most of these services involve network data transmission. Examples are "store services" for sending images to a storage device or "query/retrieve services" for finding and retrieving images from PACS.

In other clinical domains, different types of information and workflows were focused on. Patient information (names, demographic data, diagnostic information …) needs to be gathered, transmitted and stored. The HL7 (Health Level 7) format is designed to handle these tasks. It contains conceptual standards (Reference Information Model - RIM), document standards (Clinical Document Architecture – CDA) and messaging standards (HL7 v2.x and v3). HL7 v 2.x is designed to support clinical processes including administrative, financial and medical tasks. It is the most widespread version of the messaging standard and fully backwards compatible. Therefore, it is supported by most major manufacturers of medical systems. The newest version HL7 v3 is based more on formal, object-oriented principles and realized with an XML-based syntax.

To integrate systems into the complex and dynamic workflows of healthcare, specifications for their behavior are required. The current standard defining technical workflows at system level are the IHE (Integrating the Healthcare Enterprise) profiles. The profiles define actors/roles and their communication behavior with regard to the systems' domain within healthcare (radiology, laboratory…). They also specify message protocols and types to be used by referencing other standards like DICOM and HL7. Obviously, the IHE profiles are ideally validated following the model-centric testing approach. Figure 3 shows how IHE-based systems can be validated with usage models based on standard workflows. Note, that versioning, prioritization and all possible user workflows are contained in the model.

Integration of Standard-based Systems

Despite evolving standards and their growing acceptance by both users and vendors, integrating medical systems into hospital networks is still an intricate task. It is often possible to interpret certain aspects of the standard. Also, the standards are not always complete. As healthcare institutions are individually structured with different workflows and infrastructures, the standards usually cannot be implemented literally, but need to be adapted to site-specific requirements. In other words, even if a system has been developed respecting the technical standards, tailoring is required during its integration into the clinical environment.

Good communication between user and vendor is necessary to ensure time and cost efficient integration phases. However, the different views of the more functional and standard-oriented manufacturers and the customers/clinics thinking in their workflows and proprietary

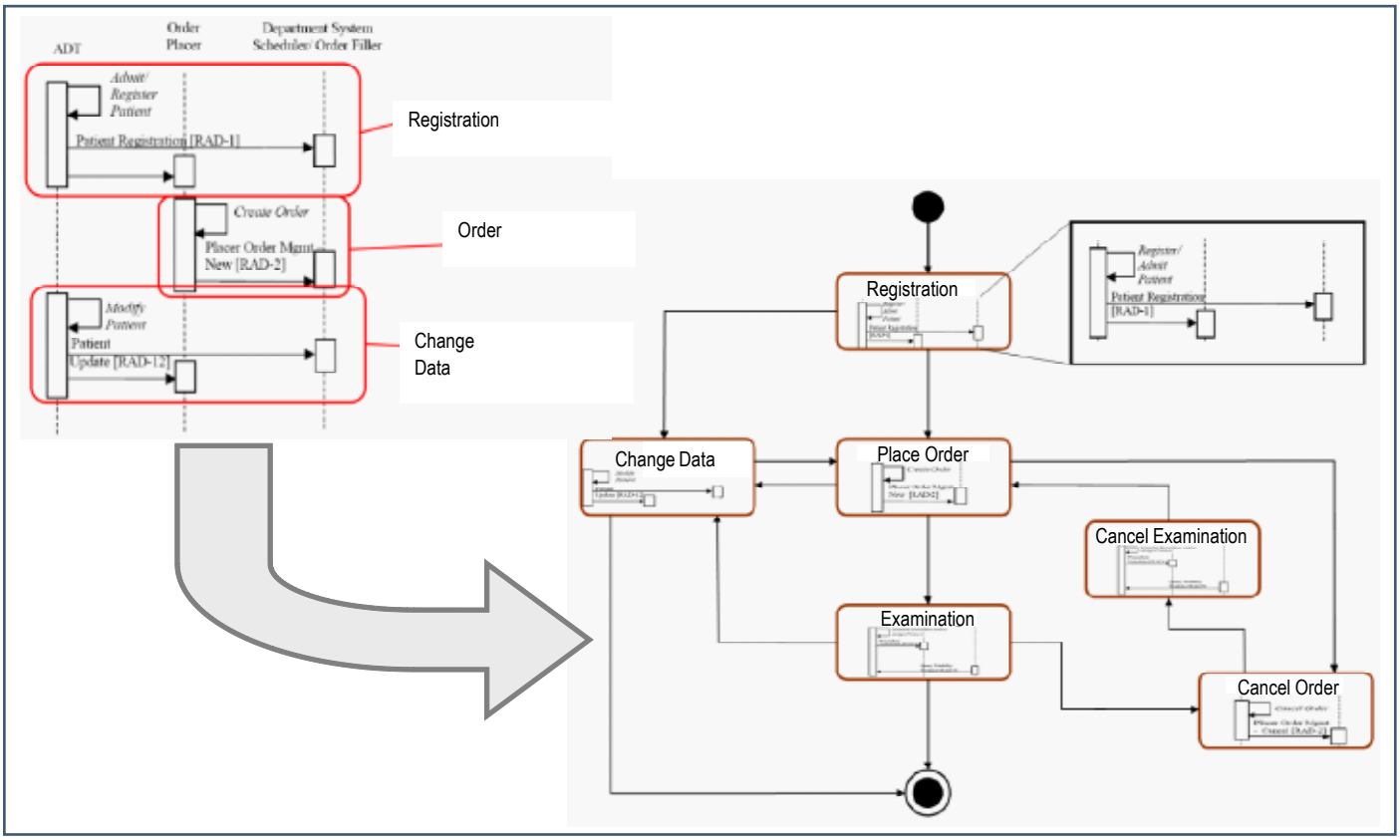


Figure 3: Usage models (right side) based on standard workflows (left side) for validating IHE-based systems

scenarios hamper mutual understanding. Developing a common language should be a major goal of standardization organizations, vendors and end users. The visual representation of clinical workflows and standard-based system communication using hierarchical models (see figure 3 right side) definitely solves at least parts of the communication issues.

Referring back to the process standards for medical IT development, the FDA explicitly requires testing at a user's site with the actual hardware and software that will be part of the installed system configuration. During user site testing the use of the software is validated within the context in which it is intended to function, a context that is given by the IHE workflows. The communication models mentioned above can also be used to generate test cases. Again, a model-centric test approach proves to be a valuable and efficient quality assurance method.

TestNGMed

A medical IT test bed was established in the context of the European research project "TestNGMed" [4]. In order to overcome the challenges described above, the TestNGMed project is dedicated to the development of a TTCN-3 based (Testing and Test Control Notation Version 3) methodology for validation of interoperability and conformance of IHE- and HL7-based medical systems and components. Using a model-centric test method, the validation focuses on interoperability and conformance tests of the interface between manufacturer and clinic. In contrast to standard document-based specification of test cases, the visual test model provides a common language based on testing and medical standards. It facilitates the understanding, especially for non-technical stakeholders. The automated derivation of TTCN-3 test scripts from the test design supports both full workflow coverage and structured reduction of test cases using test management information stored in the test model.

Standard-based Interoperability in the Future

The complexity of systems and the networks they are integrated in are growing. This happens both on the microscopic level (with sensor level systems being connected to hospital information systems) as well as on macroscopic level where cross-site communication is growing and large infrastructure changes are introduced (e.g. electronic health insurance card). Other upcoming topics are mobile care and home care. Early, effective and flexible validation of standard conformance and interoperability is crucial for the cost-efficient realization of development and integration projects on all scales.

Closing remarks

As stated at the beginning of this article, confidence in medical technology is high. At the Heidelberg ion beam therapy centre, most patients will lie below a 670 ton gantry and are moved around by a robotic arm without hesitation. This confidence is based upon the demanding standards set for development and quality assurance processes. To meet these standards, methods, tools and processes have been developed over the past years. Further evolution can be expected to keep up with new and stricter regulations. The responsibility rests on the shoulders of both manufacturers and customers. Therefore, new approaches (like model-centric testing) have to be employed that integrate stakeholders including the certification authorities.

Since standards for technical interoperability are not enforced by laws, their uptake is not as fast as could be expected. However, economical pressure and clinical process requirements demand more and stable system interoperability and abilty to integrate. Driving standardization forward with standardized test suites is necessary, but not sufficient. Usually, individual clinical networks are at least partially in place. Therefore, adaptability of both standards and system integration has to be provided. It is crucial that process standards, as e.g. TestNGMed bridge the gap between clinics and vendors.

[1] "Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices", May 11, 2005, http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm089543.htm

[2] "General Principles of Software Validation; Final Guidance for Industry and FDA Staff", January 11, 2002, http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/ucm085281.htm

[3] "Experiences with Model Centric Testing in Standard Based Medical IT Environments", K.-H. Kühnlein, G. Götz, A. Metzger, M. Seel, "Software Quality Engineering", Proceedings of the CONQUEST 2009, dpunkt.verlag

[4] http://www.testngmed.org

All links called on Nov 18th, 2009.

Biography

Anne Kramer studied Physics at the University of Hamburg. In 1995 she received her PhD at the Université Joseph Fourier in Grenoble (France). She started working for Schlumberger Systems in Paris - first as software developer for smart card tools, then as project manager for point of sales terminals. In 2001 she joined sepp.med, a service provider specialized in IT solutions with integrated quality assurance in complex, safety-relevant domains. Currently, Anne Kramer is working as project manager and process consultant.

Georg Götz completed his degree in computer science in 2005. Since then he has been working in the areas of quality assurance and software development within complex and safety relevant domains at the sepp.med gmbh. His field of activities there includes working in research and development projects with a focus on model centric methods and processes in medical IT environments. In addition he oversees diploma and master thesis projects and works as a trainer for Certified Tester courses.

The consequences if testing standards are not followed

by Julia Hilterscheid

Software bugs have in the past frequently had catastrophic consequences, which have led to death, impairment of health or immense financial losses. The list of impacts which occurred as a result of faulty programming is long.

Earlier this year the Federal Court of Germany (BGH) had to decide whether a car manufacturer is liable for damages caused by an airbag triggered accidentally as a result of a development error. The claimant in this case made a claim on the defendant, i.e. the manufacturer of his car, for payment of compensation for pain and suffering and for determining whether an obligation exists to pay damages for any future material and immaterial damage. The claimant had experienced an accident which was caused by the faulty triggering of the two side airbags on the passenger side of his car when driving through a pothole in the road, which caused him to swerve onto a dirt track. He stated during the court proceedings to have incurred injuries on the cervical artery which resulted in a cerebral infarct.

The degree of danger and the significance of the endangered object of legal protection were stated in the BGH's decision as the decisive factors with respect to the requirements and content of the obligations to provide instructions and reasonable safeguarding measures. According to the BGH, required measures are considered to be all those safeguarding measures, which at the time of bringing the product to the market were constructionally feasible according to the latest technical and scientific states-of-the-art and which were at the time deemed suitable and sufficient to prevent damage.

After the claimant failed with his claims in the lower courts, the BGH clarified that the defendant would have had the opportunity to avoid the danger, since specialized expert knowledge would have been available, and this would have prevented the erroneous triggering of the airbag. In view of the dangers to life and limb of drivers and passengers resulting from the accidental activation of airbags, it was the duty of automotive manufacturers to eliminate the risk that this type of malfunction occurs in the vehicles produced by them by taking all necessary constructive measures within the bounds of what is technically possible and economically reasonable. The vehicle manufacturer and defendant had in the view of the BGH not satisfied these demands, so that the software had to be characterized as faulty.

Previously, a variety of generally valid or industry-specific (e.g. pharmaceutical, military) standards were applied when it came to the testing of software.

This is interesting from a legal point of view, since the creation or sale of software is usually based on a contract, which normally stipulates the requirements for the software to be created or purchased. Furthermore, it is required by law that the software producer must deliver the software to the customer free of material defects and defects of title. The software satisfies this demand if it meets all criteria agreed by the parties for the work product at the time of conclusion of contract. Regarding characteristics which were not specified or were not sufficiently specified, which regrettably still happens in practice, the legal definition applies. According to this, the software is free from quality defects if it is suitable for the purpose stipulated in the contract or for its normal usage, and provides the properties which can be expected from software of the same type.

On the basis of these definitions, the BGH has declared the liability of the defendant.

In areas where the use of the software is associated with high risks to life and limb, as well as with financial and economic losses, a method called „verification" is used (amongst others) to avoid defects; this method proves formally/mathematically that the prerequisites required from the software have been fulfilled. This method, however, is somewhat limited due to the large effort it entails. In the case of more complex programs it is almost impossible to adhere to this method, which leads to the presence of software defects and to the corresponding consequences as described above.

The conclusion to draw is that in view of the impending consequences caused by faulty software, high demands should generally be made with regard to the standards of testing. This applies in particular for the more important objects of legal protection which would be affected by any software defects. Whether or not this is of any comfort to the people affected by software defects, remains to be seen.

Ms. Hilterscheid has been solicitor with practising licence since 1997. After stays abroad and studies in law in Berlin, she founded first the solicitor's office Hilterscheid, and one year later, together with her husband, the consultancy Díaz & Hilterscheid Unternehmensberatung GmbH. In addition, Ms. Hilterscheid has been teaching media law and copyright at the Berlin University of Applied Science.
Ms. Hilterscheid has been supporting enterprises for more than 10 years in the discretionary formulation of contracts, general terms and conditions and license agreements, and accompanies projects legally. She ensures for her clients that legal requirements are adhered to in their correspondence, on-line appearance as well as in their advertising and marketing activities.
Ms. Hilterscheid also offers seminars on the subjects of IT-law, trademark law, copyright and labor law, which can take place in-house if so desired.

www.kanzlei-hilterscheid.de

Berlin, Germany

IT Law
Contract Law

German
English
Spanish
French

www.kanzlei-hilterscheid.de
info@kanzlei-hilterscheid.de

k a n z l e i h i l t e r s c h e i d

Do standards keep testers in the kindergarten?

by James Christie

What are standards?

Discussion of standards usually starts from the premise that they are intrinsically a good thing, and the debate then moves on to consider what form they should take and how detailed they should be.

Too often sceptics are marginalized. The presumption is that standards are good and beneficial. Those who are opposed to them appear suspect, even unprofessional.

I believe that although the content of standards for software development and testing has great value, they should not be regarded as "standards". Turning useful guidelines into standards suggests that they should be mandatory.

My particular concern is that the IEEE 829 "Standard for Software and System Test Documentation", and the many document templates derived from it, encourage a safety- first approach to documentation, with testers documenting plans and scripts in slavish detail. They do so not because the project genuinely requires it, but because they have been encouraged to equate documentation with quality, and they fear that they will look unprofessional and irresponsible in a subsequent review or audit. I think these fears are ungrounded and I will explain why.

A sensible debate about the value of standards must start with a look at what standards are, and the benefits that they bring in general, and specifically to testing.

Often discussion becomes confused because justification for applying standards in one context is transferred to a quite different context without any acknowledgement that the standards and the justification may no longer be relevant in the new context.

Standards can be internal to a particular organization or they can be external standards attempting to introduce consistency across an industry, country or throughout the world.

Let's forget about legal requirements enforcing minimum standards of safety, such as Health and Safety legislation, or the requirements of the US Food & Drug Administration. That's the law, and it's not negotiable.

The justification for technical and product standards is clear. Technical standards introduce consistency, common protocols and terminology. They allow people, services and technology to be connected. Product standards protect consumers and make it easier for them to distinguish cheap, poor quality goods from more expensive but better quality competition.

Standards therefore bring information and mobility to the market and thus have huge economic benefits.

It is difficult to see where standards for software development or testing fit into this. To a limited extent they are technical standards, but only so far as they define the terminology, and that is a somewhat incidental role. They appear superficially similar to product standards, but software development is not a manufacturing process, and buyers of applications are not in the same position as consumers choosing between rival, inter-changeable products.

Are software development standards more like the standards issued by professional bodies? Again, there's a superficial resemblance. However, standards such as Generally Accepted Accounting Principles (Generally Accepted Accounting Practice in the UK) are backed up by company law and have a force no-one could dream of applying to software development. Similarly, standards of professional practice and competence in the professions are strictly enforced and failure to meet these standards is punished.

Where does that leave software development standards? I do believe that they are valuable, but not as standards.

Susan Land gave a good definition and justification for standards in the context of software engineering in her book "Jumpstart CMMI Software Process Improvements - using IEEE software engineering standards". [1]

"Standards are consensus-based documents that codify best practice. Consensus-based standards have seven essential attributes that aid in process engineering. They;

1) *Represent the collected experience of others who have been down the same road*
2) *Tell in detail what it means to perform a certain activity*
3) *Help to assure that two parties attach the same meaning to an engineering activity*
4) *Can be attached to or referenced by contracts*
5) *Improve the product*
6) *Protect the business and the buyer*
7) *Increase professional discipline."* (List sequence re-ordered from original)

The first four justifications are for standards in a descriptive form, to aid communication. Standards of this type would have a broader remit than the technical standards I referred to, and they would be guidelines rather than prescriptive. These justifications are not controversial, although the fourth has interesting implications that I will return to later.

The last three justifications hint at compulsion. These are valid justifications, but they are for standards in a prescriptive form and I believe that these justifications should be heavily qualified in the context of testing.

I believe that where testing standards have value they should be advisory, and that the word "standard" is unhelpful. "Standards" implies that they should be mandatory, or that they should at least be considered a level of best practice to which all practitioners should aspire.

June 7th - 8th, 2010

in Bad Homburg (near Frankfurt am Main, Germany)

Call for Paper

Infos at www.testingfinance.com

or contact us via e-mail info@testingfinance.com

Exhibitors

PureTesting
Testing Thought Leadership

CaseMaker®

SELA GROUP

iSQI®
International Software Quality Institute

Supporting Organisations

VOB SERVICE

Díaz Hilterscheid

te testing experience

Is the idea of "best practice" useful?

I don't believe that software development standards, specifically the IEEE series, should be mandatory, or that they can be considered best practice. Their value is as guidelines, which would be a far more accurate and constructive term for them.

I do believe that there is a role for mandatory standards in software development. The time-wasting shambles that is created if people don't follow file naming conventions is just one example. Secure coding standards that tell programmers about security flaws that they must not introduce into their programs are also a good example of standards that should be mandatory. However, these are local, site-specific standards. They are about consistency, security and good housekeeping, rather than attempting to define an over-arching vision of "best practice".

Testing standards should be treated as guidelines, practices that experienced practitioners would regard as generally sound and which should be understood and regarded as the default approach by inexperienced staff. Making these practices mandatory "standards", as if they were akin to technical or product standards and the best approach in any situation, will never ensure that experienced staff do a better job, and will often ensure they do a worse job than if they'd been allowed to use their own judgement.

Testing consultant Ben Simo, has clear views on the notion of best practice. He told me; *"'Best' only has meaning in context. And even in a narrow context, what we think is best now may not really be the best. In practice, 'best practice' often seems to be either something that once worked somewhere else, or a technical process required to make a computer system do a task. I like for words to mean something. If it isn't really best, let's not call it best. In my experience, things called best practices are falsifiable as not being best, or even good, in common contexts.*

I like guidelines that help people do their work. The word 'guideline' doesn't imply a command. Guidelines can help set some parameters around what and how to do work and still give the worker the freedom to deviate from the guidelines when it makes sense.

Rather than tie people's hands and minds with standards and best practices, I like to use guidelines that help people think and communicate lessons learned - allowing the more experienced to share some of their wisdom with the novices."

Such views cannot be dismissed as the musings of maverick testers who can't abide the discipline and order that professional software development and testing require. Ben is the President of the Association of Software Testing. His comments will be supported by many testers, who see how it matches their own experience. Also, there has been some interesting academic work that justifies such scepticism about standards. Interestingly, it has not come from orthodox IT academics.

Lloyd Roden drew on the work of the Dreyfus brothers as he presented a powerful argument against the idea of "best practice" at Starwest 2009 and the TestNet Najaarsevent. Hubert Dreyfus is a philosopher and psychologist, and Stuart Dreyfus works in the fields of industrial engineering and artificial intelligence. In 1980 they wrote an influential paper that described how people pass through five levels of competence as they move from novice to expert status, and analysed how rules and guidelines helped them along the way. The Dreyfus Model of Skills Acquisition can be summarized as follows.

Novices require rules that can be applied in narrowly defined situations, free of the wider context.

Advanced beginners can work with guidelines that are less rigid than the rules that novices require.

Competent practitioners understand the plan and goals, and can evaluate alternative ways to reach the goal.

Proficient practitioners have sufficient experience to foresee the likely result of different approaches and can predict what is likely to be the best outcome.

Experts can intuitively see the best approach. Their vast experience and skill mean that rules and guidelines have no practical value.

For novices the context of the problem presents potentially confusing complications. Rules provide clarity. For experts, understanding the context is crucial and rules are at best an irrelevant hindrance.

Roden argued that we should challenge any references to "best practices". We should talk about good practices instead, and know when and when not to apply them. He argued that imposing "best practice" on experienced professionals stifles creativity, frustrates the best people and can prompt them to leave.

However, the problem is not simply a matter of "rules for beginners, no rules for experts". Rules can have unintended consequences, even for beginners.

Chris Atherton, a senior lecturer in psychology at the University of Central Lancashire, made an interesting point in a general, anecdotal discussion about the ways in which learners relate to rules.

"The trouble with rules is that people cling to them for reassurance, and what was originally intended as a guideline quickly becomes a noose.

The issue of rules being constrictive or restrictive to experienced professionals is a really interesting one, because I also see it at the opposite end of the scale, among beginners. Obviously the key difference is that beginners do need some kind of structural scaffold or support; but I think we often fail to acknowledge that the nature of that early support can seriously constrain the possibilities apparent to a beginner, and restrict their later development."

The issue of whether rules can hinder the development of beginners has significant implications for the way our profession structures its processes. Looking back at work I did at the turn of the decade improving testing processes for an organization that was aiming for CMMI level 3, I worry about the effect it had.

Independent professional testing was a novelty for this client, and the testers were very inexperienced. We did the job to the best of our ability at the time, and our processes were certainly considered best practice by my employers and the client. The trouble is that people can learn, change and grow faster than strict processes can adapt. A year later and I'd have done it better. Two years later, it would have been different and better, and so on. Meanwhile, the testers would have been gaining in experience and confidence, but the processes I left behind were set in tablets of stone.

As Ben Simo put it; "If an organization is at a level less than the intent of level 5, CMM seems to often lock in ignorance that existed when the process was created".

CMMI has its merits, but also has dangers. Continuous process improvement is at its heart, but these are incremental advances and refinements in response to analysis of metrics. Step changes or significant changes in response to a new problem don't fit comfortably with that approach. Beginners advance from the first stage of the Dreyfus Model, but the context they come to know and accept is one of rigid processes and rules.

Rules, mandatory standards and inflexible processes can hinder the development of beginners. Rigid standards don't promote quality. They can have the opposite effect if they keep testers in the kindergarten.

IEEE829 & the motivation behind documentation

One could argue that standards do not have to be mandatory. Software developers are pragmatic, and they understand when standards should be mandatory and when they should be discretionary. That is true, but the problem is that the word "standards" strongly implies compulsion. That is the interpretation that most outsiders would place on the word. People do act on the assumption that the standard should be mandatory, and then regard non-compliance as a failure, deviation or problem. These people include accountants and lawyers, and perhaps most significantly, auditors.

My particular concern is the effect of IEEE 829 testing documentation standard. I wonder if much more than 1% of testers have ever seen a copy of the standard. However, much of its content is very familiar, and its influence is pervasive.

IEEE 829 is a *good* document with much valuable material in it. It has excellent templates, which provide great examples of how to document meticulously a project. Or at least they're

For us, Agile is more than a buzzword ...

SELA – Leading in offering blended management solutions, based on traditional and Agile methodologies

Becoming Agile is no longer optional for organizations wishing to stay competitive in today business. SELA helps organizations to perform such a transition as smoothly as possible. Our combination of unique courses, consulting services and practical expertise helps our customers to speed up the transition and maximize the benefits of becoming more Agile. We will help you choosing the ideal combination of Agile and traditional methods that fits most your business needs

Available Courses:

- Agile Testing
- Practical Scrum
- Scrum Master Certification Course
- Scrum Product Owner Certification Course
- Test Driven Development for C++ Developers
- Test Driven Development for Java Developers
- Test Driven Development with .NET
- Test Driven Development with VSTS

Solutions are available around the world. Local solutions are also available in:

Argentina | Canada | Germany | India | Israel | USA | Singapore

solutions@sela.co.il
www.selagroup.com

SELA TECHNOLOGY CENTER

Microsoft GOLD CERTIFIED Partner

Learning Solutions
Custom Development Solutions
Networking Infrastructure Solutions

great examples of meticulous documentation if that is the right approach for the project. That of course is the question that has to be asked. What is the right approach? Too often the existence of a detailed documentation standard is taken as sufficient justification for detailed documentation.

I'm going to run through two objections to detailed documentation. They are related, but one refers to design and the other to testing. It could be argued that both have their roots in psychology as much as IT.

I believe that the fixation of many projects on documentation, and the highly dubious assumption that quality and planning are synonymous with detailed documentation, have their roots in the structured methods that dominated software development for so long. These methods were built on the assumption that software development was an engineering discipline, rather than a creative process, and that greater quality and certainty in the development process could be achieved only through engineering-style rigour and structure.

Paul Ward, one of the leading developers of structured methods, wrote a series of articles [2] on the history of structured methods, which admitted that they were neither based on empirical research nor subjected to peer-review. Two other proponents of structured methods, Larry Constantine and Ed Yourdon, admitted that the early investigations were no more than informal "noon-hour" critiques" [3].

Fitzgerald, Russo and Stolterman gave a brief history of structured methods in their book "*Information Systems Development – Methods in Action*" [4] and concluded that "*the authors relied on intuition rather than real-world experience that the techniques would work*".

One of the main problem areas for structured methods was the leap from the requirements to the design. Fitzgerald et al wrote that "*the creation of hierarchical structure charts from data flow diagrams is poorly defined, thus causing the design to be loosely coupled to the results of the analysis. Coad & Yourdon [5] label this shift as a 'Grand Canyon' due to its fundamental discontinuity.*"

The solution to this discontinuity, according to the advocates of structured methods, was an avalanche of documentation to help analysts to crawl carefully from the current physical system, through the current logical system, to a future logical system, and finally a future physical system.

Not surprisingly, given the massive documentation overhead, and developers' propensity to pragmatically tailor and trim formal methods, this full process was seldom followed. What was actually done was more informal, intuitive, and opaque to outsiders.

An interesting strand of research was pursued by Human Computer Interface academics such as Curtis, Iscoe and Krasner [6], and Robbins, Hilbert and Redmiles [7]. They attempted to identify the mental processes followed by successful software designers when building designs. Their conclusion was that they did so using a high-speed, iterative process; repeatedly building, proving and refining mental simulations of how the system might work. Unsuccessful designers couldn't conceive working simulations, and fixed on designs whose effectiveness they couldn't test till they'd been built.

Curtis et al wrote; "*Exceptional designers were extremely familiar with the application domain. Their crucial contribution was their ability to map between the behavior required of the application system and the computational structures that implemented this behavior. In particular, they envisioned how the design would generate the system behavior customers expected, even under exceptional circumstances.*"

Robbins et al stressed the importance of iteration; "*The cognitive theory of reflection-in-action observes that designers of complex systems do not conceive a design fully-formed. Instead, they must construct a partial design, evaluate, reflect on, and revise it, until they are ready to extend it further.*"

The eminent US software pioneer Robert Glass discussed these studies in his book "Software Conflict 2.0" [8] and observed that "*people who are not very good at design ... tend to build representations of a design rather than models; they are then unable to perform simulation runs; and the result is they invent and are stuck with inadequate design solutions*".

These studies fatally undermine the argument that linear and documentation-driven processes are necessary for a quality product, and that more flexible, light-weight documentation approaches are irresponsible. Flexibility and intuition are vital to developers. Heavy-weight documentation can waste time and suffocate staff if used when there is no need.

Ironically, it was the heavy-weight approach that was founded on guesswork and intuition, and the light-weight approach that has sound conceptual underpinnings.

The lessons of the HCI academics have obvious implications for exploratory testing, which again is rooted in psychology as much as in IT. In particular, the finding by Curtis et al that "*exceptional designers were extremely familiar with the application domain*" takes us to the heart of exploratory testing.

What matters is not extensive documentation of test plans and scripts, but deep knowledge of the application. These need not be mutually exclusive, but on high-pressure, time-constrained projects it can be hard to do both.

Itkonen, Mäntylä and Lassenius conducted a fascinating experiment at the University of Helsinki in 2007 in which they tried to compare the effectiveness of exploratory testing and test case based testing. [9]

Their findings were that test case testing was no more effective in finding defects. The defects were a mixture of native defects in the application and defects seeded by the researchers. Defects were categorized according to the ease with which they could be found. Defects were also assigned to one of eight defect types (performance, usability etc.). Exploratory testing scored better for defects at all four levels of "ease of detection", and in 6 out of the 8 defect type categories. The differences were not considered statistically significant, but it is interesting that exploratory testing had the slight edge given that conventional wisdom for many years was that heavily documented scripting was essential for effective testing.

However, the really significant finding, which the researchers surprisingly did not make great play of, was that the exploratory testing results were achieved with 18% of the effort of the test case testing.

The exploratory testing required 1.5 hours per tester, and the test case testing required an average of 8.5 hours (7 hours preparation and 1.5 hours testing).

It is possible to criticize the methods of the researchers, particularly their use of students taking a course in software testing, rather than professionals experienced in applying the techniques they were using. However, exploratory testing has often been presumed to be suitable only for experienced testers, with scripted, test case based testing being more appropriate for the less experienced.

The methods followed by the Helsinki researchers might have been expected to bias the results in favour of test case testing. Therefore, the finding that exploratory testing is at least as effective as test case testing with a fraction of the effort should make proponents of heavily documented test planning pause to reconsider whether it is always appropriate.

Documentation per se does not produce quality. Quality is not necessarily dependent on documentation. Sometimes they can be in conflict.

Firstly, the emphasis on producing the documentation can be a major distraction for test managers. Most of their effort goes into producing, refining and updating plans that often bear little relation to reality. Meanwhile the team are working hard firming up detailed test cases based on an imperfect and possibly outdated understanding of the application. While the application is undergoing the early stages of testing, with consequent fixes and changes, detailed test plans for the later stages are being built on shifting sand.

You may think that is being too cynical and negative, and that testers will be able to produce useful test cases based on a correct understanding of the system as it is supposed to be delivered to the testing stage in question. However, even if that is so, the Helsinki study shows that this is not a necessary condition for effective testing. Further, if similar results can be achieved with less than 20% of the effort, how much more could be achieved if the testers were freed from the documentation drudgery in order to carry out more imaginative and proactive testing during the earlier stages of

June 7th - 8th, 2010

in Bad Homburg (near Frankfurt am Main, Germany)

Exhibitor / Sponsor

Apply as exhibitor

at www.testingexperience.com

or contact us via e-mail info@testingfinance.com

Be part of it

Exhibitors

PureTesting
Testing Thought Leadership

CaseMaker®

SELA GROUP

iSQI
International Software Quality Institute

Supporting Organisations

VOB SERVICE

Díaz Hilterscheid

te testing experience

development?

Susan Land's fourth justification for standards (see start of article) has interesting implications. Standards "*can be attached to or referenced by contracts*". That is certainly true. However, the danger of detailed templates in the form of a standard is that organizations tailor their development practices to the templates rather than the other way round. If the lawyers fasten onto the standard and write its content into the contract, then documentation can become an end and not just a means to an end.

Documentation becomes a "deliverable". The dreaded phrase "work product" is used, as if the documentation output is a product of similar value to the software. In truth, sometimes it is more valuable if the payments are staged under the terms of the contract, and dependent on the production of satisfactory documentation. I have seen triumphant announcements of "success" following approval of "work products" with the consequent release of payment to the supplier, when I have known the underlying project to be in a state of chaos.

Formal, traditional methods attempt to represent a highly complex, even chaotic, process in a defined, repeatable model. These methods often bear only vague similarities to what developers have to do to craft applications. The end product is usually poor quality, late and over budget. Any review of the development will find constant deviations from the mandated method.

The suppliers, and defenders, of the method can then breathe a sigh of relief. The sacred method was not followed. It was the team's fault. If only they'd done it by the book! The possibility that the developers' and testers' apparent sins were the only reason anything was produced at all is never considered.

What about the auditors?

Adopting standards like IEEE 829 without sufficient thought causes real problems. If the standard doesn't reflect what really has to be done to bring the project to a successful conclusion, then mandated tasks or documents may be ignored or skimped on, with the result that a subsequent review or audit reports on a failure to comply.

An alternative danger is that testers do comply when there is no need, and put too much effort into the wrong things. Often testers arrive late on the project. Sometimes the emphasis is on catching up with plans and documentation that are of dubious value, and are not an effective use of the limited resources and time. However, if the contract requires it, or if there is a fear of the consequences of an audit, then it could be rational to assign valuable staff to unproductive tasks.

Sadly, auditors are often portrayed as corporate bogey-men. It is assumed that they will conduct audits by following ticklists, with simplistic questions that require yes/no answers. "*Have you done x to y, yes or no*". If the auditees start answering "*No, but ...*" they would be cut off with "*So, it's no*".

I have seen that style of auditing. It is unprofessional and organizations that tolerate it have deeper problems than unskilled, poorly trained auditors. It is senior management that creates the environment in which the ticklist approach thrives. However, I don't believe it is common. Unfortunately people often assume that this style of auditing is the norm.

IT audit is an interesting example of a job that looks extremely easy at first sight, but is actually very difficult when you get into it. It is very easy for an inexperienced auditor to do what appears to be a decent job. At least it looks competent to everyone except experienced auditors and those who really understand the area under review.

If auditors are to add value, they have to be able to use their judgement, and that has to be based on their own skills and experience as well as formal standards. They have to be able to analyze a situation and evaluate whether the risks have been identified and whether the controls are appropriate to the level of risk. It is very difficult to find the right line, and you need good experienced auditors to do that. I believe that ideally IT auditors should come from an IT background so that they do understand what is going on; poachers turned gamekeepers if you like.

Too often testers assume that they know what auditors expect, and they do not speak directly to the auditors or check exactly what professional auditing consists of. They assume that auditors expect to see detailed documentation of every stage, without consideration of whether it truly adds value, promotes quality or helps to manage the risk.

Professional auditors take a constructive and pragmatic approach and can help testers. I want to help testers understand that. I used to find it frustrating when I worked as an IT auditor when I found that people had wasted time on unnecessary and unhelpful actions on the assumption that "*the auditors require it*".

Kanwal Mookhey, an IT auditor and founder of NII consulting, wrote an interesting article for the Internal Auditor magazine of May 2008 [10] about auditing IT project management. He described the checking that auditors should carry out at each stage of a project. He made no mention of the need to see documentation of detailed test plans and scripts, whereas he did emphasize the need for early testing.

Kanwal told me. "*I would agree that auditors are - or should be - more inclined to see comprehensive testing, rather than comprehensive test documentation. Documentation of test results is another matter of course. As an auditor, I would be more keen to know that a broad-based testing manual exists, and that for the system in question, key risks and controls identified during the design phase have been tested for. The test results would provide a higher degree of assurance than exhaustive test plans.*"

One of the most significant developments in the field of IT governance in the last few decades has been the US 2002 Sarbanes-Oxley Act, which imposed new standards of reporting, auditing and control for US companies. It has had massive worldwide influence, because it applies to the foreign subsidiaries of US companies, and foreign companies that are listed on the US stock exchanges.

The act attracted considerable criticism for the additional overheads it imposed on companies, duplicating existing controls and imposing new ones of dubious value. However, the response to Sarbanes-Oxley verged on the hysterical, with companies, and unfortunately some auditors, reading more into the legislation than a calmer reading could justify. The assumption was that every process and activity should be tied down and documented in great detail.

However, not even Sarbanes-Oxley, supposedly the sacred text of extreme documentation, requires detailed documentation of test plans or scripts. That may be how some people misinterpret the act. It is neither mandated by the act nor recommended in the guidance documents issued by the Institute of Internal Auditors [11] and the Information Systems Audit & Control Association [12].

If anyone tries to justify extensive documentation by telling you that "*the auditors will expect it*", call their bluff. Go and speak to the auditors. Explain that what you are doing is planned, responsible and will have sufficient documentation of the test results.

Documentation is never required "*for the auditors*". If it is required, it is because it is needed to manage the project, or it is a requirement of the project that has to be justified like any other requirement. That is certainly true of safety-critical applications, or applications related to pharmaceutical development and manufacture. It is not true in all cases.

IEEE 829 and other standards do have real value, but in my opinion their value is not as standards! They contain a wealth of good advice and the fruits of vast experience. However, they should be guidelines to help the inexperienced, and memory joggers for the more experienced.

I hope this article has made people think about whether mandatory standards are appropriate for software development and testing, and whether detailed documentation in the style of IEEE 829 is always needed. I hope that I have provided some arguments and evidence that will help testers persuade others of the need to give testers the freedom to leave the kindergarten and grow as professionals.

References

[1] Land, S. (2005). "Jumpstart CMM-CMMI Software Process Improvements - using IEEE software engineering standards". Wiley.

[2] Ward, P. (1991). "The evolution of structured analysis: Part 1 - the early years". American Programmer, vol 4, issue 11, 1991. pp4-16.

Ward, P. (1992). "The evolution of structured analysis: Part 2 - maturity and its problems". American Programmer, vol 5, issue 4, 1992. pp18-29.

Ward, P. (1992). "The evolution of structured analysis: Part 3 - spin offs, mergers and acquisitions". American Programmer, vol 5, issue 9, 1992. pp41-53.

[3] Yourdon, E., Constantine, L. (1977) "Structured Design". Yourdon Press, New York.

[4] Fitzgerald B., Russo N., Stolterman, E. (2002). "Information Systems Development – Methods in Action", McGraw Hill.

[5] Coad, P., Yourdon, E. (1991). "Object-Oriented Analysis". 2nd edition. Yourdon Press.

[6] Curtis, B., Iscoe, N., Krasner, H. (1988) "A field study of the software design process for large systems". Communications of the ACM, Volume 31, Issue 11 (November 1988), pp1268-1287. http://conway.isri.cmu.edu/~jdh/MethodsF06/res/readings/p1268-curtis.pdf

[7] Robbins, J., Hilbert, D., Redmiles, D. (1998). "Extending Design Environments to Software Architecture Design", Automated Software Engineering, Vol. 5, No. 3, July 1998, pp261-290. www.ics.uci.edu/~redmiles/publications/J003-RHR98.pdf

[8] Glass, R. (2006). "Software Conflict 2.0: The Art and Science of Software Engineering". Developer Dot Star Books.

[9] Itkonen, J., Mäntylä, M., Lassenius C., (2007) "Defect detection efficiency - test case based vs exploratory testing". First International Symposium on Empirical Software Engineering and Measurement. http://www.soberit.hut.fi/jitkonen/Publications/Itkonen_M%E4ntyl%E4_Lassenius_2007_ESEM.pdf

Itkonen, J. (2008). "Do test cases really matter? An experiment comparing test case based and exploratory testing". http://www.soberit.hut.fi/jitkonen/Publications/Juha_Itkonen_Licentiate_Thesis_2008.pdf

[10] Mookhey, K. (2008). "Auditing IT Project Management". Internal Auditor, May 2008, the Institute of Internal Auditors. http://www.theiia.org/intAuditor/itaudit/archives/2008/may/auditing-it-project-management/

[11] The Institute of Internal Auditors (2008). "Sarbanes-Oxley Section 404: A Guide for Management by Internal Controls Practitioners". www.theiia.org/download.cfm?file=31866

[12] Information Systems Audit and Control Association (2006). "IT Control Objectives for Sarbanes-Oxley 2nd Edition" http://www.isaca.org/Template.cfm?Section=Research2&CONTENTID=36549&TEMPLATE=/ContentManagement/ContentDisplay.cfm

Biography

James lives in Perth in Scotland . He is currently working as a consultant through his own company, Claro Testing Ltd (www.clarotesting.com).
James has 24 years commercial IT experience, mainly in financial services, with the General Accident insurance company and IBM (working with a range of blue-chip clients) throughout the UK and also in Finland.
His experience covers test management (the full life-cycle from setting the strategy, writing the test plans, supervising execution, through to implementation), information security management, project management, IT audit and systems analysis.
In 2007 he successfully completed an MSc in IT. His specialism was "Integrating usability testing with formal software testing models". He is particularly interested in the improvement of testing processes and how the quality of applications can be improved by incorporating usability engineering and testing techniques.
James is a Chartered IT Professional and a professional member of the British Computer Society (MBCS). He holds the ISEB Practitioner Certificate in Software Testing and is a member of the Usability Professionals Association.

Foto: © swondie - Fotolia.com

The time has come!

Together with the global e-exam provider Pearson VUE, iSQI enables its customers from now on to take their exams no longer merely paper-based, but computer-based in the test center around the corner at the time of their choice all over the world.

ISTQB® Certified Tester - FL (English & German) / ISTQB® Certified Tester - AL Test Manager, Test Analyst, Technical Test Analyst (English) / IREB® Certified Professional for Requirements Engineering (English) / ISEB® Intermediate Certificate in Software Testing (German)

www.isqi.org

CERTIFYING PEOPLE

iSQI®
International Software Quality Institute

Standards: to be used with common sense!

by Erik van Veenendaal

Probably, I'm one of those persons who doesn't like standards, because they tend to restrict me and provide me with a harness. As many of you, I have experienced that a quality document, e.g. a test plan, wasn't accepted by QA because it wasn't according to the standard. Who cares! As long as it does the job. Having co-developed the TMap methodology and written several TMap books – a de-facto testing standard in amongst others the Netherlands, I'm probably as much to blame as anyone else. However, the real question to ask ourselves here is"Who's to blame: the standard, or those who apply the standard?" Having worked in many industries with many different standards over the years, I probably have to state that it is down to those applying the standards to use them effectively. Of course, it makes a difference whether it's a mandatory external standard, e.g. FDA, or "just" a company-internal standard, but still …….

Unfortunately, there is no single software testing standard yet (ISO is in the process of developing one to be released in 2012). There are many standards that touch upon software testing, but many of these standards overlap and contain what appear to be contradictory requirements. Perhaps worse, there are large gaps in the coverage of software testing by standards, such as integration testing, where no useful standard exists at all. Where standards related to software and system testing do exist, I would like to point to my personal top 3 recommended standards, for both building confidence between supplier and consumer, and providing information on good practice to the new or even experienced tester.

IEEE 829 – Software and System Test Documentation standard

One of the most popular and well-known testing standards is IEEE 829. IEEE 829 is referenced in many testing book and lectured as part of the ISTQB certification scheme. The recently updated version from 2009 has many benefits over the "old" version of 1998. Amongst others, it now has a dedicated template for a master test plan and a dedicated template for a level test plan; also there are various instances of test reports provided. Of course, IEEE 829 should again not be used as a restrictive harness, but rather as a source of inspiration. In fact, there are three main ways you can use this standard effectively:

1. If you do not yet have templates in your organization or project for test documentation, do not start re-inventing the wheel! IEEE 829 can be a great source to use when defining your own customized test documentation standards.
2. If you do already have a set of templates in use, it can be used as a checklist. A review your own set of documentation standards against IEEE 829 can lead to some improvement ideas and is a verification of completeness of your own test documentation standards.
3. Many of us now work with third parties, e.g. outsourcing, and try to make agreements on the test processes to be used and documentation to be produced. However, what is a "good" test log or test plan? This question often leads to a never ending discussion. My recommendation would be to use the well-known and internationally accepted IEEE 829 as part of the test agreements and state that test documents to be delivered have to comply with IEEE 829. An easy, but quality way out of an on-going discussion.

IEEE 1028 – Reviews

When discussing reviews, I always wonder why we are not all practicing such an effective and relatively low-cost technique. Surveys show that only 50% of projects practice reviews and only 25% practice them in the way they are meant to be practiced. Yet the technique has been around since 1976 when Michael Fagan published his famous paper. One of the things that strikes me as very strange is that many companies define their own review processes. Most often they end up with at least 90% of what has already been defined by IEEE 1028, but somehow succeed in mixing up terms and renaming the review types etc. Confusing and a waste of effort to say the least. Probably only great for consultancy companies that provide the service. IEEE 1028 is a very straightforward standard that provides the processes for the different types of review that can be distinguished, e.g. inspection, walkthrough, technical review and management review. Let's not waste effort in defining what is already provided; to the best of my understanding the added value is NOT in the 10% difference. Let's focus on getting reviews implemented using IEEE 1028 as a common reference process framework.

BS7925-2 Software Component Testing

Although this standard was originally targeted towards component testing, it can be used by all testers whatever test level they work at. The standard provides a test process framework, especially for component testing, but much more important provides great detailed descriptions for both structure-based and specification-based test design techniques. Not only does it provide a description, in the annex it also provides detailed examples for all test design techniques which is really helpful. A highly usable and complete overview of test design techniques, that should be used when introducing or improving test design within your company or project. Unfortunately, this standard is not lectured as part of the ISTQB certification scheme, which is a mistake according to my opinion. Nevertheless, I strongly recommend you to download this standard and start using it. It's much better than most test design books and totally independent of any testing method or process.

Finally, I strongly recommend not re-inventing the wheel, but using the testing standards available. But please, let common sense prevail…… good luck!

Erik van Veenendaal is a leading international consultant and trainer, and recognized expert is the area of software testing and quality management. He is the director of Improve Quality Services BV. At EuroStar 1999, 2002 and 2005, he was awarded the best tutorial presentation. In 2007 he received the European Testing Excellence Award for his contribution to the testing profession over the years. He has been working as a test manager and consultant in software quality for almost 20 years.
He has written numerous papers and a number of books, including "The Testing Practitioner", "ISTQB Foundations of Software Testing" and "Testing according to TMap". Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, the vice-president of the International Software Testing Qualifications Board and the vice chair of the TMMi Foundation.

Standards – Curse or Blessing

by Anke Löwer

"Standard" - an often used (or misused) word that shouldn't be missing in the area of testing. What is the meaning of "standard"? Which (test) standards exist? What keeps us from applying the standards or from being compliant? Are they a *curse* or a *blessing*?

These questions originate from a questionnaire which was filled in by test managers, quality assurance managers and managers of centralized testing groups. Their results were compared with the practical experience of the author, which will be clarified in this article.[1]

Definition

According to the definition of the British Standards Institution, a standard is a "published document that contains a technical specification or other precise criteria designed to be used consistently as a rule, guideline, or definition".

In Wikipedia, *standard* is defined as "relatively uniform or standardized, widely approved and mostly applied (or at least this is the aim) way to produce or execute something which has asserted itself compared with other ways".[2]

Let's extract a few concepts from the above definitions and approach the question "*Curse or Blessing*":

Published standards

Let's first look at the term "published".

The Institute of Electrical and Electronics Engineers (IEEE) has more than 375,000 members in more than 160 countries and has published more than 1300 standards (31.12.2007); many of these belong to the area of testing and related areas:

- IEEE 730 – Standard for Software Quality Assurance Plan
- IEEE 828 – Standard for Software Configuration Management Plans
- IEEE 829 – Standard for Software Test Documentation
- IEEE 1008 – Standard for Software Unit Testing
- IEEE 1012 – Standard for Software Verification and Validation
- IEEE 1028 – Standard for Software Reviews
- IEEE 1044 – Standard Classification for Software Anomalies
- …

In addition, numerous standards exist which are published by other national and international organizations: For example ISO 9126 for the definition of software quality characteristics, or DIN EN ISO 9001, etc.

Presently through the ISO/IEC JTC1 / SC7 Workgroup 26, a new international standard ISO/IEC 29119 Software Testing is being developed, which will probably substitute some of the existing standards (IEEE 829 Test Documentation, IEEE 1008 Unit Testing, BS 7925-1 Vocabulary of Terms in Software Testing, BS 7925-2 Software Testing, Software Component Testing) and is planned to be be presented in 2012 as the "Final International Standard". The objective is to develop an international standard for testing software in all areas, from the analysis to design, development and maintenance of software products.

Missing overview

Obviously, there are enough publications of test standards that anyone can have access to. The difficulty here is getting an overview of the variety of standards (including those which are in development) and to retain this information. There is a distinct lack of a central "test standard portal" which would help to find different process-specific, domain-specific and documentation-related standards, etc. – well structured and across different organizations.

At least 67% of the interviewees gave this information.

Through the home pages of single standardization organizations it is possible to search for standards. This is a simple search if the organization (e.g., ISO, IEEE) and the number of the required standard is known. The more commonly used method to find standards is via suitable reference lists / literature lists in test-related books as well as in the syllabi of tester training schemes, e.g. ISTQB Certified Tester. Also, this is always just an extract of all relevant standards. An overview to all publications is not available.

We contribute to part of the missing overview over the variety of different standards:

A regular search is carried out only by 42% of the interviewees. Most of them indicated to find changes / updates in the area of the test standards only by chance. None of the interviewees were members of the publishing international organizations (e.g. IEEE).

Assertion or avoidance

Have published standards really "asserted" themselves according to the above definition?

Where products are concerned, standardizations in the production process have found their way in: Our worldwide credit card usage is only possible because the credit card manufacturers follow a standard with regard to the physical characteristic and the format of the cards.

Nevertheless, in spite of standardized products, the standard itself can also be avoided. The QWERTY standard describes the layout for English-language computer keyboards and typewriters. This name is derived from the first

1 In this case, the questionnaire carried out from June to July 2009 was a non representative questioning among customers.

2 As opposed to the norm, a standard does not go through several stages of official standardization procedure, for example by international organizations (ISO, IEC, ANSI etc.) as carried out for this purpose. This article does not distinguished between the terms standard and norm, and the term standard will be uniformly used throughout.

six letters in the top left letter row of the keyboard. The standard dates from 1874 and was developed during more than six years of trial and error lay to find the best arrangement of letters and figures. This arrangement had the aim to enable one to write as many words as possible with one single hand and to avoid the frequency of type lever collisions (on typewriters). From freely accessible shareware tools to 10-finger system learning facilities (costing only about 20 Euros), a lot of aid is available to use this standard. Henceforth, it cannot be the costs that keep enough PC users to still serve the keyboard with the 2-finger system. Why? It has been proven that the use of the 10-finger system does save time. But how many of the test managers, test analysts, test automation experts among us write a huge number of test documents with 2 fingers only?

Does the claim to individuality stand in the way of using standards?

Why do we in practice often see test plans which do not follow the table of contents according to IEEE 829? Why (from the author's experience) are attributes so often missing in incident reports pertaining to software versions and releases, even though IEEE 1044 lists all the attributes which should be contained in an incident report?

Does the time factor play a role here? Quite often, the development of standards and even more so the process of standardization can take years. Individual problem solving, i.e. the use of older standards, are used in the meantime. Time and benefit to adapt a new or revised standard must be weighed.

Are test standards "used consequently"?

Publication is a requirement for the – billable - use of standards. For example, the price for the IEEE 829 standard amounts to $110 for IEEE members, $138 for non-members. Two-thirds of the questionnaire participants consider this price too high. Only 25% of the participants stated that a large part of demanded standards are presently on hand either electronically or as hard copies. Does the cost factor contribute to this low percentage?

Approx. 58% of the participants indicated that they use IEEE 829 for software test documentation. Almost as many use the ISO 9126 standard. All other standards reached a low rate of utilization:

Besides, it is probably fair to assume that the participants who returned the questionnaire, belong to a group of "standard enthusiasts" who more than likely use much more standards than those who did not respond at all.

Benefits

"Standards help to make life simpler and to increase the reliability and the effectiveness of many goods and services we use. They are a summary of best practice and are created by bringing together the experience and expertise of all interested parties – the producers, sellers, buyers, users and regulators of a particular material, product, process or service." [BSI, FAQ 2009].

This statement is also true for test standards. Why are they then not always used strictly and consequently? Standards are designed for voluntary use. Is this a part of the problem?

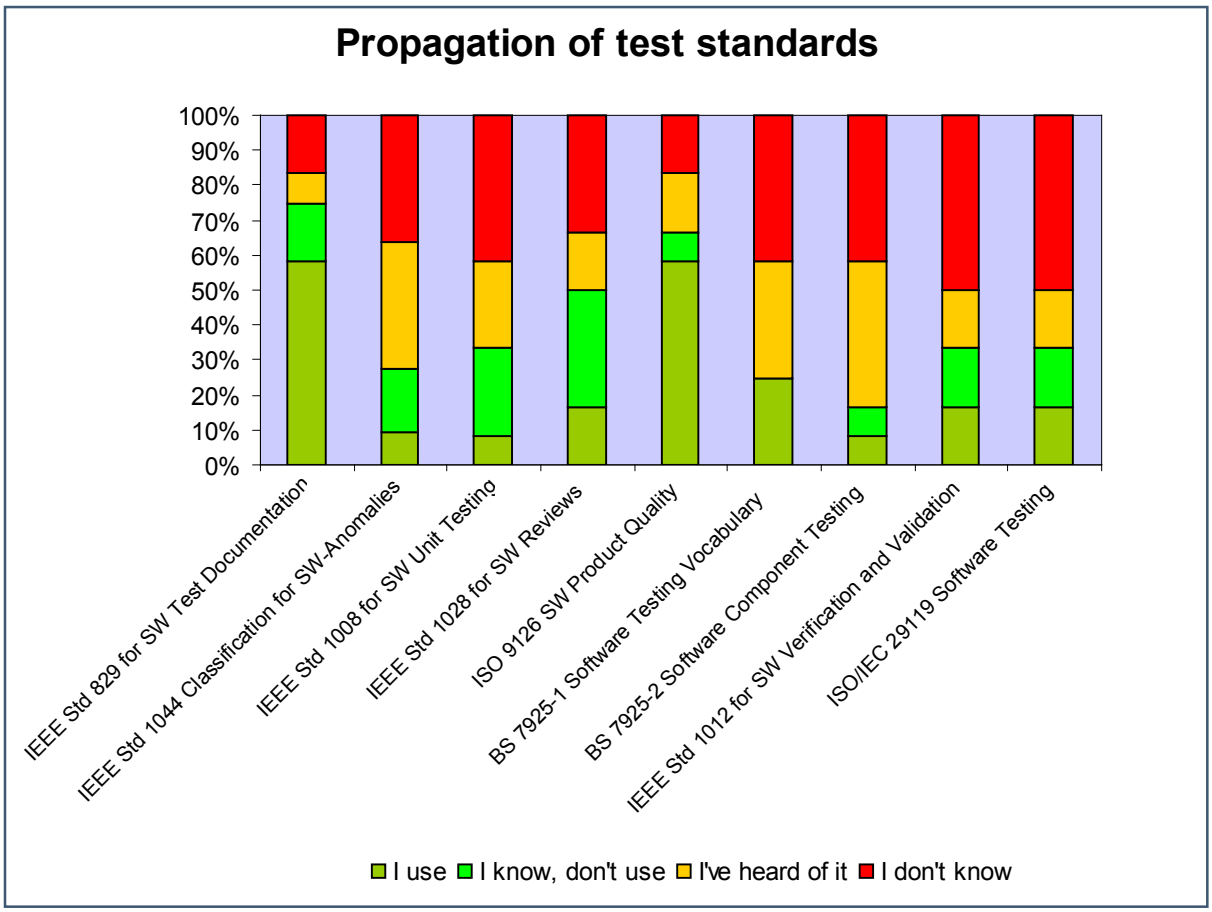
The advantages of standardization are obvious – at least when an enterprise reaches a certain size:

The cooperation of testers whether in a project team or on the corporate level is made easier on account of common understanding. At least 92% of the interviewees stated that this was the case *always* or *often* with the use of standards. In addition, standards can also benefit on a personal level: There is an increase in employment ads that require test managers and tester to be certified. This raises the quality of the personnel.

Standards are valuable for producing a standard glossary of test terms. All interviewees agreed that the training time of new colleagues is reduced *always* or *often* when using standards.

Standardization promotes synergies and allows benchmarking (67% of the interviewees also agreed). However, is this wanted?

Why don't external consultants introduce uniform guidelines with their customers, but instead create their own process models, templates, etc.? Is this a "man-made feature" which brings a competitive advantage to the table? One third of the interviewees expressed the wish that external enterprises should increase propagating test standards, instead of custom-made models.



Obstacles

This leads us directly to other possible obstacles when applying the use of standards: With standardization, compromises must be made and become a rule. Who really likes to compromise? Indeed, test managers and testers should be able of compromise in general …

As the case may be, one could get the impression that the introduction of a standard automatically generates expenditures, because formal and bureaucratic restrictions are to be followed. [Kaner, 2002] writes in addition: "If you follow a test documentation standard, such as IEEE 829, you will spend a lot of time and money documenting your test cases". Is this right? A standard does not express the level of detail and amount of content that must be contained in the test documentation. Longer-term advantages of standardization (lower training time, synergies through easy exchange / application of testers, creation of transparency and the conditions for economic efficiency, central supply of "help" within the framework of projects, etc.) must be confronted with short-term individual solutions on the basis of costs and benefits. The introduction of standards is not always a winner.

Criticism on the available standards is also passed by others [C. REID, 1995]:

"Many of these standards overlap and contain what appear to be contradictory requirements".

Or do some standards lack suitable tailoring?

One quarter of the interviewees pointed out that standards were partially non-applicable to operationalization, at least 67% were of the opinion that standards need *always* or *often* individual adaptation.

Test design techniques like equivalence class partitioning are explained in books or taught in seminars. Nevertheless, no author tells us how the test cases should be documented so that the method is supported in the best way possible. It would be beneficial for operationalization to have templates within the standards.

An additional interpretation of some standards might be necessary: Although 100% of the interviewees indicated that they have guidelines and templates for test plans in their company which follow the IEEE 829 standard, only half indicated that they have guidelines for test procedure specifications which are described in the same standard.

Conclusion

A vast number of reasons speak for the use of standards:

There are many test standards available in which a wide "knowledge base" of experience has flowed in. Standards have proven to make cooperation and the common understanding of projects and overlapping organizations easier. They allow synergies and benchmarking.

However, some things still prevent consistent utilization: costs, lack of central overview, recognition of proven standards over custom standards, partly missing concepts of operationalization integration.

Thus, the question of whether standards are a "*curse or a blessing?*" is decided after all as being a question of faith.

Are we ready to accept standards without making home-made experiences? Or do we see in them a curse which calls for exclusive compromises and a restriction of individuality?

I personally think that with a central "*test standard portal*" plus an even stronger spreading of the standards, for example through external consulting firms, with suitable tailoring possibilities and operationalization within the standards themselves, they will eventually be a blessing for testers and enterprises.

References:

[Kaner, 2002] Kaner, C. (2002) Requirements for Test Documentation

[BSI, FAQ 2009] http://www.standardsuk.com/FAQs.php

[C. REID, 1995] Reid, Stuart C. (1995) Software Testings Standards – do they know what they're talking about?

Biography

Anke Löwer is for more than 15 years well grounded in theory and practical experience regarding test and quality management. Her main focus is on test planning and control, as well as on risk-based test case design techniques using black and white box tests. The management of test teams, the definition of procedures, selection and introduction of tools for test management and quality assurance, configuration management, UML modeling and test automation were part of her work for BHF bank and Logica. At present she is working for corporate quality consulting as a senior test manager. She is conducting training courses within the training programme "ISTQB certified tester" for several years now.

Tutorial with Rex Black

As part of the Testing & Finance 2010 we are proud to be able to present a workshop guided by Rex Black.

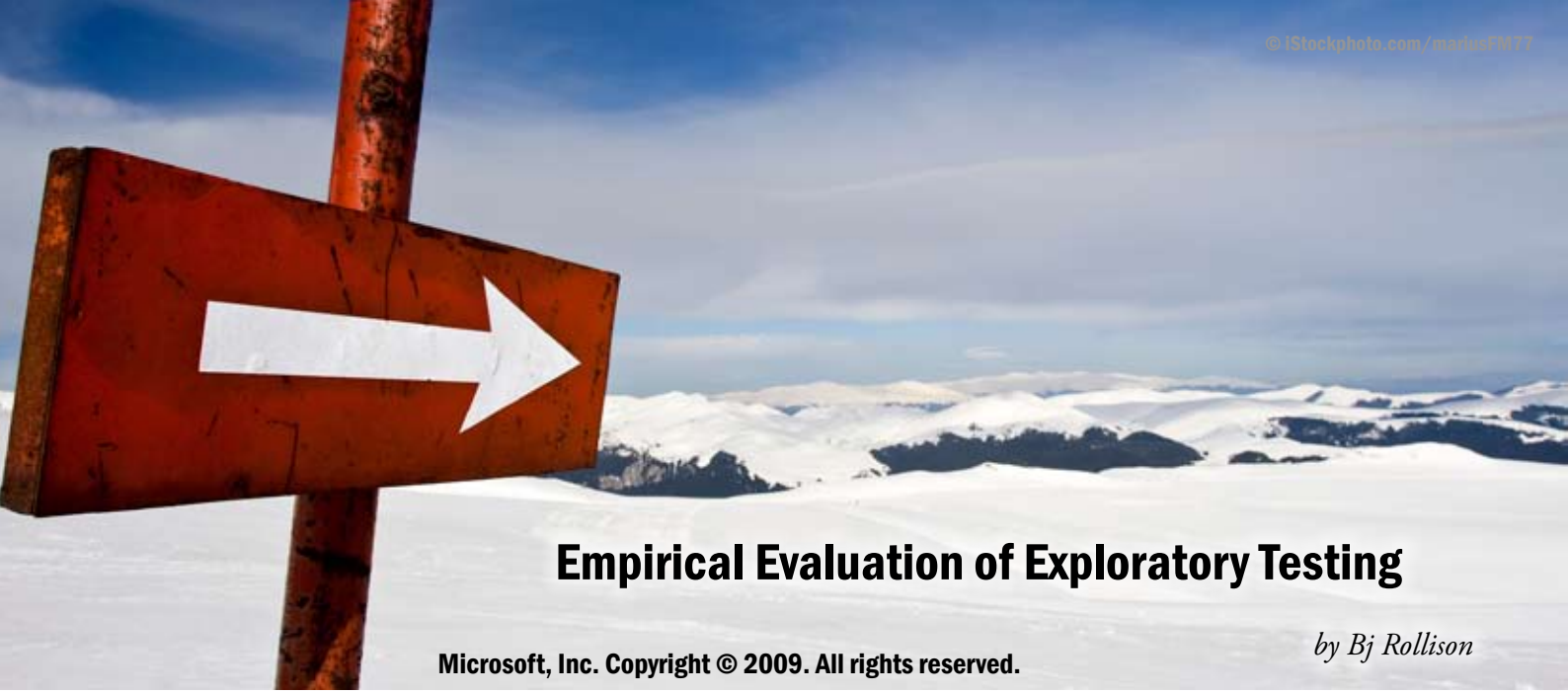
Theme:

Managing the Testing Process

The Workshop will take place from June 09 to 11, 2010

in Bad Homburg.

If you would like to receive further informations, please contact us via mail: info@testingfinance.com



Empirical Evaluation of Exploratory Testing

by Bj Rollison

My career in software testing probably started like some of you. I relied on domain knowledge and some intrinsic traits such as problem solving, critical thinking, curiosity about how things worked, and creativity to find bugs in software. I didn't have any formal training, and my only knowledge of software testing was gathered from glancing over a few chapters of a book, and some sporadic direction from my managers and leads. Learning how to test was mostly trial-by-fire, and testing focused on bug finding by designing a variety of negative tests on the fly, asynchronously executing those tests, gathering and evaluating the information, and learning more about the product and patterns of problems.

My professional career at Microsoft began in 1994 testing Windows 95; mostly focusing on the East Asian language versions. When I started at Microsoft I was already familiar with a few classic attacks using double-byte encoded character sets (DBCS), and those attacks exposed countless bugs. It was almost too easy! But then the developers learned to prevent these types of bugs and my attack patterns became less effective. I had learned more about the internal workings of the Windows operating system and I designed new tests, executed more tests, and found lots of pretty cool bugs. In fact, my bug counts were so high the VP of Windows rewarded me and other top performers with tickets to the Rolling Stones' Voodoo Lounge Tour at the Seattle Kingdome in December of that year.

Much of my early testing was based on exploratory testing approach, and it shouldn't be a surprise to anyone that exploratory testing is probably the single most common approach used in software testing. Perhaps this is because many practitioners in our field approach software testing as a purely destructive process, and exploratory testing can be a very effective approach in designing negative tests on-the-fly. Cem Kaner initially coined the term "exploratory testing" and the definition of exploratory testing has somewhat evolved over time. A commonly recognized definition suggested by James Bach is "*Simultaneous learning, test design, and test execution; **any testing to the extent that the tester actively controls the design of the tests as those tests are performed …***" So, whether we call it exploratory testing or not we all do it to some extent, and it may be applied differently because there is not one single way to engage in exploratory testing.

Questioning Exploratory Testing

When I really started studying practices in our profession I learned about Boris Beizer's first law - The Pesticide Paradox. The Pesticide Paradox is a metaphor that suggests no single approach to testing is effective in exposing all categories of defects, and that the effectiveness of a single approach to testing wears out over time. I also began reading books on software testing by experts such as Glenford Myers, Boris Beizer, and Robert Binder. These early pioneers of the software testing profession studied fault models and categories of defects, and then used heuristics and other problem solving approaches to develop patterns or techniques that are effective in identifying specific types of problems. I also learned that some approaches to testing are more effective than others in exposing different types of defects in software.

There are numerous papers and talks on the perceived benefits of exploratory testing as an approach to software testing. However, the benefits of exploratory testing are mostly based on emotional reaction rather than demonstrable evidence. I always questioned these claims; not because I don't appreciate the value of exploratory testing approaches, but because while this approach to testing has found a lot of important bugs it also missed many important bugs as well. Basically, I didn't buy into the idea that exploratory testing was a revolutionary approach to testing, and I wanted to study exploratory testing to learn more about it. I wanted to discover if exploratory testing was equally effective in exposing all categories of defects, or if it is more efficient in exposing a particular category of defects? Essentially, I wanted to better understand some of the benefits and limitations of exploratory testing based on empirical studies rather than anecdotal evidence so we could learn how to use it more effectively in our software testing process.

So, in 2002 I began a series of experiments in our new tester training program at Microsoft intended to help me answer 2 questions about exploratory testing:

1. Does a knowledge and practiced application of formal systematic testing techniques such as equivalence partition testing, combinatorial testing, and boundary testing increase the effectiveness of exploratory testing and test design?
2. Is there evidence to support the claim that "exploratory testing is orders of magnitude more effective than scripted testing?"

Interestingly enough, while conducting these experiments over the past 7 years I discovered others in the industry and academia who were conducting similar studies. Although the internal Microsoft study used reasonably simple software simulations, the external studies used publically released software. Surprisingly, these different studies carried out around the world on different types of products had very similar results and conclusions. And, the general findings of these studies are also nearly identical to the results reported in a case study involving 3 separate software companies. Of course, studies such as these rely on control groups and the results and the conclusions are based on the observations of these control groups. However, since these findings are very similar I think we are beginning to better understand the benefits and limitations of the exploratory testing approach.

The Impact of Knowledge

Marne Hutcheson conducted a study with a company who was developing a web store, and she shared her results at the Pacific Northwest Software Quality Conference (PNSQC) in 2007. The study was intended to compare the effectiveness of an exploratory testing approach to table-driven testing approach. The testers at the company tested a web client and data-base seeded with errors. Then, Marne taught the testers how to design tests using a table-driven test design approach. A comparative analysis of the results is illustrated in the charts in Figures 2.

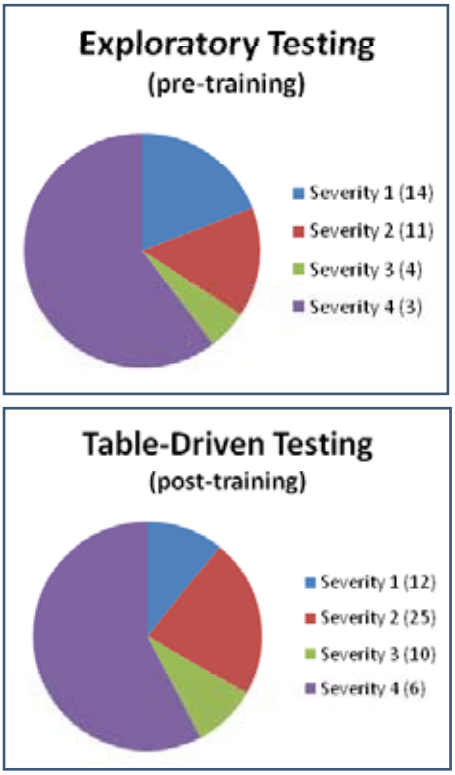


Figure 2

Key findings by Marne Hutcheson include:

- Only 1 of the 25 seeded bugs in the database was found by the untrained testers, and 21% of the untrained testers found 0 bugs during exploratory testing
- Testers found 15 of the 25 seeded bugs in the database after trained to use a table driven testing approach
- The table driven testing approach by trained testers found almost 60% more errors as compared to an exploratory testing approach by untrained testers.
- Exploratory testing by untrained testers found slightly more Severity 1 type problems in the client

Based on her study, Marne Hutcheson concluded:

- "Exploratory testing tends to trigger issues through spot negative testing."
- "Exploratory testing is not useful in systematically identifying or predicting the prevalence of a particular category of defect."
- Exploratory testing does not logically transform into automated tests as compared to table driven tests (but of course, that is expected.)"

Although I think Marne used her study to compare an exploratory testing approach to a more systematic table-driven approach to testing this web store, I looked at her study from a slightly different perspective. In our internal study at Microsoft, we were trying to evaluate the impact of training in testing effectiveness. When reviewing Hutcheson's case study there seemed to be a similar thread, and so instead of viewing her study as a comparison between exploratory testing approaches and table-driven test designs, we were both really comparing untrained testers who thought they were performing exploratory testing to the same group of testers who received some form of training in one or more formal systematic approaches to testing to see how it impacted their overall testing effectiveness.

This becomes clearer when we look at the first Microsoft case study. This experiment was based on Gerald Weinberg's Triangle problem as outlined by Glenford Myers in his book, The Art of Software Testing."

"A program takes three integer values representing the sides of a triangle, and the output will be either equilateral, scalene, or isosceles triangle."

The study participants were provided with this simple requirement, a simulation developed in C#, and instructions to outline the necessary tests and specific input values or actions to evaluate the programs' ability to satisfy the stated requirements.

The testers who participated in the Microsoft multi-year study ranged from less than 1 month to more than 6 years of experience in software testing. In the first 2 years of this study the average experience of tester's was slightly over 2 years, but in subsequent years the average experience has dropped to less than 6 months. Interestingly, our initial findings indicated the more experienced testers who lacked formal training in testing concepts tended to raise issues such as globalization, performance, security, compatibility, etc. during their initial testing. These areas are important; however, they were tangential to the specific testing goals outlined in the study. These results may indicate testers have a tendency to engage in context-free exploratory testing in hopes of finding a bug rather than focus on specific goals. It could also indicate that exploratory testing may lead testers to try to cover many different things in a limited amount of time rather than correctly prioritizing testing objectives. However, in general we found that experience as a tester without any prior formal training in software practices had no significant impact on their ability to adequately evaluate our implementation of the triangle program.

Following two consecutive days of training on functional testing techniques, each group of testers was given a brief but clear requirement statement, a simulation similar in size and complexity to the triangle program with the same instructions to evaluate the program's ability to satisfy the stated requirement. In each case we collected data on input variables, path analysis, and defect detection effectiveness. The charts below in Figure 1 illustrate overall effectiveness of testing pre and post training.

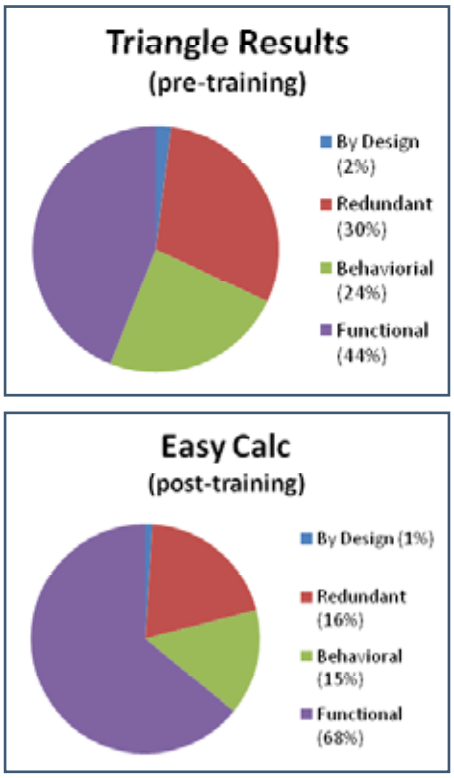


Figure 1

Key findings of this Microsoft study include:

- Slightly less than 45% of the tests designed by the untrained testers effectively evaluated the critical computational logic in the program
- Test effectiveness increased post training, but still failed to adequately evaluate all critical paths
- Less than a 10% probability of untrained testers identifying a critical seeded bug.
- There was more than a 65% probability of trained testers finding a critical seeded bug.
- Almost one quarter of tests executed by untrained testers tended to focus on user interface elements and the software behavior, and 96% of these tests focused on usability aspects.
- 80% of the untrained testers incorrectly misidentified specific boundary values

Based on the results of this study conducted over the past 7 years in several countries, I reached the following conclusions:

- Testers with no formal training in testing methodologies and approaches tend to be less effective as compared to trained testers regardless of experience in practice. (Practice doesn't make perfect; perfect practice makes perfect!)

- Untrained testers tend to report a greater number of false positives and also tend to execute tests that are out of context, or out of scope.
- There is no apparent correlation between the number or diversity of untrained testers and test effectiveness. (More doesn't mean better!)

I think that many of us can agree that training is important. Unfortunately studies show that very few testers in the industry have ever attended any formal training such as courses or conferences, and many testing practitioners have read less than 2 books specifically on the subject of software testing or software engineering. I believe that we all strive to be as effective as possible in our practice, and these studies may help illustrate one way to increase the potential effectiveness of exploratory testing or other testing approaches through more formal training. The bottom line as I see it is that testing effectiveness, regardless of approach used, is heavily dependent on our knowledge, skills, and experience.

Productivity and Exploratory Testing

In 2005, Juha Itkonen presented a case study of 3 separate companies comparing the effectiveness of exploratory testing to test case based testing (TCT), or 'scripted testing.' The results seemed to surprise him, and he spent the next few years studying the defect detection effectiveness of exploratory testing in comparison to 'scripted testing' or TCT as he refers to it in his research. His licentiate thesis at the Helsinki University of Technology presented additional findings that supported his earlier case studies. For his thesis work he used his graduate students to test 2 feature sets (FS) in an open source program called jEdit. Control groups created test cases based on the available documentation (TCT), and other control groups used an exploratory testing approach. The study found revealed no statistically significant difference between exploratory testing and TCT regarding defect detection effectiveness as illustrated in Figure 3.

Approach	Feature	Defects	Mean	s
ET	FS A	44	6.275	2.172
	FS B	41	7.821	2.522
TCT	FS A	43	5.359	2.288
	FS B	39	7.350	2.225

Figure 3. Overall defect detection effectiveness of ET versus TCT

What is more interesting about this case study is the types or categories of defects exposed by exploratory testing as compared to test case based testing. Figure 4 illustrates that exploratory testing was especially more effective in exposing behavioral and usability type issues and 'scripted testing' was more effective in exposing technical defects.

TYPE	ET	TCT	EE/ TCT (%)
Documentation	8	4	200%
GUI	70	49	143%
Inconsistency	5	3	167%
Missing function	98	96	102%
Performance	39	41	95%
Technical defect	54	66	82%
Usability	19	5	380%
Wrong behavior	263	239	110%

Figure 4 Defect by Category in comparing ET versus TCT

Key findings by Juha Itkonen include:

- Exploratory testing found 10% more behavioral issues
- Exploratory testing found 43% more GUI defects
- Exploratory testing found 380% more usability issues.
- Test case based testing found 22% more technical defects
- And, test case based testing found 4% more severe defects and 2% more critical defects

And based on his research, Juha Itkonen concluded that:

- "The data showed no benefit in terms of defect detection efficiency of using predesigned test cases in comparison to an exploratory testing approach."
- The data reveals "no significant differences between the two approaches in terms of the detected defect types, severities, or detection difficulty."

I think the data collected by J. Itkonen did reveal exploratory testing was more effective in exposing behavioral and usability issues as compared to test case based testing, and that is similar to observations we made in conducting our experiments when comparing the effectiveness of exploratory testing to test cases derived from a specification. However, it should be noted that in both studies, the 'testers' had received formal training in various software testing techniques and approaches and practiced using them to design test cases prior to these experiments. Again, this seems to indicate the effectiveness of both exploratory testing and 'scripted testing' is heavily dependent upon a tester's knowledge, skill, experience, and the perspective from which he approaches the problem.

In our internal study at Microsoft we also compared the effectiveness of 'scripted testing' to exploratory testing by evaluating condition coverage and defect detection effectiveness. In this experiment trained testers would design the set of 'pre-defined' test cases based only on the product requirements. It is interesting to note that these testers not only designed positive tests based on the product specification, but also designed negative test cases. Next, to reduce bias a different tester in the group executed the pre-defined set of 'scripted' test cases against an instrumented build of the product. In this study we tracked conditional code coverage measures, and also defect detection effectiveness. Then once all the pre-defined test cases were executed, the testers in the group were given 15 minutes to perform exploratory testing. Much to my surprise, the exploratory testing approaches used by trained testers had no significant impact on the code coverage measure. On average the code coverage metric only increased slightly (2 – 3%) when testers began using an exploratory testing approach after executing the predefined test cases. Even when we increased the time for exploration, the code coverage metric tended to remain flat. Next we modified the experiment to have one group start with exploratory testing, and another group design test cases based on the specifications and execute them. Again, the code coverage and defect detection effectiveness remained nearly identical as compared to the earlier experiments as illustrated in Figure 5.

Key findings of our study indicated:

- Exploratory testing had a 75% probability of exposing a bug in the menu structure; whereas 'scripted testing' had less than a 10% probability of exposing that defect
- Both exploratory testing and 'scripted testing' are most effective in exercising common code paths, but unlikely to exercise uncommon code paths.
- There is a 10% probability that a tester will achieve the optimal level of code coverage without performing code coverage analysis and using structural testing

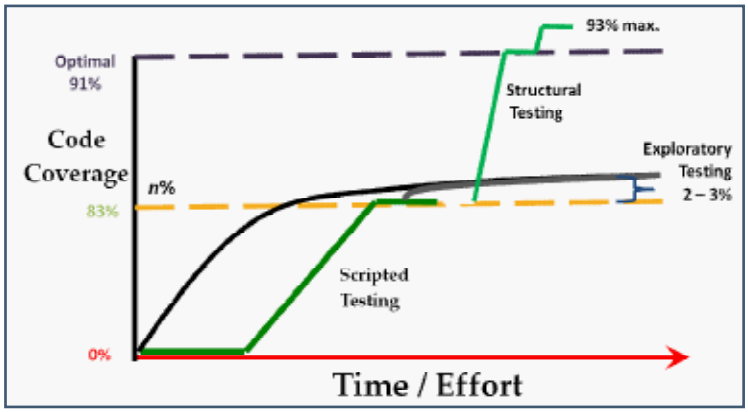
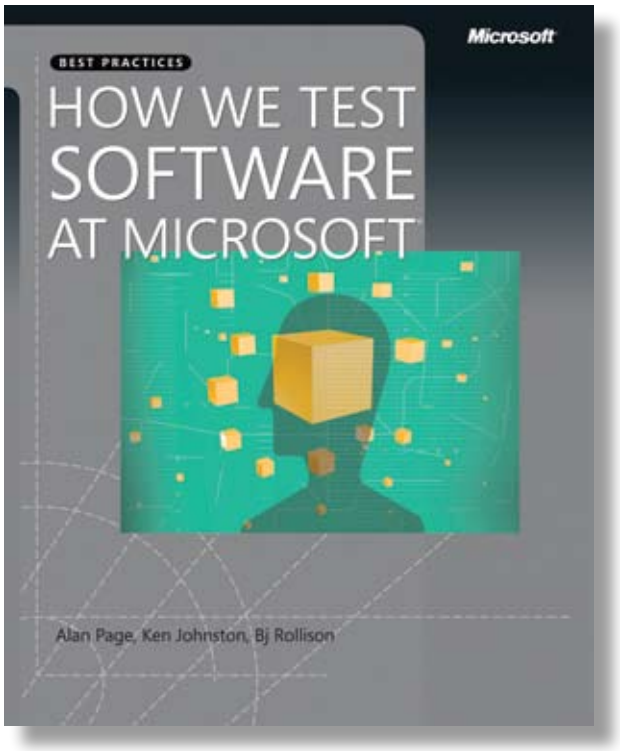


Figure 5. Code coverage effectiveness comparing scripted testing to exploratory testing

INTRODUCING

- Discover how Microsoft implements and manages the software-testing process company-wide
- Gain expert insights on effective testing techniques and methodologies
- Shares such facts as the number of test machines at Microsoft, how the company uses automated test cases, and bug statistics
- Answers key testing questions, such as who tests what, when, and with what tools
- Describes how test teams are organized, when and how testing gets automated, testing tools, and feedback



HOW WE TEST SOFTWARE AT MICROSOFT
ISBN: 9780735624252

ABOUT THE AUTHORS

Alan Page has been a software tester for more than 14 years, including more than 12 years at Microsoft Corporation.
Ken Johnston is group manager for testing in the Microsoft Office business group.
Bj Rollison is a Test Architect in the test excellence organization at Microsoft.
Authors' website: http://www.hwtsam.com

Microsoft® Press

Buy the book! **http://www.microsoft.com/learning/en/us/books/11240.aspx**

techniques. However, it is unlikely that tester will be able to repeat those tests and achieve the same level of coverage within the same period of time, and sometimes not at all.

Based on this study I concluded:

- There is no significant difference between 'exploratory testing' and 'scripted testing' in exercising common code paths.
- Exploratory testing is not an effective approach for increasing code coverage; although it may be an effective approach at increasing test coverage.
- Coverage analysis and white box structural testing the most effective approach to evaluate untested areas of the program after other (black box) approaches have been exhausted if testing those areas to reduce overall risk is necessary or important.

Our findings are not really surprising since the simulation we used for this experiment was a simple game with a single window with only 4 possible input events, and only had 1 critical seeded defect. We are introducing a more complex simulation with multiple forms and a variety of input variables to confirm this correlation of coverage and defect detection effectiveness occurs with more complex and more realistic applications. I suspect that with more a complex application, that we might see several incremental increases in code coverage and defect detection effectiveness by introducing mini bug bashes (a bug bash at Microsoft is typically a ½ day or daylong event where everyone on the product team focuses on one aspect of quality (globalization, usability, etc) and tests the product from an exploratory testing approach). Also, it is important to note that we are not using code coverage to measure quality or the effectiveness of our test coverage, but simply to evaluate the ability of tests to exercise different paths through the code in order to help us identify untested portions of the software. Depending on the risk and criticality of the product, code coverage analysis and white box structural testing may not be important or necessary for all product lines.

Benefits and Limitations

Exploratory testing has several benefits, and it also has its limitations. Just like any other approach to testing exploratory testing can be quite effective at finding many types of issues, but it can also miss other defects. Remember, no single approach to testing is effective in exposing all types of defects. So, let's look at some of the benefits of the exploratory testing approach.

Exploratory testing is useful in providing a rapid overall assessment of new or updated software. Many testers often rely on an exploratory testing approach as a way to get a general "feel" for the stability or 'buggy-ness" of new builds or changes in the product without much overhead. Even when we do component level testing on new fixes, we run a series of regression tests, and also do some amount of exploratory testing of the changed code to help us cover potential holes in our test cases.

Exploratory testing is an effective approach at exposing various categories of defects, but it seems most effective in helping us detect behavioral and usability type issues. This is not really surprising since many practitioners tend to rely on an exploratory testing approach while interacting with the software during runtime through the user interface. Although we also do exploratory testing below the UI layer during component and API testing, we currently do not have any defect detection effectiveness data when exploratory testing is used at this level of testing. However, my intuition suggests that it is helpful simply because it can provide an alternative perspective on the problem.

Exploratory testing does not necessarily require in-depth domain or system level knowledge, but a greater understanding of the system under test and various testing practices can potentially increase our effectiveness in how we approach exploratory testing. Exploratory testing is rooted in negative testing in order to expose bugs. Clicking on every pixel on a button control, or setting a brick on the keyboard doesn't take an incredible amount of ingenuity or creativity. But effective exploratory testing does require curiosity about how things work, creativity, problem solving, and the ability to approach a problem from multiple perspectives. Understanding the underlying system we are testing, and understand other testing approaches and tools will likely improve the tests we design on-the-fly while we are testing.

Of course, exploratory testing approaches should not be the only approach to software testing. In our case studies and those of others, as well as observations in practical application we have learned that exploratory testing may not be the best approach in all circumstances, and that other approaches to testing are more effective in exposing certain types of bugs more efficiently. Some limitations of exploratory testing in practice that we have discovered include:

Exploratory testing does not scale well as product size or complexity increases, or the scope of testing increases. We learned this lesson after James Bach was contracted to help develop an exploratory testing strategy for our application compatibility testing efforts for the Windows 2000 operating system. Although this effort seemed to add value initially, this approach was abandoned at Microsoft after Windows 2000 for 2 primary reasons. First, as the number of applications and hardware devices increased there was no efficient way to scope the testing effort without hiring more and more testers. When the matrix of applications and/or hardware being tested for compatibility is relatively small an exploratory testing approach may work well. However, the Windows 7 team tested over 750,000 applications and hardware devices in their compatibility testing, an effort of this magnitude is clearly not feasible without an army of testers, or without an effective automation strategy.

Another reason Microsoft stopped using exploratory testing as a primary approach for testing in the Windows application compatibility team was because of numerous false positives. A survey of the issues revealed that a significant portion of the reported bugs were actually defects in the application, and not compatibility problems caused by an adverse interaction between the 3rd party application and new features or changes in the Windows operating system. Perhaps this is due to the primary focus of exploratory testing to expose bugs through negative testing, or testers feeling that it more important to find any type of bug rather than find context specific bugs. Microsoft continues to use exploratory testing approaches; however, we balance it with other approaches as well in the appropriate situations.

Exploratory testing may be less effective for critical functional computational logic or underlying engineering quality issues. Based on these cases studies and observations exploratory testing is quite effective in exposing behavioral and usability problems. Exploratory testing appears to be less effective in finding some functional (computational logic) problems; especially when performed by untrained or undertrained testers. It is also not the most efficient approach to increase coverage of under-tested areas of the product. Again, this is not to suggest that exploratory testing approaches are not used in our component or API levels of testing, but we use it to reinforce more systematic approaches.

Exploratory testing is not especially effective for requirements traceability or repetition. This is perhaps why certain types of software such as medical, avionic, or banking software don't rely on exploratory testing as a primary testing approach. I seriously doubt many of us would get on an airplane if the only testing on the avionic systems was exploratory testing through the user interfaces in the cockpit controls. Of course, not all products mandate traceability between the requirements and the test cases. But for products that must adhere to specific contract requirements or governmental control then traceability is especially important. Also, during a long product development lifecycle regression testing is an important approach to ensure previously working functionality does not regress to a non-working state because of changes in the underlying code base as new features are added and bugs are fixed. It is sometimes difficult to repeat tests unless detailed notes are kept during the testing process.

Summing Up Exploratory Testing Approaches

There is no single right way to use exploratory testing approaches. Exploratory testing approaches will certainly vary depending on the context or the specific situation, and the business needs. Also, essentially all testing involves some amount of exploration while designing test cases, and the process of explor-

atory testing described as "simultaneous learning, test design, and test execution" is an important aspect of any comprehensive testing strategy. However, similar to other testing approaches, it may also instill a false sense of confidence especially if the success of a testing organization is primarily based on relatively simple measures such as number of bugs found or some period of time spent testing a feature.

Exploratory testing is very useful for probing assessments, or for providing a rapid overall evaluation of a product. It is also useful for small scale products where requirements traceability is not especially important, or for products with a limited shelf-life. But exploratory testing does not scale very well as the feature set or product complexity increases, or the scope of the testing effort grows. We also need to stay aware of the Pesticide Paradox and use a variety of testing approaches and methods throughout the software life cycle for maximum effectiveness.

Finally, the overall effectiveness of exploratory testing and other testing approaches depends heavily on the tester's knowledge, skill, and experience. We work in a very dynamic industry and we should strive to learn more about the systems we test, and the various practices in our profession to increase our skills to be more efficient and effective, and to broaden our experiences on new technologies as they emerge.

Bibliography

The Venerable Triangle Redux. Bj Rollison, STARWEST 2005

Do Test Cases Really Matter? An Experiment Comparing Test Case Based and Exploratory Testing. Juha Itkonen, Licentiate Thesis, Helsinki Univ. of Technology, 2008

Exploratory Testing: A Multiple Case Study, Juha Itkonen, Proceedings of the International Symposium Empirical Software Engineering, 2005

Exploratory Testing Versus Requirements Based Testing: A Comparative Study Marnie Hutcheson, PSQT Conference 2007

Exploratory Testing: A Workshop in Risk-Based Testing, Stale Amland 2002

Exploratory Testing, Erik van Veenendaal, Improve Quality Services BV, 2002

Biography

Bj Rollison is a Principal Test Architect with Microsoft's Engineering Excellence group where he designs, develops, and teaches technical training to Microsoft's test and development engineers. Bj got his first computer in 1979 and taught himself Q-Basic. He started his professional career at a small OEM company in Japan in 1991 building custom hardware solutions for small businesses. In 1994 he joined Microsoft and worked on several key projects including Windows 95 and Internet Explorer, and also served as the Director of Test responsible for managing the training programs for more than 9000 testers. In 2003 Bj decided his passion was teaching and mentoring testers and became a Test Architect in Microsoft's Engineering Excellence group. Bj also teaches software testing and test automation courses at the University of Washington, sits on the advisory boards at the University of Washington, the University of California Extension Santa Cruz, and Lake Washington Technical College. Bj is a frequent speaker at international conferences, and is co-author of "How We Test At Microsoft."

TPI® NEXT: Test Process Improvement improved!

by Bert Linker & Ben Visser

Since its launch in 1998, Sogeti's Test Process Improvement® has offered logical and practical steps to enhance test efficiency and effectiveness, and moreover, to instil a belief in achieving a permanent improvement cycle.

But, as in most things in life, there is always room for improvement. In over a decade of use, our customers, colleagues and Sogeti's own testing consultants have gathered a huge amount of experience and best practices in applying TPI 'in the field'. This experience, coupled with a changing IT environment and technology developments, has been the driver for a completely updated version of the model.

Developed by Sogeti TPI experts from several countries, with support from other colleagues and the active involvement of customers worldwide, the result is TPI® NEXT: Test Process Improvement has itself been improved. It is a powerful next step in the evolution of test process improvement, as it takes as its starting point the close alignment with business drivers, while leveraging the strengths of the original model.

The TPI® NEXT model – an overview

The TPI NEXT model is used to analyze the current situation of a test process, showing its relative strengths and weaknesses; the model also provides a roadmap for reaching specific business, IT or test goals.

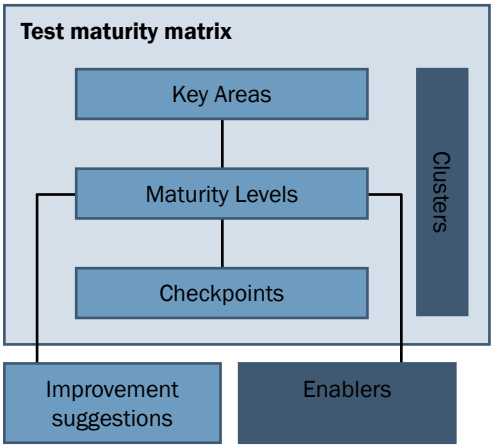


Figure 1. The key elements of the Business Driven TPI model

We kept the strengths

Clarifying the Key Areas

Each test process can be divided into a combination of coherent aspects called Key Areas. Integral to the original model, we have made some changes to the Key Areas - TPI NEXT identifies 16 – and the most important changes are explained here:

- We combined 'Life-cycle model' and 'Test process management', since they showed a great deal of similarity;
- The goal of TPI NEXT is to achieve a certain result from optimizing a test process. The scope of the test process improvement may cover all or any of the evaluation and test activities within the Software Development Life Cycle (SDLC). Because the scope or focus of TPI NEXT cannot be a Key Area as well, the Key Areas 'Evaluation' and 'Low-level testing' have been removed;
- The original Key Area 'Commitment and motivation' addressed two distinct success criteria for any test process. We believed commitment of stakeholders to be so essential for the success of test projects that we created a completely the new Key Area 'Stakeholder commitment'. Motivation has been integrated within the Key Area 'Tester professionalism';
- We found that the position of testing within an organization and the way test resources, products and services are provided for projects have changed. We have addressed these issues as a new separate Key Area 'Test organization'.

In more detail: a retrospective view of the original model

The original TPI® model looked at the test process from different points of view, for example the use of test automation, test specification techniques and reporting. These are called Key Areas. Examination of each Key Area leads to a classification of the test process at certain Levels of Maturity. The ascending levels improve in terms of time (faster), money (cheaper) and/or quality (better). For example, for the Key Area 'Reporting' defined Levels A through to D.

Key Areas and Levels are not equally important for the performance of the complete test process, and dependencies exist between the different Key Areas and Levels. Therefore they are all mutually linked in a Test Maturity Matrix.

To ensure that the classification of Levels is done objectively, Checkpoints (being requirements) are assigned to each Level. If a test process passes all the checkpoints at a certain level, then the process is classified at that level.

In addition to mapping the current situation of the test process, Key Areas and Levels can also be used to define the required situation and intermediate steps to get there. Improvement suggestions provide additional support and guidance to reach a certain level.

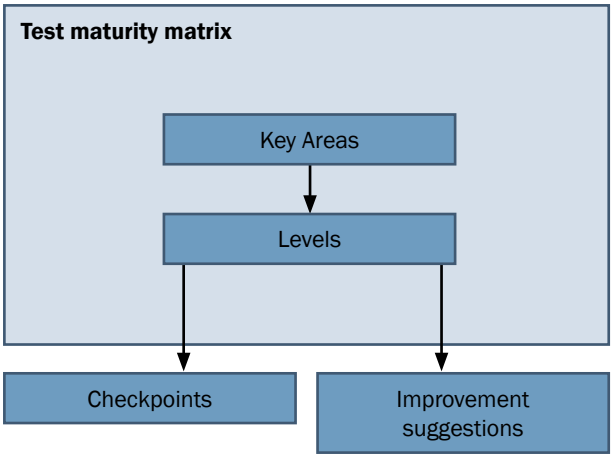


Figure 2. The elements of the original TPI model

Maturity levels have been redefined

As each Key Area can have a different level of maturity, it is the combination of the Key Areas which defines the maturity of the test process as a whole. TPI NEXT has 4 Maturity Levels for each Key Area, characterizing the test maturity:

- Initial: *Ad hoc* activities
- Controlled: Doing the right things
- Efficient: Doing things the right way
- Optimizing: Continuously adapting to ever-changing circumstances.

Compared with the original model, visualizing the maturity of the test process is now much clearer. With the original model, the levels (A to D) could be spread across the levels Controlled, Efficient and Optimizing. In TPI NEXT, all Key Areas comprise the 4 Maturity Levels.

Example: Statements per maturity level for Key Area 'Test process management'

Managing the test process maximizes the execution of the test assignment within the required time, costs and results.

- Controlled: Proactive management of the test process enables the fulfilment of the test assignment
- Efficient: Managing the test process with clear authorizations makes instant adjustments possible, to keep the test project on track
- Optimizing: Lessons learned on test process management advance the effectiveness and efficiency of steering test projects to their required end result.

All Checkpoints have been reassessed

Expectations for each of the Key Areas are defined by Checkpoints for each maturity level (except for the 'Initial' level). A Checkpoint is an objective statement that can be confirmed by a simple 'yes' or 'no'. TPI NEXT contains 157 Checkpoints across the 16 Key Areas and 3 main Maturity Levels.

Example: A Checkpoint for Key Area 'Test process management' at the Controlled level

At the start of the test project, a test plan is created. The test plan includes at least the test assignment, the test scope, the test planning, roles and responsibilities.

Improvement suggestions have been updated

For each Maturity Level per Key Area, we have provided Improvement suggestions, which describe hints and tips for reaching a specific Maturity Level. Checkpoints that have not been met should definitely be looked at to reach the Maturity Level, but while Checkpoints describe what needs to be reached, it is Improvement suggestions that describe how this can be done.

Example: An Improvement suggestion for Key Area 'Test process management' at the Controlled level

Search for appropriate test activities in available test methods in order to break them down over the separate phases.

The Maturity Matrix has been improved

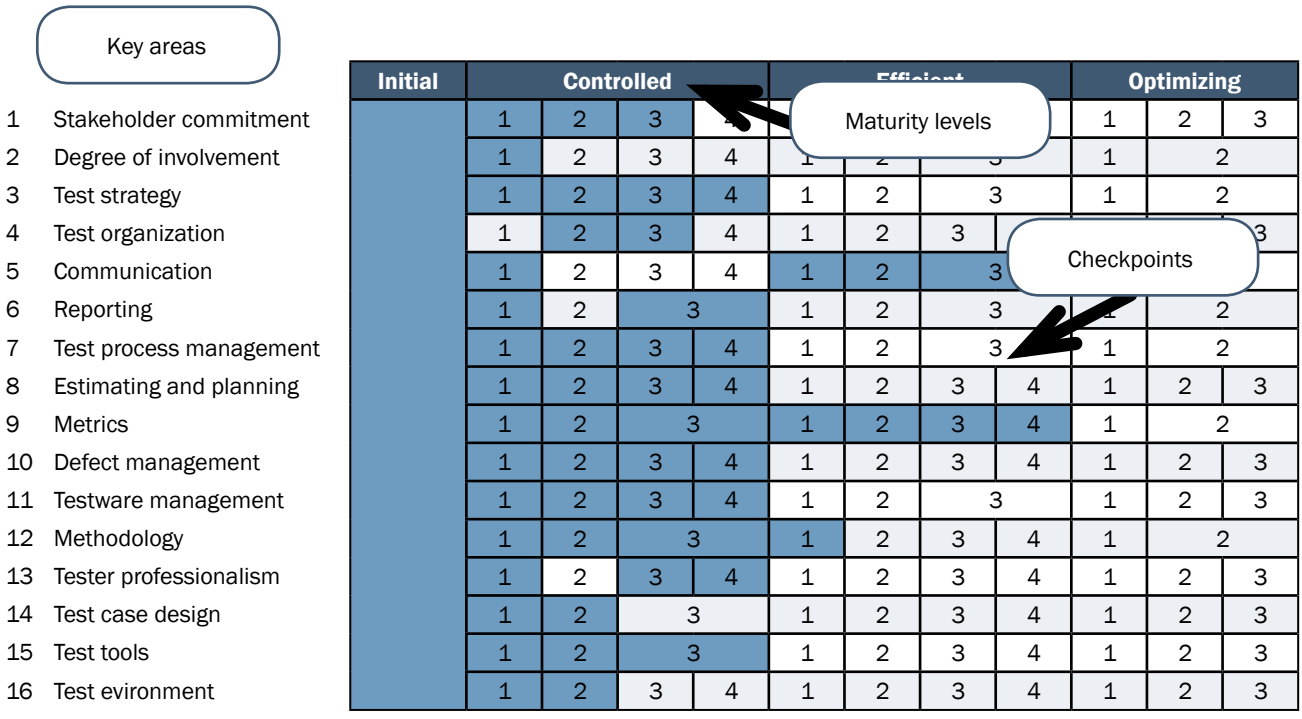


Figure 3. An example of a 'current' situation displayed in the Test Maturity Matrix

The Test Maturity Matrix provides a clear overview of all Key Areas, together with their respective maturities. The Matrix lists all 16 Key Areas in the horizontal rows, and the Maturity Levels (Initial, Controlled, Efficient and Optimizing) in the columns. The Checkpoints for each Key Area are displayed in order. Figure 3 shows that the Checkpoints in the matrix are not visualized to a standard 'width'. Please note that this does not indicate relative complexity or 'size' of that Checkpoint.

We added new features

- Enablers

Our testing experience has led us to realize more than ever that a test process cannot be seen as a stand-alone process, but is very closely related to other processes within the SDLC. In TPI NEXT we have introduced Enablers, which show where the test process and other processes within the software development life cycle can benefit from their respective best practices. An Enabler can both strengthen and accelerate the improvement process for testing and also stimulate the other SDLC processes to adopt similar measures to improve their maturity.

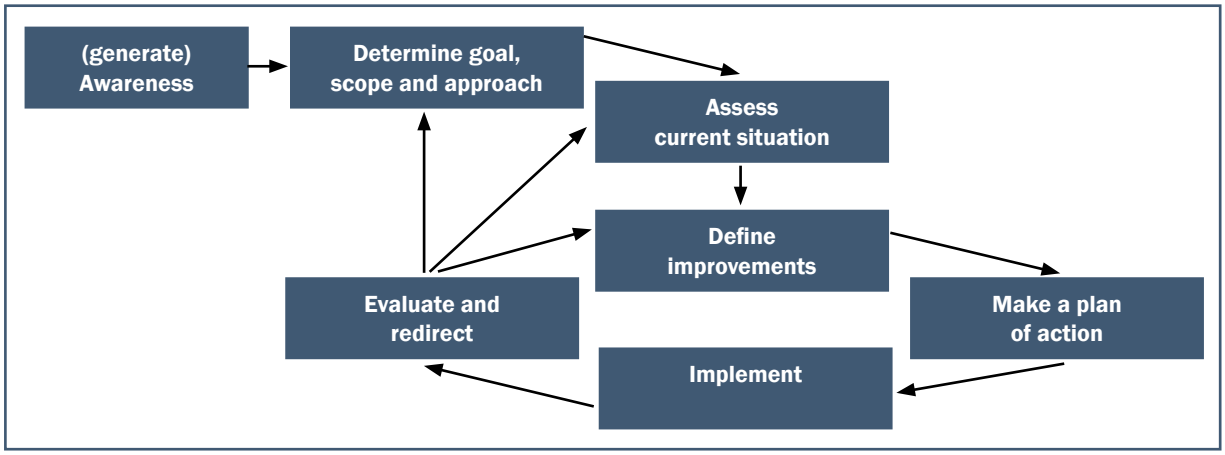
- Clusters

One of the strong points of the original model was the possibility of step-by-step improvement; the steps consisted of reaching a Maturity Level for specific Key Areas. The new TPI NEXT model also enables a stepwise improvement, from Initial, Controlled or Efficient levels through to fully Optimizing. However, the main difference is that now each step is indicated by Clusters of Checkpoints. We defined a Cluster as a group of Checkpoints from multiple Key Areas that may contain mutual relationships and that function as one improvement step.

TPI NEXT can be both process-driven and business-driven

To achieve a higher level of test process maturity, it is obviously necessary to implement improvement measures that not only depend on understanding the current state of maturity, but also on the improvement goals to be reached. To support the improvement process, the business-driven TPI model provides a step-by-step approach, in which each step is described in a Cluster as described above. The composition of the Clusters can vary according to the range of improvement goals.

Depending on which specific business driver is selected (or in some situations, which improvement drivers from within the test process itself), the Key Areas will be rearranged into different categories of priority. This will result in different improvement measures and activities.



Process-driven TPI NEXT: improving the full breadth of testing

For straightforward improvement, we have created a base clustering, illustrated in Figure 4. The improvement of the test process is regarded as a general improvement process, in which no one single business driver is predominant. To try and reach the Controlled level in one step would be, in our experience, usually a leap too far!

What is more, to make any improvement process successful, you need to celebrate successes on a regular basis. Although there's no one way to accomplish this, some ways are better than others, and we provide proven suggestions to help. It all starts with applying some generally acknowledged best test practices: think before you start, involve the appropriate people in this thinking, and monitor the actions agreed. Checkpoints addressing these first basic best practices together make up Cluster A.

Next, pay more and more attention to the details involved in managing the test process and the actual test activities. The Checkpoints for this next step are grouped together in Cluster B. Clusters C, D and E rep-

		Initial	Controlled				Efficient				Optimizing			
1	Stakeholder commitment		A	B	B	C	F	H	H		K	M	M	
2	Degree of involvement		A	B	C	E	H	H	J		L	L		
3	Test strategy		A	A	B	E	F	F	H		K	L		
4	Test organization		A	D	D	E	I	I	J	J	K	L	L	
5	Communication		B	C	C	D	F	F	J		M	M		
6	Reporting		A	C	C		F	G	G		K	K		
7	Test process management		A	A	B	B	G	H	J		K	M		
8	Estimating and planning		B	B	C	C	G	H	I	I	K	L	L	
9	Metrics		C	C	D		G	H	H	I	K	K		
10	Defect management		A	A	B	D	F	F	H	J	K	L	L	
11	Testware management		B	B	D	E	I	I	J		L	L	L	
12	Methodology		C	D	E		F	H	J	J	M	M		
13	Tester professionalism		D	D	E	E	G	G	I	I	K	K	M	
14	Test case design		A	A	E		F	I	I	J	K	K	M	
15	Test tools		E	E	E		F	G	G	I	L	M	M	
16	Test evironment		C	D	D	E	G	H	J	J	L	M	M	

Figure 4. The view of base Clusters in the Test Maturity Matrix (with Cluster A highlighted)

Subscribe for the printed issue!

Please fax this form to +49 (0)30 74 76 28 99, send an e-mail to info@testingexperience.com
or subscribe at www.testingexperience-shop.com:

Billing Adress

Company: _____
VAT ID: _____
First Name: _____
Last Name: _____
Street: _____
Post Code: _____
City, State: _____
Country: _____
Phone/Fax: _____
E-mail: _____

Delivery Address (if differs from the one above)

Company: _____
First Name: _____
Last Name: _____
Street: _____
Post Code: _____
City, State: _____
Country: _____
Phone/Fax: _____
E-mail: _____
Remarks: _____

1 year subscription
☐ **32,- €**
(plus VAT & shipping)

2 years subscription
☐ **60,- €**
(plus VAT & shipping)

Date

Signature, Company Stamp

resent further consecutive steps, until you can truly say: "We're doing the right things" – i.e. the test process can be categorized as Controlled. Likewise, moving from Controlled to Efficient Maturity Levels, and from Efficient to Optimizing, can be achieved through relatively small steps, as shown in Figure 4.

In more detail: backward compatibility

Over the past decade, many organizations have used the original TPI model. Although the business-driven TPI model is different in some aspects, it is still possible to use the results of former (original) TPI assessments and improvement processes together with the new business-driven TPI.
If the base information from an original TPI assessment is still available, you can use this data to produce a new business-driven TPI matrix. Even if only the original TPI Maturity matrix is available, you can use a conversion table that translates the value of original TPI Checkpoints to the value of the new associated business-driven TPI Checkpoints. A tool to enable this conversion is provided through the TPI NEXT website (www.tpinext.com).

Business-driven TPI NEXT: improvement with a specific goal in mind

Not all test improvement initiatives are directed at achieving better test processes in general. Probably most address a specific business or IT objective. To accomplish such a specific goal, following the base clustering might not be the best way to proceed. In this situation, some Checkpoints can be safely put on hold, while others are best realized at an earlier stage. In other words, the content of the Clusters should be adjusted to reflect the specific needs of the desired improvement.

To create Clusters for a specific business driver, a 6-step approach is provided. The first task is to define which Key Areas are most relevant to the required business driver focus (e.g. cost, time, quality as business drivers). You then need to determine how this influences the distribution of the Checkpoints across the Clusters. The 6 steps are described below:

1. Identify the business driver

An important part of this step is to describe how to measure progress against the driver. If the business driver is well formulated, it can serve as the basis for reporting. It can also prevent future disappointments, since discussions on how to measure often reveal implicit expectations by the business.

In more detail: Business drivers

A business driver is a management directive, usually directly derived from the organization's vision and/or business strategy, which wants specific outcomes of the organization at an operational level. A business driver is a reason, motivator or challenge for test process improvement, commonly indicated as (a combination of) cost, time, quality and risk.

2. Translate the business driver into an IT goal

This is especially helpful since test process improvement is not always initiated within a business department. Often the IT department recognizes the need for improvement and acts accordingly. This step makes it possible to engage in a business-driven TPI NEXT program initiated by the IT department.

3. Identify the most (and least) important Key Areas for the IT goal

It is important that although some general considerations are valid, industry and even company-specific circumstances play a highly decisive part in establishing the relative importance and prioritization of Key Areas.

4. Shift Checkpoints to an adjacent Cluster in line with the Key Area's importance

Checkpoints of Key Areas considered to be the most important are shifted to 'earlier' Clusters (Checkpoints in Cluster M move 'back' to L, L moves to K, etc.). Checkpoints of Key Areas considered to be least important are shifted to 'later' Clusters (Checkpoints in Cluster A to B, B to C, etc.).

5. Take into account the dependencies between Checkpoints

The base clustering constitutes a logical and coherent way to gradually fulfil Checkpoints and to reach a certain Maturity Level. By shifting Checkpoints to other Clusters, some of this logic might become invalidated or compromized.

For example, the first Checkpoints at the Controlled level of Key Area 'Stakeholder commitment' ("The principal stakeholder is defined") and 'Test strategy' ("The principal stakeholder agrees with the documented Test strategy") are both part of Cluster A (see Figure 4). Obviously, Checkpoint "The principal stakeholder is defined" must always be in the same Cluster or an earlier one than "The principal stakeholder agrees with the documented Test strategy".

6. Balance the Clusters

The primary goal of Clusters is to provide convenient, well-organized sets of coherent Checkpoints. If, after re-arranging the Checkpoints, Clusters become too big, too small or in any way unbalanced, feel free to improve the Cluster arrangement by moving a limited number of Checkpoints.

In more detail: TPI NEXT adjusted to a specific environment

The same principle can be followed to create Clusters that are attuned to specific IT environments and situations, such as Agile development methods, software maintenance, development testing and organizations with multiple test processes.

TPI NEXT can also be used to cut test costs

Not unsurprisingly, we have found that a common business driver is to "Provide a good return on (IT-enabled) business investments". This business driver translates into the IT goal "Improve IT's cost efficiency", which for testing means "Reduce the cost of testing". Our example below describes how to adjust the cluster arrangement to accommodate this driver.

The essential 5 Key Areas for cost-cutting

We believe that the following 5 Key Areas are the most effective in reducing the costs of the test process and therefore these have increased priority:

- **Test strategy**; establishing a test strategy, based on a product risk analysis, helps the principal stakeholder to decide which test items have the highest priority and the required test coverage. Rather than testing everything equally thoroughly, the test depth can be reduced for identified system parts and quality characteristics that have a relatively low product risk.
- **Test tools**; the use of test tools can significantly reduce the costs of testing. Automated test execution is much cheaper than manual test execution, but the use of specialist test tools facilitates more efficient test, defect and testware management (configuration management). Without specific tools, this is hard to accomplish, especially in larger development and test teams.
- **Defect management**; proper defect management contributes to solving defects as efficiently as possible, especially if developers too have sufficient access to the same tools as testers. More advanced defect management employs measures to avoid defects, for example in root cause analysis.
- **Test organization**; many organizations rationalize their business activities by concentrating on projects that will generate profits immediately, while keeping the line organization (viewed as a cost centre) at a minimum. However, projects are constrained if they cannot take advantage of economies of scale. Significant cost sav-

SOGETI

Introducing TPI® NEXT

Sogeti's Business Driven Test Process Improvement

The world's leading approach for improving test processes... has now been enhanced!

Sogeti's new book, **TPI® NEXT, Business Driven Test Process Improvement** written by Sogeti test experts and validated by extensive customer field tests, builds on the strengths of the successful original model and provides invaluable insight and recommendations on the maturity of an organization's test processes.

What's new about TPI® NEXT?

— Key focus on alignment with business drivers is at the heart of the model

— Reflects Agile and Outsourcing changes in testing environment

— New 'Enablers' to identify the impact on broader Software Development Lifecycle

— Clear maturity levels – Controlled, Efficient and Optimizing

— Easy-to-view representation for business management.

How TPI® NEXT can help your test organization

— Develop a clear improvement path to improve quality, reduce time and save costs

— Ensure processes support the most important business drivers

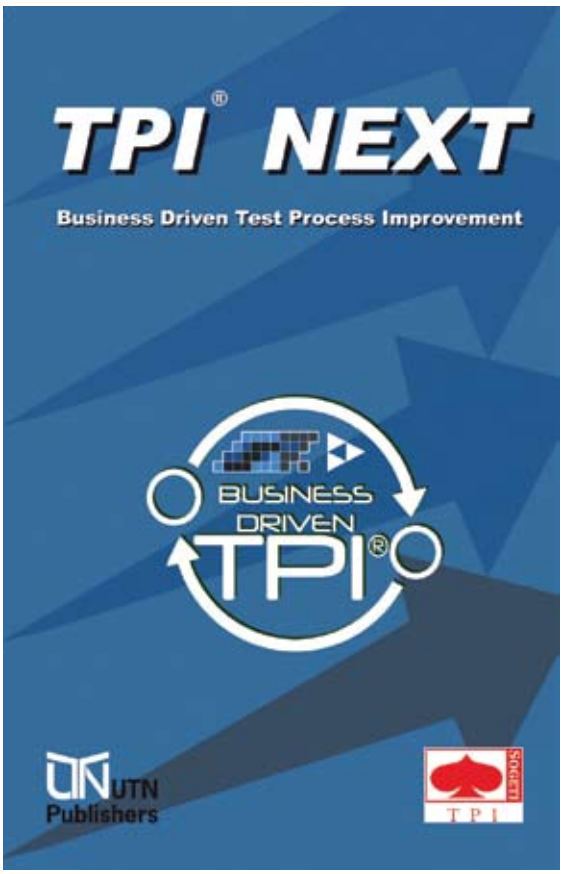
— Gain a better understanding of the correlation between testing and other software development processes.

Order your copy now!

Available from 17 November 2009 from specialist publisher UTN at www.utn.nl or contact Sogeti at tpi@sogeti.nl.

TPI® NEXT an indispensable step-by-step guide to improving your organization's testing processes.

www.sogeti.com



A Communit

over 110,000 C

Are you on th

www.is

*March 2009


y Worldwide!
Certifications*
e right track?
tqb.org
ISTQB®
International Software
Testing Qualifications Board

ings can be realized by acquiring and maintaining scarce resources, like skilled test specialists, test environments, and test tools within the line organization, and then providing them as a service to other projects as required.

- **Degree of involvement**; many problems in the development process manifest themselves in the test process; typically project costs at the testing phase appear to run out of control. By involving testing earlier in the development process and allowing testing to influence the development project, many of these problems can be avoided. Sufficient involvement early in the project also results in defects being found sooner, resulting in lower fixed costs and less uncertainty about project rework.

Conversely, the following 4 Key Areas are usually considered to be least effective in contributing to test cost reduction:

- Reporting; while reporting is important for monitoring costs and managing the test process in a cost-effective way, basic reporting will suffice. Improving reporting in itself does not contribute to saving test costs.
- Metrics; as with reporting, basic metrics are necessary to provide support for saving costs. But while more intense use of metrics provides more detailed control of test process management, it does not contribute to cost reduction itself.
- Stakeholder commitment; testing certainly benefits from strong stakeholder commitment in many ways, but saving costs for testing is not really one of them.
- Test environment; the management of test environments and test data can cost a lot of money and often there is a strong desire to reduce these costs. In this case, effort should be spent on improving test environment management.

Impact of prioritizing Key Areas on the Clusters

Figure 5 below provides a visual presentation of the Clusters. The Key Areas have been prioritized into High, Neutal and Low (H, N & L).

		H	N	L	Initial	Controlled				Efficient				Optimizing		
1	Stakeholder commitment			x		B	C	C	D	G	I	I		L	M	M
2	Degree of involvement	x				A	A	B	D	G	G	I		K	K	
3	Test strategy	x				A	A	A	D	E	E	G		J	K	
4	Test organization	x				A	C	C	D	H	H	I	I	J	K	K
5	Communication		x			B	C	C	D	F	F	J		M	M	
6	Reporting			x		B	D	D		G	H	H		L	L	
7	Test process management		x			A	A	B	B	G	H	J		K	M	
8	Estimating and planning		x			B	B	C	C	G	H	I	I	K	L	L
9	Metrics			x		D	D	E		H	I	I	J	L	L	
10	Defect management	x				A	A	A	C	E	E	G	I	J	K	K
11	Testware management		x			B	B	D	E	I	I	J		L	L	L
12	Methodology		x			C	D	E		F	H	J	J	M	M	
13	Tester professionalism		x			D	D	E	E	G	G	I	I	K	K	M
14	Test case design		x			A	A	E		F	I	I	J	K	K	M
15	Test tools	x				D	D	D		E	F	F	H	K	L	L
16	Test evironment			x		D	E	E	F	H	I	K	K	M	M	M

Figure 5. Categorized Key Areas for 'Test cost reduction' as the business driver

After re-arranging the Checkpoints, the Key Area 'Stakeholder commitment' has no more Checkpoints in Cluster A. Two dependencies have now been compromized: '1.C.1' cannot be part of a Cluster later than '3.C.1', so '1.C.1' moves from Cluster B back to Cluster A, and '1.E.1' cannot be part of a Cluster later than '3.E.1' and so moves back from Cluster G to Cluster E.

The Cluster arrangement for Test cost reduction has now changed and is represented in Figure 6:

		H	N	L	Initial	Controlled				Efficient				Optimizing		
1	Stakeholder commitment			x		A	C	C	D	E	I	I		L	M	M
2	Degree of involvement	x				A	A	B	D	G	G	I		K	K	
3	Test strategy	x				A	A	A	D	E	E	G		J	K	
4	Test organization	x				A	C	C	D	H	H	I	I	J	K	K
5	Communication		x			B	C	C	D	F	F	J		M	M	
6	Reporting			x		B	D	D		G	H	H		L	L	
7	Test process management		x			A	A	B	B	G	H	J		K	M	
8	Estimating and planning		x			B	B	C	C	G	H	I	I	K	L	L
9	Metrics			x		D	D	E		H	I	I	J	L	L	
10	Defect management	x				A	A	A	C	E	E	G	I	J	K	K
11	Testware management		x			B	B	D	E	I	I	J		L	L	L
12	Methodology		x			C	D	E		F	H	J	J	M	M	
13	Tester professionalism		x			D	D	E	E	G	G	I	I	K	K	M
14	Test case design		x			A	A	E		F	I	I	J	K	K	M
15	Test tools	x				D	D	D		E	F	F	H	K	L	L
16	Test evironment			x		D	E	E	F	H	I	K	K	M	M	M

Figure 6. Final Cluster re-arrangement for the business driver 'Test cost reduction'

In this revised situation, Cluster A contains 14 Checkpoints, Cluster B has 9 Checkpoints, etc. Other Clusters have Checkpoints from different levels of maturity: For example, Cluster E has 8 from the Controlled level and 6 from the Efficient level. This is caused by the fact that a further step in improvement (in our cost-driven example, reducing the cost of the test process) may also require Checkpoints from other levels of maturity.

Cluster D contains many Checkpoints, as do Clusters I and K; Cluster F on the other hand contains just a few Checkpoints. Depending on the specific situation in the assessed organization or project, you should consider shifting Checkpoints so that the number of Checkpoints per Cluster is more evenly spread.

Conclusion

Business dynamics and drivers change over time and from entity to entity, but TPI NEXT is highly flexible and adaptable, and can work independently or in sync with other test methodologies, including of course Sogeti's own TMap.

By putting the business drivers at the heart of the new model, we have seen very clear benefits; a clear improvement path understood and endorsed by the business, optimization of test processes in line with the most important business drivers, and a better understanding of the correlation between testing and adjacent software development processes. We believe that our enhancements make TPI®NEXT an indispensable step-by-step guide to improving your organization's testing processes.

More information on TPI NEXT can be found at www.tpinext.com. Copies of TPI®NEXT are now available from specialist publisher UTN at www.utn.nl or contact Sogeti at tpi@sogeti.nl.

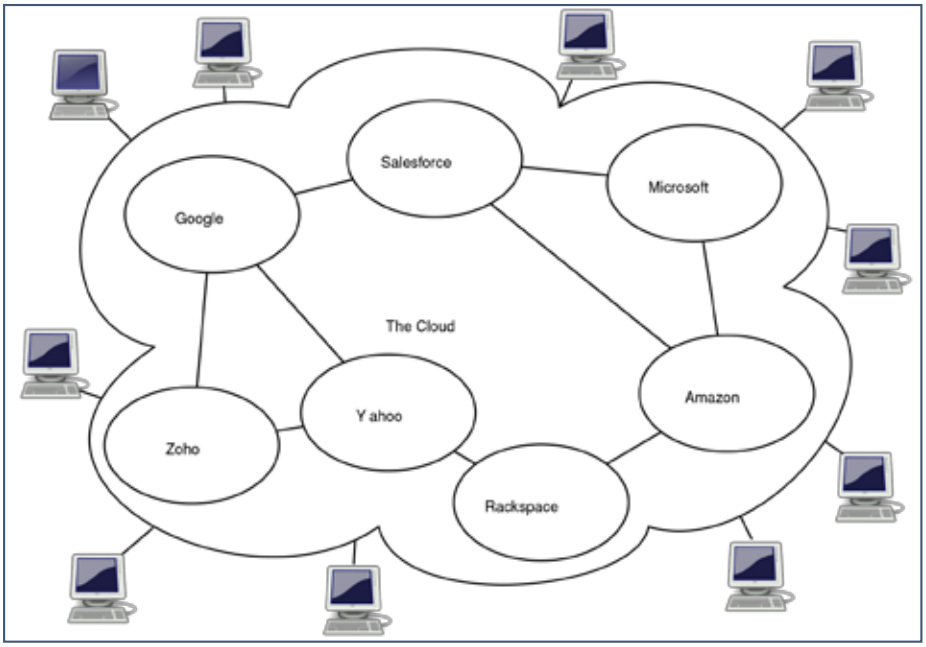
Biography

Bert Linker has over 12 years testing experience, in the Netherlands and other countries, as test consultant, test manager and senior trainer of test topics and methodologies. He has gained his knowledge and experience from numerous test assignments in both traditional as well as RUP environments. He is one of the initiators and co-authors of Sogeti's latest publication TPI® NEXT.

Ben Visser is a highly experienced test manager and test consultant. In a career of over 15 years as a tester, he has fulfilled a wide range of test functions, from programmer of automated scripts, to test manager of a large international program. He has worked in traditional waterfall development as a change manager responsible for test and acceptance environments, as well as model-based testing. Based on this extensive experience, he co-authored Sogeti's TMap NEXT® Business Driven Test Management (Nov 08) and more recently TPI® NEXT.

Standards: Do We Really Need Them In The Age Of Cloud Computing?

by Koray Yitmen



Cloud computing logical diagram from Wikipedia.org

Nowadays' buzz word is Cloud Computing. In the near future, all users are expected to have only LCD screens, keyboard and an Internet Access to perform their daily routine tasks in business and social life. This means a shift from decentralization, where each individual has the whole power, to a centralization, where all control is in the hands of the main players of the web. These main players will provide services mainly in three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). So if users will just need a dummy terminal to perform all their tasks, and there are a few main players, who will control all the web like oligarchy, the word 'standard' may lose its meaning? Let us try to answer this question by imagining our future daily life. Here we go:

- Enter the office
- Open your dummy terminal
- Check your emails from self-customized gmail specific to your company's needs
- Generate a spread sheet from Google Docs to analyze your sales figures and just click share button to send them to your colleques
- Watch your favourite clips from your customized YouTube library and share them
- and so on …

This scenario seems similar to the one that we experienced twenty years ago, when we connected to mainframes via dummy terminals with greenish screens, doesn't it? This scenario may lead you to think that there is less need for standards, since everything is under the control of a few players who will agree easily on a common standard and our lives will be very easy. This may be the case in the mature ages of cloud computing, but not in its childhood stages. Let us make some analogies:

- In 1899, there were 2000 local telephone companies in US many, all of which were running their own standards of transmission
- In 1890, electricity came in a variety of voltages and frequencies

Today, we never even think about the standard of electricity we have to use while travelling among cities, which are not only in the same country but also on different continents. The standards have been reduced to two: 110 V and 220 V, and it becomes very easy to convert between these voltages with a simple, cheap appliance. But the way to come up to this consolidation is a hard drive, where many companies and standards have disappeared or been acquired by their competitors. So the weather forecast for standards during this cloudy hard drive would probably be:

Weather Forecast for Standards in Cloud Computing		
Foggy		• Lots of similar standards • Every standard seems doing well • 'Aha!' ideas are all around
Fake sun		• Two or three main players • A small mistake may lead to death
Clear sky		• Winner takes it all • De facto or de jure standard • Users are not even aware of the standard, it is embedded

Let us check the ground in this foggy weather during today's initial stages:

File: test_reports_and_screenshots.zip Size:1.67 GB

100%

Recipients: John M., Edward J.

SEND COMPLETED!!!

- **Google**: born as a pure cloud company is now expanding its roots by giving Android and Chrome OS away as open source, and Google Docs
- **Microsoft**: once a very closed network company now supports many open standards. Bing search engine initiative is a move to get stake in search engine market to promote its future cloud offerings
- **Apple**: another company who has strong roots in cloud is now building a $1 billion data center in North Carolina to handle its future cloud services

This changing weather will trigger a chain reaction in all disciplines, especially in testing, where there is heavy fog over its own standard test process models , such as Test Maturity Model (TMMi), Systematic Test and Evaluation Process (STEP), Critical Testing Processes (CTP) and Test Process Improvement (TPI). These models are used for test process improvement and are still in the early stages of their lifecycles. They should be very carefully chosen while selecting which hill to climb, since in these turbulent times it is very difficult to discern which hills are highest and which summits are false. Cloud computing will blow unexpected winds, which may change the foggy areas.

So our answer to the above topic is 'Yes. We will need standards even more, but we have to wait a little bit for the mature one.'

Biography

Koray Yitmen has been working as a Managing Partner at Keytorc Software Testing Services (www.keytorc.com) since 2004. He is a board member of ISTQB (International Software Testing and Qualifications Board), and he is the President of the Turkish Testing Board. Formerly, he worked as a consultant in various technology and business consulting projects for local and international clients during his engagement at Andersen, Accenture and Booz Allen Hamilton. He holds a B.Sc. degree in Computer Science from Middle East Technical University, Turkey and MBA degree in Entrepreneurship from Babson College, USA. He has published many articles about software testing and given speeches at many software testing conferences. He is a strong supporter of the importance of software testing and tries to achieve his vision through conducting seminars and workshops at various non-profit organizations.

NEU

Graham Bath, Judy McKay

Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst

Aus- und Weiterbildung zum Certified Tester – Advanced Level nach ISTQB-Standard

2009, 430 Seiten, Festeinband
€ 42,00 (D)
ISBN 978-3-89864-591-1

„Erstes deutsches Buch zum Modul „Test Analyst und Technical Test Analyst" des ISTQB Certified-Tester-Programms, Advanced Level

Die englische Originalausgabe ist im Rocky Nook Verlag erschienen.

Das Buch deckt sowohl funktionale als auch technische Aspekte des Softwaretestens ab und vermittelt damit das notwendige Praxiswissen für Test Analysts und Technical Test Analysts – beides entscheidende Rollen in Testteams.

Es umfasst den benötigten Stoff zum Ablegen der Prüfung »Certified Tester-Advanced Level – Test Analyst/Technical Test Analyst« nach ISTQB-Standard. Der Lehrstoff wurde um zusätzliche Informationen und Beispiele aus der Praxis erweitert.

dpunkt.verlag

Ringstraße 19 B · D-69115 Heidelberg · fon: 0 62 21 / 14 83 40 · fax: 0 62 21 / 14 83 99 · e-mail: bestellung@dpunkt.de · www.dpunkt.de

dpunkt.computing

Advanced Software Test Design Techniques: Use Cases

by Rex Black

The following is an excerpt from my recently-published book, *Advanced Software Testing: Volume 1*. This is a book for test analysts and test engineers. It is especially useful for ISTQB Advanced Test Analyst certificate candidates, but contains detailed discussions of test design techniques that any tester can—and should—use. In this third article in a series of excerpts, I discuss the application of use cases to testing workflows.

Use Cases

At the start of this series, I said we would cover three techniques that would prove useful for testing business logic, often more useful than equivalence partitioning and boundary value analysis. First, we covered decision tables, which are best in transactional testing situations. Next, we looked at state-based testing, which is ideal when we have sequences of events that occur and conditions that apply to those events, and the proper handling of a particular event/condition situation depends on the events and conditions that have occurred in the past. In this article, we'll cover use cases, where preconditions and postconditions help to insulate one workflow from the previous workflow and the next workflow. With these three techniques in hand, you have a set of powerful techniques for testing the business logic of a system.

Conceptually, use case testing is a way to ensure that we have tested typical and exceptional workflows and scenarios for the system, from the point of view of the various actors who directly interact with the system and from the point of view of the various stakeholders who indirectly interact with the system. If we (as test analysts) receive use cases from business analysts or system designers, then these can serve as convenient frameworks for creating test cases.

Remember that, with decision tables, we were focused on transactional situations, where the conditions—inputs, preconditions, and so forth—that exist at a given moment in time for a single transaction are sufficient by themselves to determine the actions the system should take. Use cases aren't quite as rigid on this point, but, generally, the assumption is that the typical workflows are independent of each other. An exceptional workflow occurs when a typical workflow can't occur, but usually we have independency from one workflow to the next. This is ensured—at least for formal use cases—by clearly defined preconditions and postconditions, which guarantee certain things are true at the beginning and end of the workflow. This acts to insulate one workflow from the next. Again, if we have heavy interaction of past events and conditions with the way current events and conditions should be handled, we'll want to use state-based testing.

The model is less formal than what we've seen with decision tables and state-based testing. Indeed, the concept of a "use case" itself can vary considerably in formality and presentation. The basic idea is that we have some numbered (or at least sequential) list of steps that describes how an actor interacts with the system. The steps can be shown in text or as part of a flow chart.

The use case should also show the results obtained at the end of that sequence of steps. The results obtained should benefit some party, either the actor interacting directly with the system or some other stakeholder who indirectly receives the value of the results.

At the very least, the set of steps should show a typical workflow, the normal processing. This normal processing is sometimes called the primary scenario, the normal course, the basic course, the main course, the normal flow, or the "happy path". However, since things are not always happy, the set of steps should also show abnormal processing, sometimes called exceptions, exceptional processing, or alternative courses.

Approaches to documenting use cases that are more formal cover not only typical and exceptional workflows, but also explicit identification of the actor, the preconditions, the postconditions, the priority, the frequency of use, special requirements, assumptions, and potentially more. The formal approach might also entail the creation of a use case diagram that shows all the actors, all the use cases, and the relationship between the actors and the use cases.

Now, an assumption that I'm making here—in fact, it's an assumption implicitly embedded within the ISTQB syllabi—is that you are going to receive use cases, not create them. If you look at both the Advanced and Foundation Level syllabi, they talk about use cases as something that test analysts receive, upon which they base their tests. So, rather than trying to cover the entire gamut of use case variation that might exist in the wild and wooly world of software development, I'm going to talk about using basic, informal use cases for test design, and talk about using more formalized use cases for test design, and leave out too much discussion about variations.

So, assuming we receive a use case, how do we derive tests? Well, at the very least, we should create a test for every workflow, including both the typical and exceptional workflows. If the exceptional workflows were omitted, then you'll need to figure those out, possibly from requirements or some other source. Failing to test exceptions is a common testing mistake when using informal use cases.

Creating tests can involve applying equivalence partitioning and boundary value analysis along the way. In fact, if you find a situation where combinations of conditions determine actions, then you might have found an embedded, implied decision table. Covering the partitions, boundaries, and business rules you discover in the use case might result in two, five, ten, twenty, or more test cases per work flow, when you're all done.

Remember that I said that a use case has a tangible result. So, part of evaluating the results of the test is verifying that result. That's

above and beyond verifying proper screens, messages, input validation, and the like as you proceed through the workflow.

Note the coverage criterion implied above: At least one test per workflow, including both typical and exceptional workflows. That's not a formal criterion, but it's a good one to remember as a rule of thumb.

What is our underlying bug hypothesis? Remember in decision tables we were looking for combinations of conditions that result in the wrong action occurring or the right action not occurring. With use cases, we're a bit more coarse-grained. Here, we are looking for a situation where the system interacts improperly with the user or delivers an improper result.

E-commerce purchase: Normal workflow

1. Customer places one or more Items in shopping cart
2. Customer selects checkout
3. System gathers address, payment, and shipping information from Customer
4. System displays all information for User confirmation
5. User confirms order to System for delivery

Exceptions

- Customer attempts to checkout with empty shopping cart; System gives error message
- Customer provides invalid address, payment, or shipping information; System gives error messages as appropriate
- Customer abandons transaction before or during checkout, System logs Customer out after 10 minutes of inactivity

Figure 1: Informal Use Case Example

In Figure 1, we see an example of an informal use case describing purchases from an e-commerce site, like the rbcs-us.com example shown for decision tables in the earlier article.

At the top, we have the web site purchase normal workflow. This is the happy path.

1. Customer places one or more Items in shopping cart
2. Customer selects checkout
3. System gathers address, payment, and shipping information from Customer
4. System displays all information for User confirmation
5. User confirms order to System for delivery

Note that the final result is that the order is in the system for delivery. Presumably another use case having to do with order fulfillment will describe how this order ends up arriving at the customer's home or place of business.

We also see some exception workflows defined.

For one thing, the Customer might attempt to checkout with an empty shopping cart. In that case, the System gives an error message.

For another thing, the Customer might provide an invalid address, payment, or shipping information. On each screen—if we're following the typical e-commerce flow—the System gives error messages as appropriate and blocks any further processing until the errors are resolved.

Finally, the Customer might abandon the transaction before or during checkout. To handle this, the System logs the Customer out after 10 minutes of inactivity.

Now, let's look at deriving tests for this use case. In Figure 2, you see the body of the test procedure to cover the typical workflow. (For brevity's sake, I've left off the typical header and footer information found on a test procedure.)

#	Test Step	Expected Result
1	Place 1 item in cart	Item in cart
2	Click checkout	Checkout screen
3	Input valid US address, valid payment using American Express, and valid shipping method information	Each screen displays correctly and valid inputs are accepted
4	Verify order information	Shown as entered
5	Confirm order	Order in system
6	Repeat steps 1-5, but place 2 items in cart, pay with Visa, and ship international	As shown in 1-5
7	Repeat steps 1-5, but place the maximum number of items in cart and pay with MasterCard	As shown in 1-5
8	Repeat steps 1-5, but pay with Discover	As shown in 1-5

Figure 2: Deriving Tests Example (Typical)

As you can see, each step in the workflow has mapped into a step in the test procedure. You can also see that I did some equivalence partitioning and boundary value analysis on the number of items, the payment type, and the delivery address. Because all of these selections are valid, I've combined them. Note that space-saving approach of describing how to repeat the core steps of the test procedure with variations, rather than a complete re-statement of the test procedure, at the bottom.

In Figure 3, you see the body of the test procedure to cover the exception flows. You can see that I use equivalence partitioning on the points at which the customer could abandon a transaction.

This is a good point to bring up an important distinction, that between logical and concrete test cases. For the ISTQB exam, you'll want to make sure you know the Glossary definitions for these terms. For our purposes here, we can say that a logical or high-level test case describes the test conditions and results. A concrete or low-level test case gives the input data to create the test conditions, and the output data observed in the results.

As you just saw in Figure 2 and Figure 3, you can easily translate a use case into one or more logical test cases. However, translation of the logical test case into concrete test cases can require additional documentation. For example, what was the maximum number of items we could put in the shopping cart? We'd need

#	Test Step	Expected Result
1	Do not place any items in cart	Cart empty
2	Click checkout	Error message
3	Place item in cart, click checkout, enter invalid address, then invalid payment, them invalid shopping information	Error messages, can't proceed tonext screen until resolved
4	Verify order information	Shown as entered
5	Confirm order	Order in system
6	Repeat steps 1-3, but stop activity and abandon transaction after placing item in cart	User logged out exactly 10 minutes after last activity
7	Repeats steps 1-3, but stop activity and abandon transaction on each screen	As shown in 6
8	Repeat steps 1-4; do not confirm order	As shown in 6

Figure 3: Deriving Tests Example (Exception)

some further information, ideally a requirements specification, to know that. What items can we put in shopping cart? Some description of the store inventory is needed.

Is it cheating to define logical test cases rather than concrete ones? No, absolutely not. However, notice that, at some point, a test case must become concrete. You have to enter specific inputs. You have to verify specific outputs. This translation from logical test case to concrete test case is considered an implementation activity in the ISTQB fundamental test process. If you choose to leave implementation for the testers to handle during test execution, that's fine, but you'll need to make sure that adequate information is at hand during test execution to do so. Otherwise, you risk delays.

So, what's different or additional in a formal use case? Usually, a formal use case contains more information than an informal one. Here, you can see some of the typical elements of a formal use case:

- ID—some use case identifier number
- Name—a short name, like E-commerce Purchase
- Actor—the actor, such as Customer
- Description—a short description of the use case
- Priority—the priority, from an implementation point of view
- Frequency of use—how often this will occur
- Preconditions—what must be true to start the use case normally
- Typical workflow—often like the informal use case, but sometimes broken into two columns, one for the actor actions and one for the system response
- Exception workflows—one for each exception, often also with actor action and system response columns.
- Postconditions—what should be true about the state of the system after the use case completes normally

Notice that you can use some of this information as a test analyst. Some, like the priority and frequency of use, you might not use, except during the risk analysis process. Also, notice that the breakdown on the workflows, especially the exception workflows, is finer-grained, so your test traceability can be finer-grained, too.

Typical workflow	1. System gathers address, payment, and shipping information from Customer 2. System displays all information for User confirmation 3. User confirms order to System for delivery
Exception 1	Customer attempts to checkout with empty shopping cart System gives error message
Exception 2	Customer provides invalid address, payment, or shipping information System gives error message as appropriate
Exception 3	Customer abandons transaction before or during checkout System logs Customer out after 10 minutes of inactivity
Exception 4	Order is active in system

Figure 5: Formal Use Case Example (Part 2)

Figure 4 shows the header information on a formal version of the informal use case we saw earlier. Notice that some of the steps of the informal use case became preconditions. This means that the shopping portion of the use case would become its own use case, allowing this use case to focus entirely on the purchase aspects of the e-commerce site. Notice also that we didn't know about that "logged in" requirement before. That's important information for our testing.

Figure 5 shows the main body of the formal use case, the normal workflow and the three exceptions. Notice the normal workflow is a bit shorter now because some of its steps became preconditions. Also, each exception has its own row in the table. Finally, notice that the postcondition is true only if the normal workflow is ultimately completed.

Conclusion

In this article, I've shown how to apply use cases to the testing of typical and exceptional workflows. We have looked at decision tables as a way to test detailed business rules, state-based methods to test state-dependent systems, and now use cases for workflows. With these three techniques, you can perform a full range of internal business logic testing.

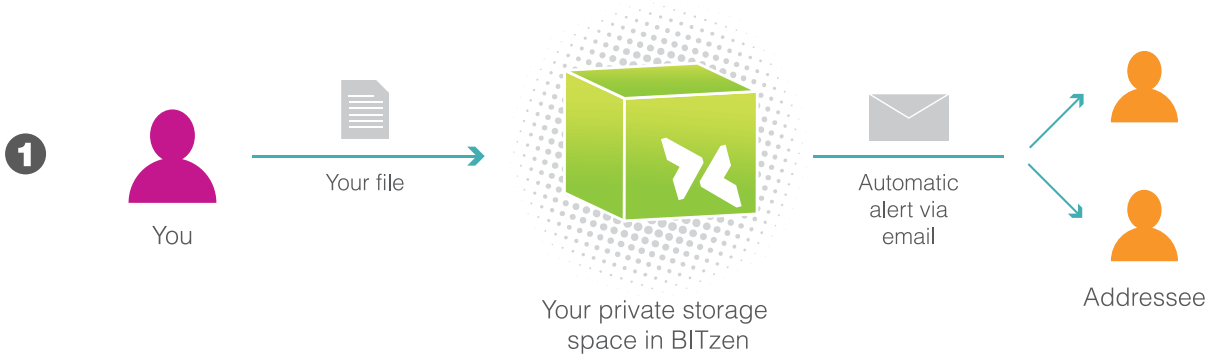
ID	02.001
Name	E-commerce Purchase
Actor	Customer
Description	Allow Customer to complete a transaction by purchasing the item(s) in her shopping cart
Priority	Very high
Frequency in use	25% of Customers, up to 1,000 customers per day
Preconditions	1. One or more items in shopping cart 2. Customer is logged in 3. Customer has clicked on checkout

Figure 4: Formal Use Case Example (Part 1)

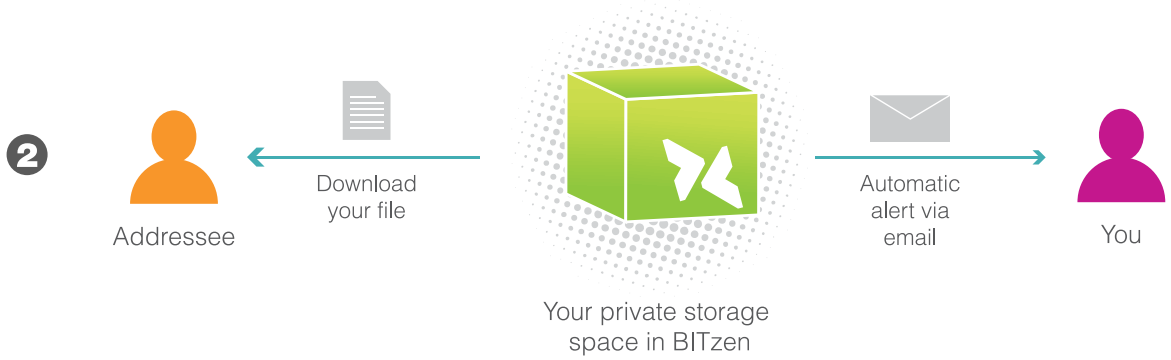
Biography

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide.

Send smoothly...



Upload your file to your private storage space in BITzen.
BITzen inform your addressee via email that they have recieved a new file to download.



Your addressee downloads your file over a secure SSL connection.
BITzen informs you of the successful download.

...with www.bitzen.net

bitzen.net

Exchange files
with customers, suppliers and partners in
a simple, fast and secure way.

- Specifically created for companies and professionals.
- Free account with 100 MB storage space.
- Monthly plans starting from EUR 6.00.
- Paypal.
- Send large files up to 2GB.
- Batch files upload.
- Unlimited contacts.
- Automatic alerts for sent and uploaded files.
- Resend files.
- No additional software installation needed (SaaS).
- No Virtual Disk, no FTP and no Webmail.
- Supports any types of files.
- Compatible with IE, Firefox, Safari, Opera and Chrome.
- Automatic daily backups.
- No advertising. No Spam.
- No public links.
- Unlimited number of downloads.
- Secure SSL.

Web: www.bitzen.net Email: contact@bitzen.net Twitter: @bitzennet

The Test SPICE Approach

Test process assessments follow in the footsteps of software process assessments

by Monique Blaschke, Michael Philipp, Tomas Schweigert

When reviewing commonly used test assessment approaches such as ISTQB, TMMi® and TPI®/TMAP® one soon arrives at the conclusion that there has yet to be an approach that conforms to the ISO/IEC 15504 II standard. The new Test SPICE approach fills this gap. The ISO/IEC 15504-5 standard was used as a starting point for Test SPICE. This article describes the new model.

The ISO/IEC 15504 standard was designed to be enhanced by the development of specific process reference models (PRMs) and process assessment models (PAMs) [1]. Specific models are now available. The best known is automotive SPICE®, developed by the user group in coordination with the automotive domain. Additional parts published as technical reports provide an exemplar system life cycle process assessment model (Part 6) and serve to assess organizational maturity (Part 7).

There are two major assessment methods in the test business market: TPI®/TMAP® and TMMi®. There are also various schemes for training test professionals such as ISTQB, which is recognized as a de facto standard at least in Germany. The ISTQB education scheme implies an own process model that is used, say, for organizing the test from a test management perspective. Because of this our team decided to judge the ISTQB approach as a topic for evaluation and a potential source for test processes. The first question to answer was: Does one or more of these models comply with ISO/IEC 15504 Part 2, and if not, what should a compliant model be like?

The conformity requirements of ISO/IEC 15504-2

The PAM must specify

- The selected PRM(s)
- The selected processes taken from the PRM and
- Capability levels taken from the measurement framework.

The model must also describe the mapping between it and

- the process reference model
- the measurement framework

As long as a model for assessing tests follows this structure, conformity with ISO/IEC 15504 Part 2 is assured. The Test SPICE approach is designed to fulfil these essential preconditions.

The test process models currently available

With ISTQB®, TPI®/TMAP® and TMMi® three major models are available in the market. But do these models fulfil the requirements of ISO/IEC 15504-2?

ISTQB

The International Software Testing Qualifications Board (ISTQB) provides a set of syllabi for the qualification of test people (e.g. foundation level, advanced level: functional tester or test manager and expert level certified test process improver). [2] [3] [4] [5] Even though we knew that the ISTQB does not claim to provide a process reference model, the team decided to include the model in the evaluation. The ISTQB syllabus provides a fundamental test process: planning and control; analysis and design; implementation and execution; evaluating exit criteria and reporting; test closure activities. It also contains a glossary [8] [9]. The description of processes is heterogeneous. Sometimes a process is described with its purpose, but no explicit description of outcomes is available. Based on this the fundamental ISTQB test process does not meet the conformity requirements of ISO/IEC 15504 Part 2. Our team nevertheless considered the content of this model to be a useful input.

TPI®/TMAP®

TPI®/TMAP® is the test process assessment and improvement method of the testing service provider SOGETI. The assessment is based on a questionnaire that covers the needs of software testing. The approach uses a two-step maturity model. Each check point can be fulfilled to up to four levels (A–D) that require fulfilment of different aspects of the check point. These levels are mapped to three general maturity levels (controlled, efficient, optimizing). The model does not provide for mapping to the capability levels and process attributes of ISO/IEC 15504 [7] [10] (This result is based on the first version of TPI®. Since the development of this paper a new version has been published. It could be that analysis of this version will lead to a different conclusion). Result: TPI®/TMAP® (first version) does not meet the conformity requirements of ISO/IEC 15504-2.

(see Fig. 1)

TMM(SM)/TMMi®

TMM(SM) was initially developed by the Illinois Institute of Technology and is now maintained as TMMi® by the TMMi Foundation. The aim of this initiative was to use the CMM®/CMMI® approach for test process assessment and improvement. The current published model is based on the graduated CMMi®approach, with processes directly linked to maturity levels. In contrast to CMMI®, TMMi® currently has no continuous representation [6]. A continuous approach enables the tester to define the capability level of each process and to deliver a capability profile. Unlike a model that only provides a graduated representation, continuous representation may fulfil the conformity requirements of ISO/IEC 15504-2. Result: TMMi® also fails to meet the compliance requirements of ISO/IEC 15504-2.

(see Fig. 2)

The new Test SPICE approach

The aim of the Test SPICE approach is to deliver a PRM and a PAM that both meet the conformity requirements of ISO/IEC 15504-2 and cover the processes required to effectively and efficiently assure the quality of software products.

Key Area	Reached TPI Matrix													
	0	Controlled					Efficient					Optimizing		
1 - Test Strategy		A					B				C		D	
2 - Life Cycle Model		A			B									
3 - Moment of Involvement			A				B				C		D	
4 - Estimating and Planning				A							B			
5 - Test Design Techniques		A		B						C				
6 - Static Test Techniques					A		B							
7 - Metrics						A			B			C		D
8 - Test Automation				A				B			C			
9 - Test Environment				A				B						C
10 - Office and Laboratory Environment				A										
11 - Commitment and Motivation		A				B						C		
12 - Test Functions and Training				A			B			C				
13 - Scope of Methodology					A		B				C			D
14 - Communication			A		B							C		
15 - Reporting		A			B		C					D		
16 - Defect Management		A				B		C						
17 - Testware Management			A			B				C				
18 - Test Process Management		A		B								C		
19 - Evaluation			A				B			C				
20 - Low-Level Testing					A		B		C					
21 - Integration				A			B				C			

Fig.1: The improvement strategy table of TPI® (example Automotive TPI®)

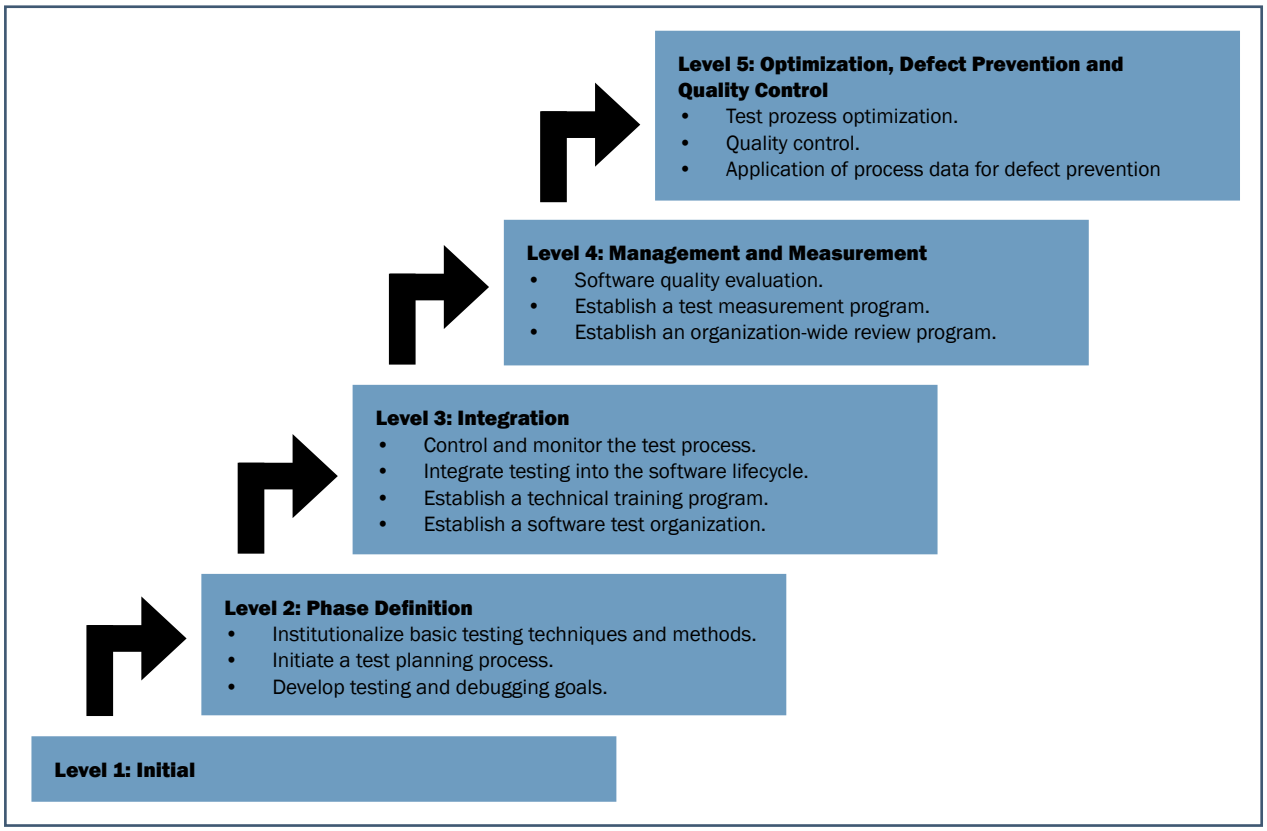


Fig.2: The maturity levels and processes of TMMi®
Quelle: "Developing a Testing Maturity Model, Part II" Ilene Burnstein, Taratip Suwannasart, C.R. Carlson Illinois Institute of Technology

The basis: ISO/IEC 15504-5

Test SPICE was developed with ISO/IEC 15504-5 as its starting point. This model is structured in process categories, process groups and processes. A well-known and practice-proven structure, it is used for the Test SPICE model. The main difference is that the processes themselves are designed to meet the requirements for setting up efficient testing.

For readers who are not familiar with ISO/IEC 15504-5 here is a short overview of the ISO/IEC 15504-5 process model.

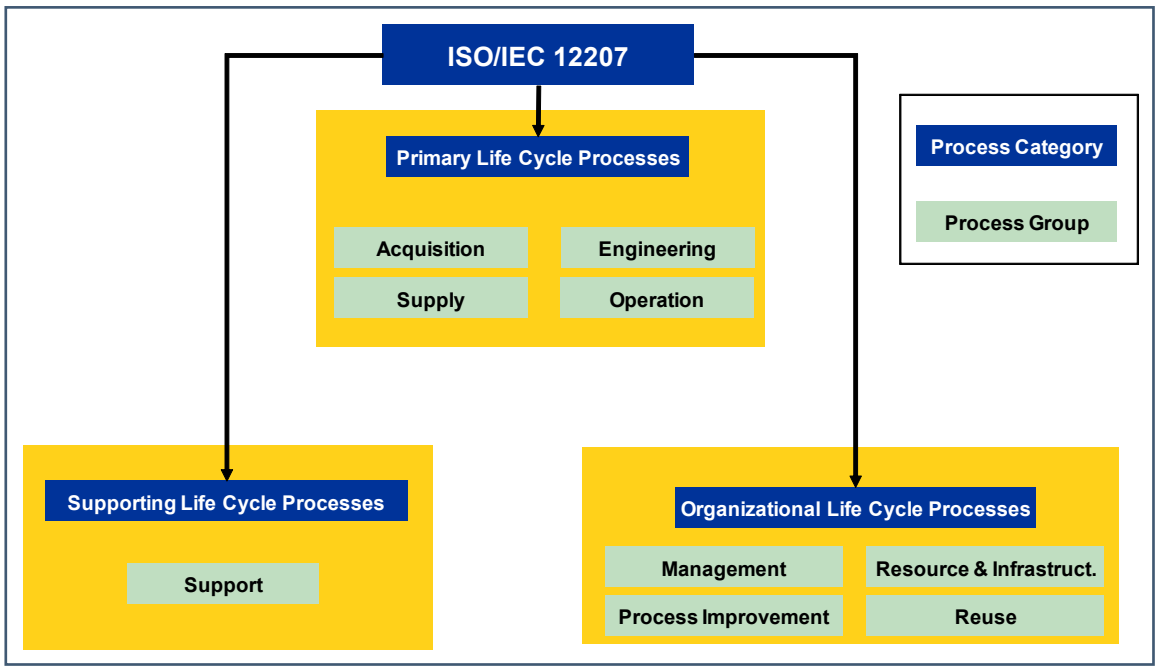


Fig. 3: First we see the overall structure of this model, its process categories and its process groups.

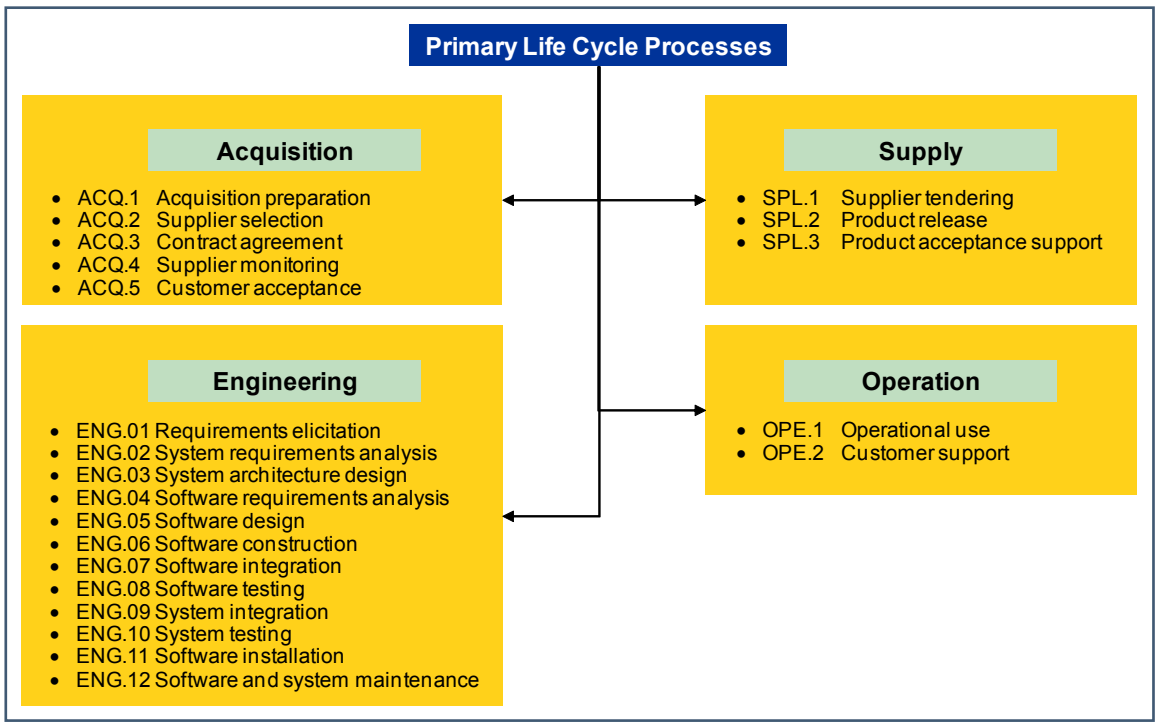


Fig. 4: Next we see the primary life cycle processes of ISO/IEC 15504-5:

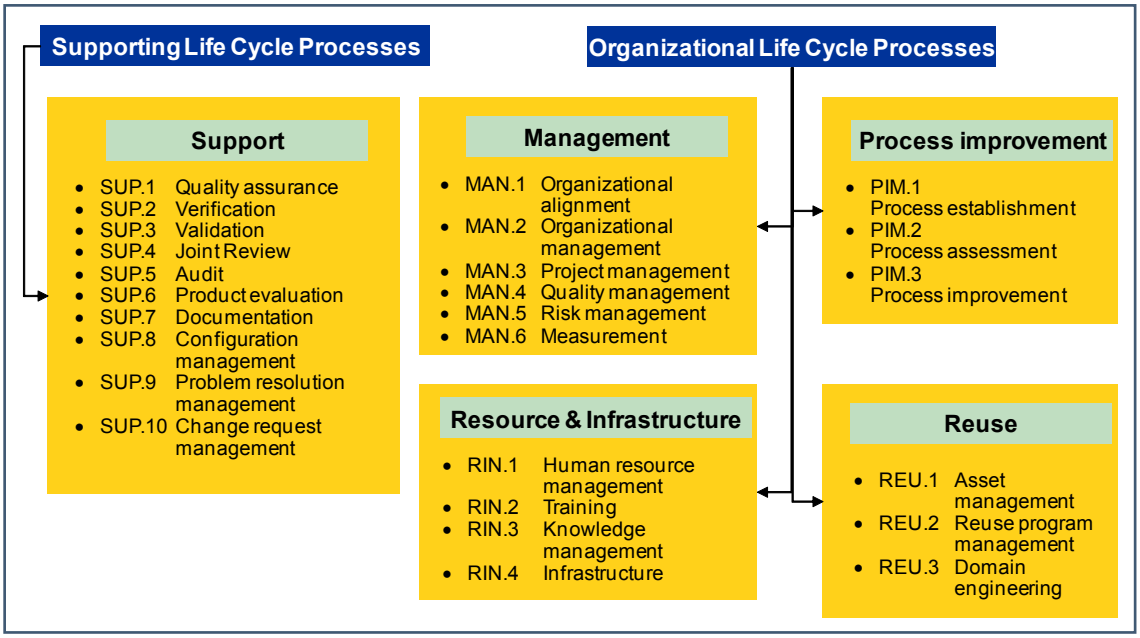


Fig. 5: And finally we see the supporting and organizational life cycle processes:

Looking at the Test SPICE model we see that it has a similar structure to ISO/IEC 15504-5 To transform the original model, four methods were used:

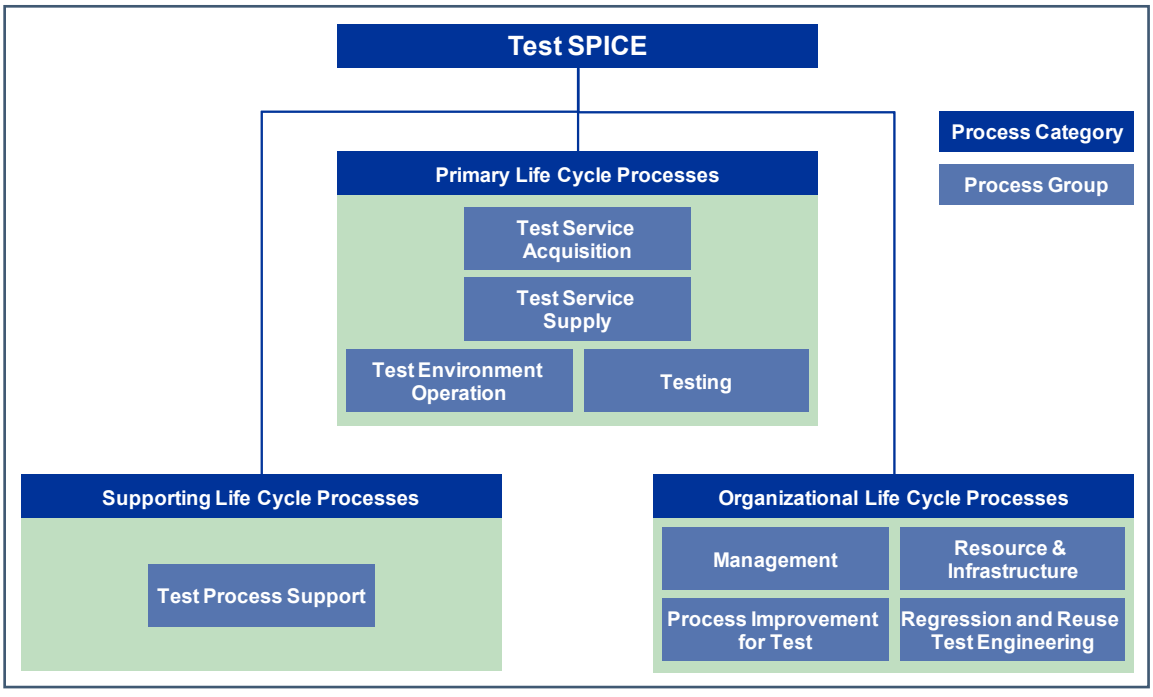


Fig. 6: The overall structure of Test SPICE

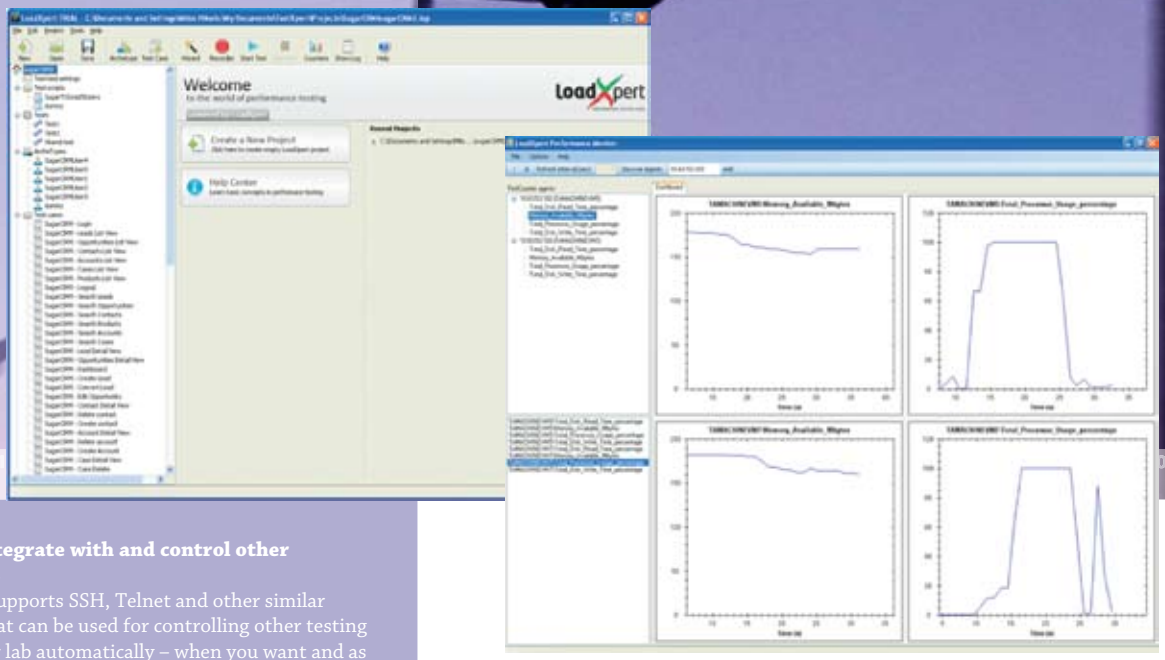
- Do nothing: If the content and focus of a process is the same for the testing the process is transferred 1:1 from ISO/IEC 15504 V to Test SPICE. Example: Project Management
- Replace: Where the content and focus of process is changed the process from the original model is replaced by a process reflecting the specific needs for testing. Example: Domain Engineering was replaced by Regression Test Management
- Rename: If the focus of a process or process group is changed to the testing area the name was changed to reflect this. Example: Support was renamed Test Process Support
- Insert: If specific topics for testing are missing they are added by inserting a new process. Example: Test planning

The process categories were used as defined in ISO 15504.5 and not changed.

Low hardware requirements, scalable system with variable number of load generator machines.
System is a client-server application, where clients are controlled by the server. Server dispatches the load generation tasks onto as many clients as you wish, one client can be installed on one machine. You can use whatever PC machines you have in your LAB.

Easy to use script recording process
Preparing your test scripts has never been easier with our one-click-recording process. You can start the recorder and then continue using your application as you would normally do. LoadXpert will capture all the traffic that goes between your browser and your application. Later that traffic will be parameterized and multiplied onto as many instances as you want.

Test your applications Just as your clients do. And do it affordably.



Ability to integrate with and control other testing tools
LoadXpert supports SSH, Telnet and other similar protocols that can be used for controlling other testing tools in your lab automatically – when you want and as much as you want. Let LoadXpert be your main tool for your complex performance tests.

Flexible and easy to use load design wizard
After you have recorded your test scripts, you can easily design the load that you want to throw at your application, by using LoadXpert load design wizard. You can choose how you want the load to reach its maximum and then how long you want your load test to last, by following on-screen instructions in our wizard.

Server performance monitoring system, which monitors windows/linux performance counters in real-time.
With LoadXpert Monitor you can keep track on your server performance in real-time and record all performance measurements into a database for later processing. LoadXpert Monitor supports both Windows and Linux based servers and applications. It can be used prior, during and after load tests.

Fast growing happy customers list!
Proven tool for performance testing of Web Applications (Cisco and Hypo-Alpe-Adria bank as customers)

- Integrated Development Environment (IDE) for test script editing and traffic recording
- Load Script Wizard, for quick and easy way to design the load scenario during the test
- HTTP/HTTPS Recorder, simple utility within IDE used for capturing HTTP and HTTPS traffic between users and application
- Scalable Client-Server architecture, for emulating concurrent user workload, where clients are controlled by the server. Server dispatches the load generation tasks onto as many clients as you wish, with one client software per machine.
- Scheduled Test Execution, you can schedule your tests, to start at any given moment in time (overnight testing etc)
- Real-time Server Performance Monitoring System, for monitoring and recording server performance by collecting performance counters from all underlying hardware and software servers (both Windows and Linux servers are supported). Performance counters are recorded to a database, for later detailed analysis. Web viewer version available also.
- Automatic Report Generation, test execution report with graphs and data analysis results is just a few clicks away.

- **Performance testing tool offering features of contemporary products**
- **Low hardware requirements tool for SMB and enterprise sector**
- **Easy and intuitive to use environment with quick test preparations**
- **True simulation of real-life load using archetype methodology**
- **Quick test reports generation with more time for results quality judgment**
- **Variability and scalability of test process**
- **Predictability and productivity for performance testing professionals**
- **Return on your investment based on your quantitative results**

Contact us for more information, trial version or try-out option at
http://www.loadxpert.net

LoadXpert
PERFORMANCE TESTING TOOL

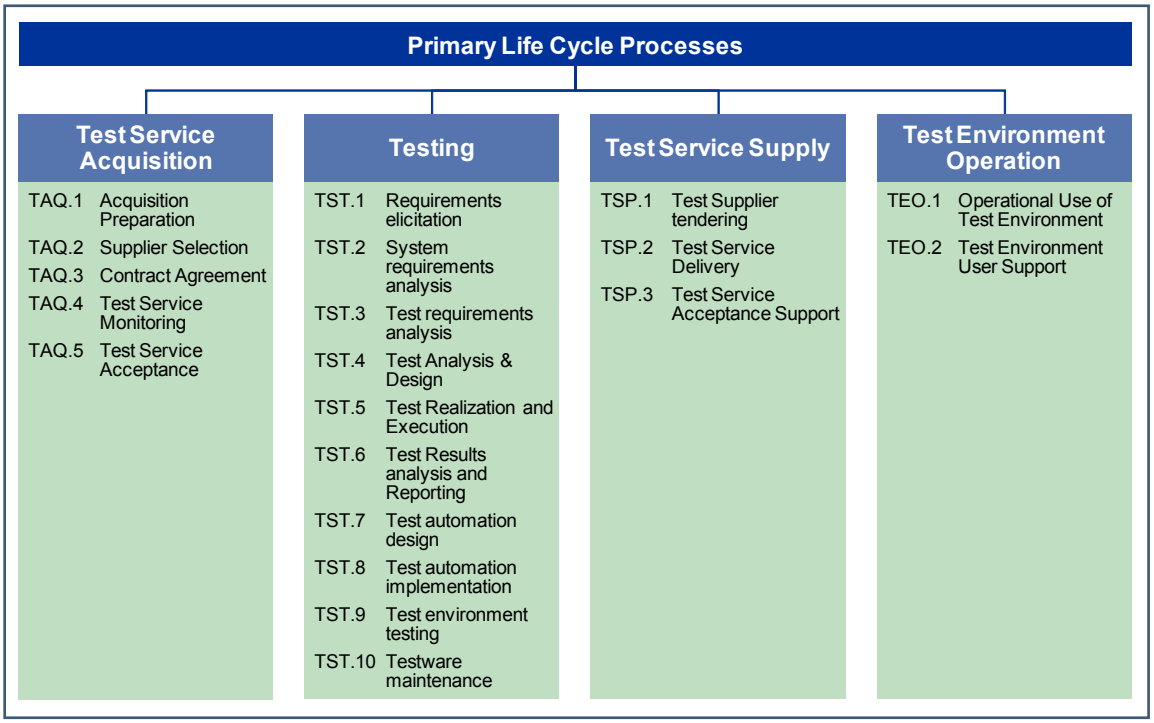


Fig. 7: The primary life cycle processes of Test SPICE

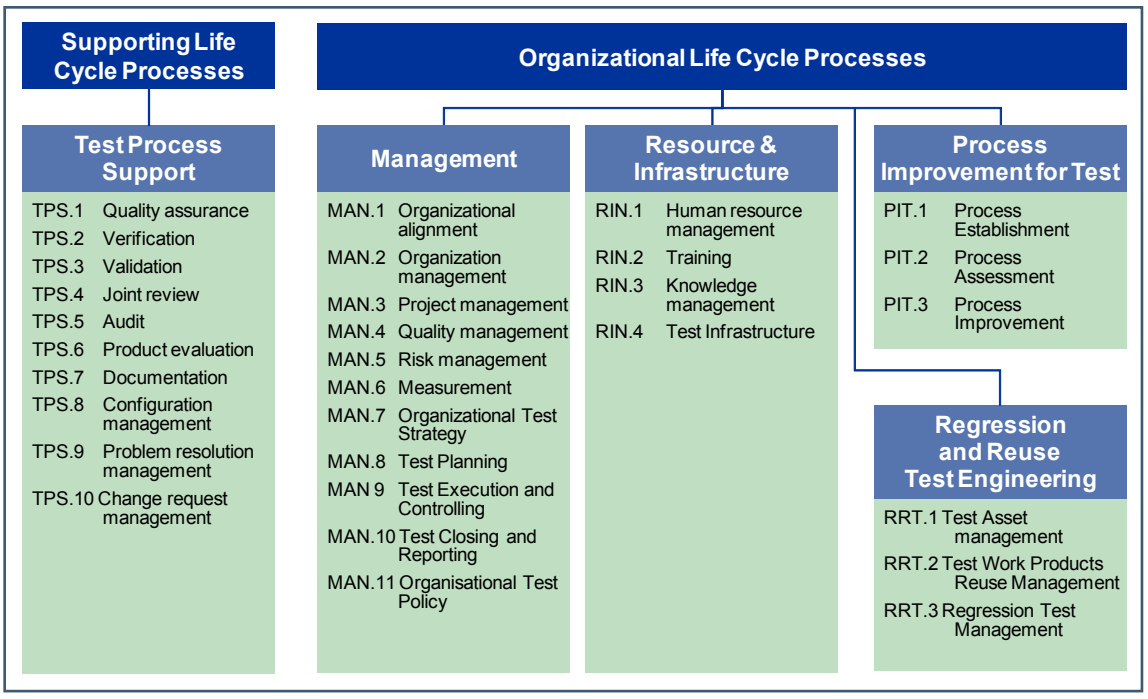


Fig. 8: The supporting and organizational life cycle processes of Test SPICE

The new model at a glance

The following figures show the overall content structure of the model:

This approach delivers some benefits. The structure of content and, based on this, usage is very easy for people experienced in assessments and process engineering based on the standard. The requirements content and indicators can be found and used as already used by the original standard. The second advantage is that Test SPICE is an amendment that focusses on an area of special interest (well suited for test organizations) but it can be easily integrated in a complete approach to stabilize and improve the process capability.

Mapping from ISO/IEC 15504-5 to Test SPICE

The following mappings show the original content of ISO/IEC 15504-5 to Test SPICE:

Mapping by Process Groups

ISO/IEC 15504-5	*Test SPICE*
Acquisition	Test Service Acquisition
Supply	Test Service Supply
Operation	Test Environment Operation
Engineering	Test
Support	Test Process Support
Management	Management
Resource and Infrastructure	Resource and Infrastructure
Process Improvement	Process Improvement for Test
Reuse	Regression and Reuse Test Engineering

Mapping by Processes for the Acquisition Process Group

ISO/IEC 15504-5: Acquisition	*Test SPICE: Test Service Acquisition (TAQ)*
Acquisition preparation	TAQ.1 Acquisition Preparation
Supplier selection	TAQ.2 Supplier Selection
Contract agreement	TAQ.3 Contract Agreement
Supplier monitoring	TAQ.4 Test Service Monitoring
Customer acceptance	TAQ.5 Test Service Acceptance

Mapping by Processes for the Supply Process Group

ISO/IEC 15504-5: Supply	*Test SPICE: Test Service Supply (TSP)*
Supplier tendering	TSP.1 Test Supplier Tendering
Product release	TSP.2 Test Service Delivery
Product acceptance support	TSP.3 Test Service Acceptance Support

Mapping by Process for the Operation Process Group

ISO/IEC 15504-5: Operation	*Test SPICE: Test Environment Operation (TEO)*
Operational use	TEO.1 Operational Use of Test Environment
Customer support	TEO.2 Test Environment User Support

Mapping by Process for the Engineering Process Group

ISO/IEC 15504-5: Engineering	*Test SPICE: Test (TST)*
Requirements elicitation	TST.1 Requirements elicitation
System requirements analysis	TST.2 System requirements analysis
System architecture design	TST.3 Test requirements analysis
Software requirements analysis	TST.4 Test analysis and design
Software design	TST.5 Test realization and execution
Software construction	TST.6 Test results analysis and reporting
Software integration	TST.7 Test automation design
Software testing	TST.8 Test automation implementation
System integration	TST.9 Test environment testing
System testing	TST.10 Testware maintenance
Software installation	
Software and system maintenance	

As the charts show, this is not a mapping in the sense that you can compare processes 1:1, but it helps you to understand the comparison between software engineering and software testing. Therefore both models use requirements elicitation as a starting point because requirements are crucial for software engineers and for software testers. For the same reason, both models contain a maintenance process.

Mapping by Process for the Management Process Group

ISO/IEC 15504-5: Management	*Test SPICE: Management (MAN)*
Organizational alignment	MAN.1 Organizational alignment
Organizational management	MAN.2 Organization management
Project management	MAN.3 Project management
Quality management	MAN.4 Quality management
Risk management	MAN.5 Risk management
Measurement	MAN.6 Measurement
	MAN.11 Organizational test strategy
	MAN.12 Test planning
	MAN 13 Test execution and controlling
	MAN.14 Test closing and reporting
	MAN.15 Organisational test policy

The design of this process group reflects that on the one hand there are standard processes in the management area and on the other hand there are specific processes to manage the test in the organization and in the projects. In contrast to ISTQB, Organizational Test Strategy and Organizational Test Policy are taken as processes (ISTQB: Work Products). Behind this more formal reason the design of the process group reflects the typical problems faced by test teams:

- Poor project estimation
- Poor time planning
- Abuse of planned test time as an undeclared time buffer for development activities
- Unrealistic goals
- Blaming the test team for slowing down the project speed.

The design of the process group makes it possible to look at the test management as well as at the project management to see not only the symptom (test is late) but also if the symptom is caused by the test management or by the project management.

Mapping by Processes for the Resource & Infrastructure Process Group

ISO/IEC 15504-5: Resource & Infrastructure	*Test SPICE: Resource & Infrastructure (RIN)*
Human resource management	RIN.1 Human resource management
Training	RIN.2 Training
Knowledge management	RIN.3 Knowledge management
Infrastructure	RIN.11 Test Infrastructure

In Test SPICE "Infrastructure" was changed to "Test Infrastructure" and was given a new ID to this process to make sure that assessors are looking for the right evidences.

Mapping by Processes for the Reuse Process Group

ISO/IEC 15504-5: Reuse	*Test SPICE: Regression and Reuse Test Engineering (RRT)*
Asset management	RRT.1 Test Asset Management
Reuse program management	RRT.2 Test Work Products Reuse Management
Domain engineering	RRT.3 Regression Test Management

Mapping by Processes for the Process Improvement Process Group

ISO/IEC 15504-5: Process Improvement	*Test SPICE: Process Improvement for Test (PIT)*
Process establishment	PIT.1 Process Establishment for Test
Process assessment	PIT.2 Process Assessment for Test
Process improvement	PIT.3 Process Improvement for Test

This group and the processes were renamed to ensure that only evidence relevant to the improvement of the test process is taken into account during an assessment.

Benefits of Test SPICE

Test SPICE is designed for all organizations which currently use a SPICE model that is documented in ISO/IEC 15504 or is currently driven from organizations outside the ISO, such as Automotive, Banking, Enterprise or Medi Spice, as well as for organizations that currently do not use a process assessment model.

Organizations that are already familiar with a SPICE model benefit from the coherent look and feel of the process reference model, the measurement framework and the assessment process. This makes it easy to integrate the assessment of software or system processes. These organizations will save money by using the Test SPICE model to add to their current SPICE assessments the assessment of specific test processes.

Organizations that are not familiar with SPICE or other process assessment models are provided with a complete process assessment model that covers all aspects of test process assessment.

Conclusion

If we consider ISO/IEC 15504 as an open standard for process assessment and improvement especially for the IT industry, then this open standard should also be applied to test processes. The first version of this model shows that this is achievable. As a benefit for the IT industry there is no longer a need to translate the results of proprietary models to the ISO/IEC 15504 measurement framework, thereby saving money for training (one measurement framework fits all), data collection and analysis.

References

[1] ISO/IEC15504-5 Information Technology - Process Assessment - Part 5: An exemplar Process Assessment Model

[2] ISTQB® Certified Tester - Foundation Level: Grundlagen des Software-Testens (SQS Software Quality Systems AG, Version 2.10.0, Seminarunterlagen)

[3] ISTQB® Certified Tester - Advanced Level Test Management (SQS Software Quality Systems AG, Version 1.4.1, Seminarunterlagen)

[4] Certified Tester - Foundation Level Syllabus (German Testing Board e.V. & Swiss Testing Board, German-language edition, 2007 version)

[5] Softwaretest - Lehrplan zum Aufbaukurs, ISTQB Certified Tester, Advanced Level (ASQF e. V., Version 1.2, July 2003)

[6] Test Maturity Model Integration (TMMi) (TMMi Foundation, Version 1.0, February 2008)

[7] TPI® Prüfkatalog (Übersicht TPI® Kontrollpunkte) (SOGETI)

[8] Standard glossary of terms used in Software Testing ('Glossary Working Party', International Software Testing Qualifications Board, Version 2.0, December 2007)

[9] SQS-Glossar - Glossar gemäß ISTQB Certified Tester (SQS Software Quality Systems AG, August 2006)

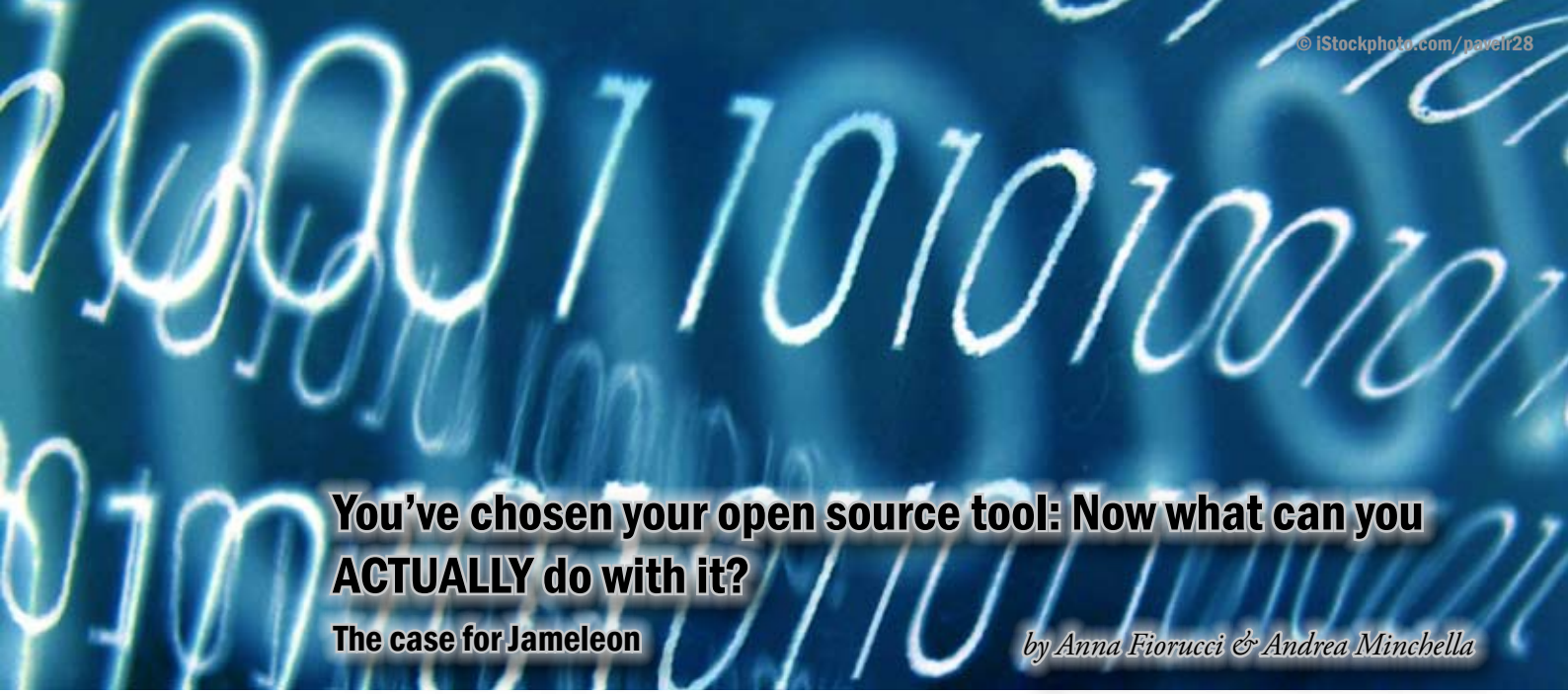
[10] Martin Pol, Tim Koomen, Andreas Spillner, Management und Optimierung des Testprozesses, Heidelberg, dpunkt, 2000

Biography

Monique Blaschke is a junior consultant at SQS Software Quality Systems AG. After studying she spent two two years with a leading telecommunication company as a technical project manager in different test projects for mobile communications before joining SQS in October 2008. Apart from another test project for a telecommunications distributor, her major project has been participation in the development of the test assessment approach Test SPICE.

Michael Philipp is a senior consultant at SQS Software Quality Systems AG. He has a broad experience of software quality management, software testing and process improvement, having been with SQS since 1998. He is particularly characterized by experience in international projects (testing, improvement, IT life cycle), conducting assessments (ISO 15504) and as a trainer for the SQS training program.

Tomas Schweigert is a principal consultant at SQS Software Quality Systems AG. He has a long experience of software quality management, software testing and process improvement (PI), having been with SQS since 1991. His special interest is the analysis of projects in crisis situations. Tomas Schweigert is now a SPICE principal assessor (INTACS scheme). His current research topics are the SPI Manager qualification and the Test SPICE test assessment approach.



You've chosen your open source tool: Now what can you ACTUALLY do with it?

The case for Jameleon

by Anna Fiorucci & Andrea Minchella

1. The project needs

In February 2007 we were asked to perform a feasibility study for selecting an open-source automated test tool. We were working as an integration and system test group for a project in which patches and requirements were released daily, if not twice a day, and the development team needed a regression test suite that they could run without support from us. As for ourselves, we needed a test suite that was easy and quick to maintain. Moreover, we did not want anything that would take much time from our regular activities, i.e. we needed a tool that could allow us to register the test scripts with virtually no additional effort compared to the manual run.

2. Why Jameleon: what we had (existing test environment, our skills)

We selected Jameleon as our tool for the following reasons:

- Jameleon is specifically designed for projects that use Java as a programming language (Jameleon is written in Java) and xml-files for configuration, data transmission etc. In Jameleon, test scripts are xml-files.
- Since it is a framework for automated tests, Jameleon offers the possibility to define test suites. Suites are useful both for regression testing and for testing a logical sequence of events (the real business case).
- Jameleon offers the possibility of extending its functionalities, i.e. the tags on the xml- page that makes up the script.
- Apart from its custom tags, Jameleon offers many plug-ins from other frameworks, such as JUnit, Jiffie, etc.
- Jameleon custom tags allow the user to connect to an Oracle database, and to extract data, right in the xml for the script. The framework allows configuring connections to the database. This will be shown in the next paragraph. Data pools can be also used by simply accessing a file.
- Our previous test environment was rather mature. We had a client that we used to insert input xml files into the system. We figured that we could expand Jameleon's tags with a new one that could simulate our client.
- In our team we had a tester who was also a proficient programmer, who knew the system to be tested, the environment, and who could understand how to extend Jameleon to provide a custom tag for our project.

Jameleon offers a framework that prints out test case specifications and the test report from the information in the xml script file. There is no need to produce additional documentation, because the one produced automatically is fit for most quality standards.

3. The plug into the configuration, how to work when the plug-in is ready: scope, post-conditions, configuration

3.1. Tag "online session" and maintenance of the environment

Since we already had a simulation environment, the only thing that was needed was extending the plug-in package with a tag that could connect us to the environment. This was done by extending the session and function classes. Once the new on-line-session was coded, it was added to the jameleon-core.properties file. See picture 1.

In order to allow the test team to use and maintain the plug-in to test environment without having to tamper the code, a new file was added to the properties. This file is a simple text file that contains the list of business functions that our system implements. When a new function is added, the tester simply adds its name to the list.

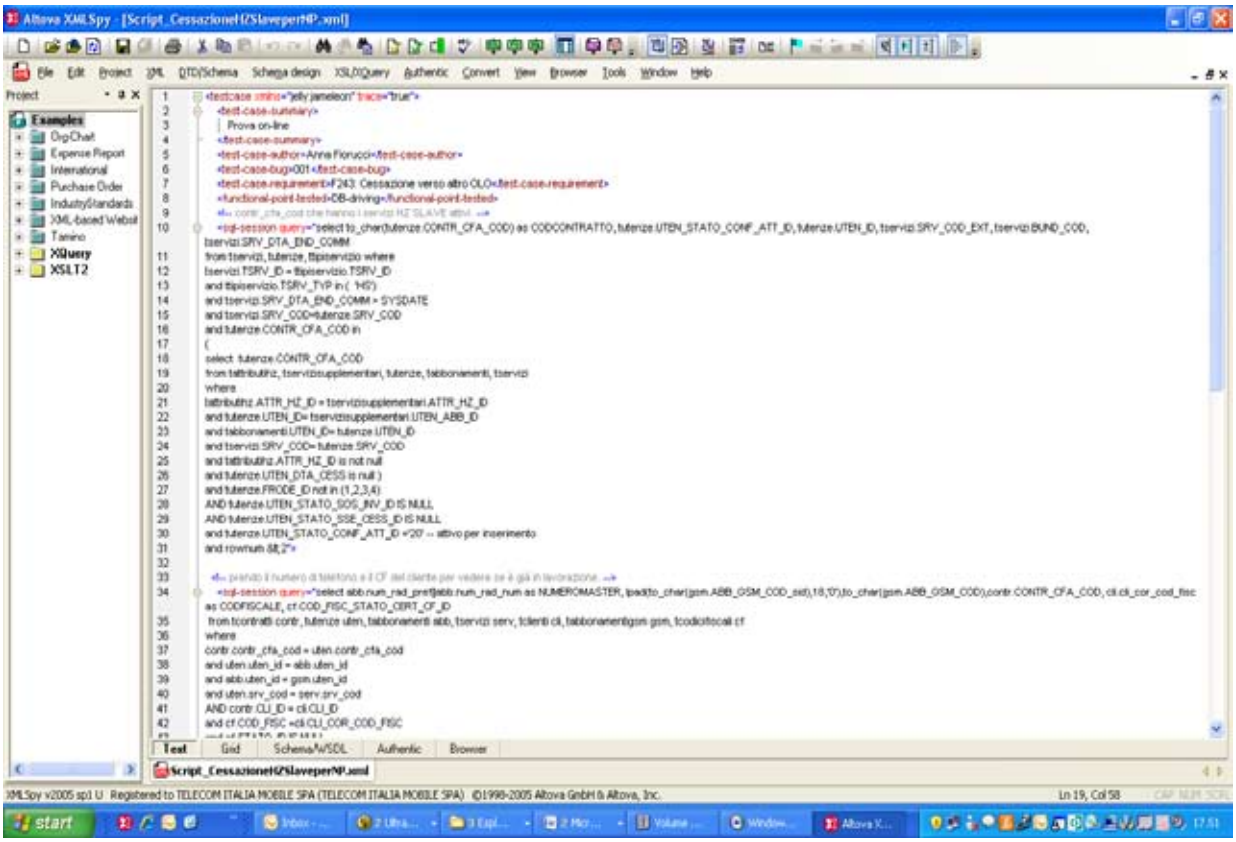
3.2. The scripts and the input files: solutions, scope issues and the quick fix

Once our custom tag was ready, we could implement our scripts. As for many other automated test frameworks, the test files needed are an input file template, a script file that is used to add real data to the input template and start the system function (and pass the input file to it, of course). The Jameleon framework, by using xml files, has all the scope features that xml provides. Therefore it is possible to perform a sequence of queries in order to select data that cannot be selected with just a few conditions. This was the typical case in our

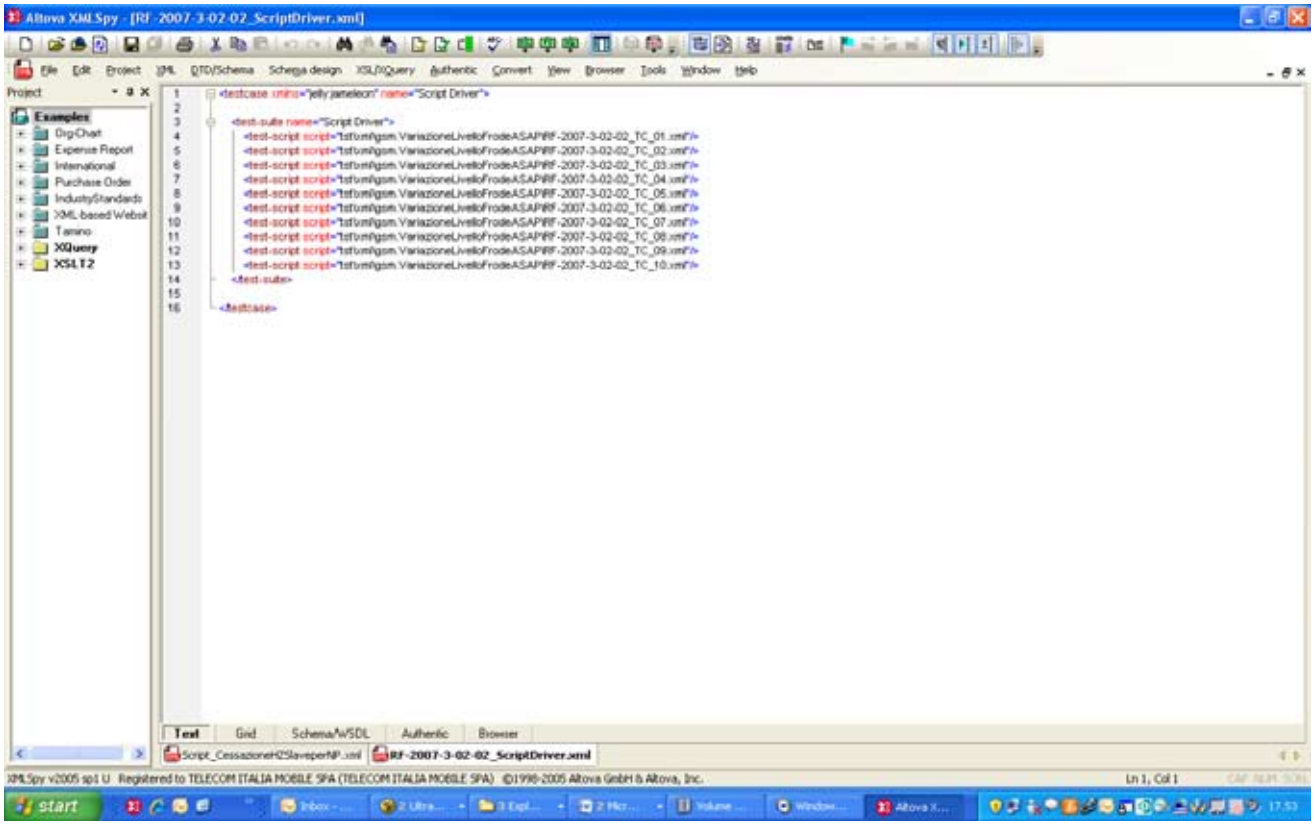
```
#Function Tag Definitions generated by Jameleon.
#Tue Nov 18 14:20:37 CET 2008
test-case-summary=net.sf.jameleon.TestCaseSummaryTag
online-session=net.sf.jameleon.plugin.maconline.OnlineSessionTag
test-case-requirement=net.sf.jameleon.TestCaseRequirementTag
test-suite=net.sf.jameleon.TestSuiteTag
iterate=net.sf.jameleon.data.IterateTag
wait=net.sf.jameleon.WaitTag
application-tested=net.sf.jameleon.ApplicationTestedTag
param=net.sf.jameleon.ParamTag
sql-param-value=net.sf.jameleon.sql.SqlParamValueTag
precondition=net.sf.jameleon.PreconditionTag
function-doc=net.sf.jameleon.function.FunctionDocTag
test-case-id=net.sf.jameleon.TestCaseIdTag
sql-update=net.sf.jameleon.sql.SqlUpdateTag
map-variable=net.sf.jameleon.VariableMappingTag
test-case-author=net.sf.jameleon.TestCaseAuthorTag
functional-point-tested=net.sf.jameleon.FunctionalPointTestedTag
param-value=net.sf.jameleon.ParamValueTag
sql-param=net.sf.jameleon.sql.SqlParamTag
http-session=net.sf.jameleon.plugin.jwebunit.HttpSessionTag
batch-session=net.sf.jameleon.plugin.macbatch.BatchSessionTag
sql=net.sf.jameleon.data.SqlTag
ju-assert-equals=net.sf.jameleon.plugin.junit.tags.AssertEqualsTag
param-type=net.sf.jameleon.ParamTypeTag
```

Picture 1: jameleon-core.properties file

project. Multiple queries are also needed to check post conditions. Typically, Jameleon performs the script only once, that means that even though the result set for the query is greater than 1, Jameleon will take the first available value, perform the script with that value, register the result and stop. However, we have noticed that in some cases, probably due to a lack of integrity, Jameleon can loop the script. If the test is not stopped (you can stop/abort a test within the Jameleon client), Jameleon might go on performing the test until there is no more data available! We did not have the time to inspect the issue, or to inspect the (very good!) Jameleon Forum, so we figured out a quick fix, by adding a global Boolean variable to the script. We then added the check to the query condition, i.e. the query is performed only if the variable is false, when the system function is invoked, the variable is set to true.



We also managed to implement a test suite. This was useful for testing the sequence of actions on a particular input. In the system we tested, for example, a client can buy a cell phone contract with additional services, buy additional ones or remove some. The customer can forget to pay a bill, his phone line can be blocked, then the bill gets paid and the line is restored. The test suite is easily managed with a file called the script driver.



3.3. Running the tests: test sessions, logs, etc

The Jameleon suite offers the possibility of recording your test sessions, you always get a test record when you run a test, even if, for any reason, you must stop the test before the script is done. This may be useful even when the test case is aborted. Another feature that was implemented in the on-line-session tag was the automatic extraction of log entries. Our test framework, in fact, allows us to log entries connected to the service "tid" (transaction identification code). When the test is executed manually, logs can be extracted using the tid code with a Unix command. The on-line session automatically extracts the transaction logs. By adding a Local-log-extraction property, logs can be sent to the local tester computer, when the property is true.

3.4. Open-source version management: subversion, eclipse

Since test scripts are created, used and modified by more than one test engineer, it is good practice to use a configuration management tool. Since our project was to be completely open-source, we used Eclipse with a subversion plug-in. Eclipse turned out perfect for managing the various types of files that the project had, and the executable file could always be launched from the Eclipse framework with updated test script. When we had something stable, i.e. new requirement did not affect extraction logic in the script, we could develop a compiled suite that the developers used for their own regression tests.

4. Costs

As mentioned briefly in the introduction, this automation project was not supposed to interfere with normal testing activities. That means that no budget entry was created for it, and it can be considered a "moonlight project", at least as far as the initial part of development was concerned. As already explained, we had a proficient programmer in our group, who in ca. 20 man-days managed to develop the new custom tag and all the configuration files. Developing test-scripts and input xml files took as much time as was needed for the launch of the first automatic run. We mean here the study of the kind of data to be extracted from the database (resulting in the data mining query for the script), the study of the effect on the system and its database (resulting in the check query and control assertion for exit in the data script).

5. Evaluation

We selected Jameleon for its flexibility, and we were expecting some difficulties in the customization. Luckily, developing the custom tag proved much easier than expected. Writing the scripts proved itself also quite natural. In the end, we were able to present our management with a prototype two months ahead of what we had expected. Jameleon could help us with virtually all the test we had hoped we would use it for.

Biography

Anna Fiorucci received her MSc in Computer Science in Pisa. She has been working in QA, Validation and Verification activities for 10 years. She has extensive experience in validating and verifying safety-critical systems, complying with strict quality assurance methods. At the moment she is working as team leader of a QA team for a web-services integration project.

Andrea Minchella received his MSc in Electronical Engineering in Rome, specializing in telecommunication. He has been working in the testing business for 6 years, and he has worked in development and execution of functional, architectural and performance test. He has extensive experience in the automatic testing of large software applications. At the moment, he is working as automation expert for a web-services integration project.

Test Automation: Salutations to the World

by David Harrison

In a previous article [1], the role of test automation in a software development project context was set out. In particular the success of test automation on Java-Swing projects was described.

Whilst this article hopefully gave you, the reader, a basic grasp of this challenging topic, which is often so poorly described in the testing and quality assurance literature, it left open the low-level details of how the industrial-strength approach described was architected. It is the intention of this article to demonstrate, via a practical example, highly relevant to the test automation domain, together with it's companion Part 2 article, the key elements of the pattern of solution. It should be noted that, as set out in [1], the approach to test automation is essentially a programmatic one, and the reader needs to be open to this aspect. Once this fact is grasped, the means to obtain efficiency, scalability and effectiveness of a solution is basically the same as that for any other software development task. The unique example application, showcased in this article, Hello World!, was crafted by Peter Walser who specialises in the construction of advanced applications using Java-Swing and Eclipse RCP technology. Peter can be contacted on pwalser@frostnova.ch.

Say Hello to the World

Our example Java-Swing application is a test automation version of the ubiquitous Hello World! example – after startup it displays a textual greeting to the planet Earth, see Figure 1, below:

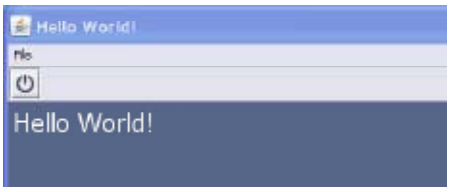


Figure 1 – Hello World!

However, to get this friendly, earthly greeting, we must first login immediately at application startup. This small piece of functionality is extremely common in industrial software and happens to be the, almost universal, example used in the automated testing literature when authors describe their approach to automation (e.g. [3]-[6]). So for these two reasons, it's a very useful feature to have in our example application. The login panel is as shown below:

Figure 2 – Login Panel

This application is designed to allow the user to obtain friendly earthly salutations. In particular we will login, initially, using the credentials "test/test123", in which case we see the eventual result as shown in Figure 1, the main application panel shows the greetings "Hello World!". However, the twist to this login story is that with alternative credentials "pluto/pluto123", we get a completely different message.

Getting different outcome on the UI of a software application depending upon the login credentials is a very common situation. For example, we might see the following:

- main panel title embellished with the logged-in user name
- a status message reflecting the logged-in user name
- other screen embellishments that reflect the access rights of the user

So this example embodies these issues.

Test Structure

In essence, then, our automated test must have the following structure, expressed below as pseudocode:

```
Start application
Wait for application to become
established
Wait for the Login panel to ap-
pear
Validate Login Panel
Enter user-name - "test"
Enter user-password - "test123"
Press Login Button
Wait for login panel to be dis-
missed
Wait for main application panel
to appear
Validate the UI elements of the
Main panel
Terminate the application
```

The structure shown here contains two Static Validation Test elements – validation of the Login panel and validation of the Main panel. This Static Validation Test concept is one explained in more detail in [1] [2]. Suffice it to say, that in this example the lack of real "business" Use Cases means that these forms of test represent the limit of what is possible.

The Automation Challenges

The above test case is deceptively simple. Why is that?

Well first of all our test, as well as starting the target application, the SUT, requires that we wait for the application to become established, in this case, within the context of the JVM, and then that we wait for the login panel to appear. These two steps require the test automation tool to have an intimate knowledge of when objects, the login and application panels, become fully instantiated as well having the capability of raising an event to signal this important outcome. In addition, we have the companion issues of intelligently tracking when objects are dismissed, essential in order to proceed to following test steps with the application in a known state. These issues, captured well in this seemingly simple example,

are extremely well handled by QF-Test[1], and this is the automation tool we use here. The singular attributes of this tool in the Java-Swing area have been noted elsewhere [7].

Let's see how we meet, and overcome, these challenges…

The Test Project

Within the QF-Test tool, the test structure, the test project, is organized as shown below:

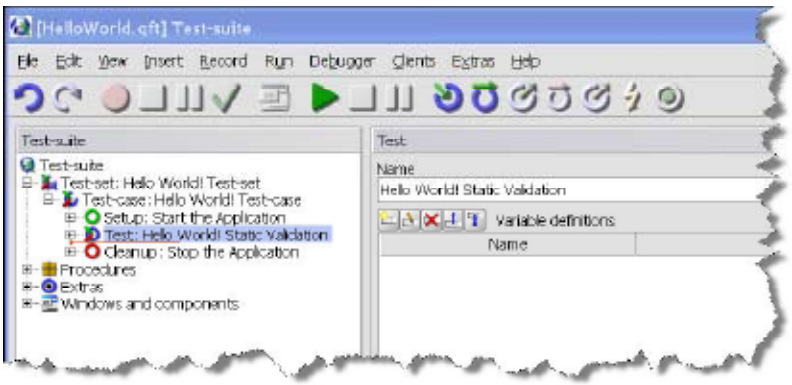


Figure 3 – Test Project

Here you can see that the test, the `Test-case`, is somewhat similar to that which would be recognizable to a developer involved in Unit testing. Within the `Test-case` we have a `Setup` element, then a `Test` element followed by a `Cleanup` element. As with Unit testing, the strategy here is that *before* each `Test`, the `Setup` action is executed and *after* each `Test` the `Cleanup` is executed. This is a *very* good structure for test automation, since prior to each test we start the application and following each test (successful or not) the application is terminated. Hence, each Test is executed with the application in a known and clean state. This is a fundamental strength of our pattern of solution.

The Detail

Let's take a look in more detail at the parts of our Test project (Figure 3).

The Setup Element

The `Setup` element starts our Hello World! application via a specified batch file. The mechanism here follows the documented approach from QFS. Expanding the `Setup` node in the project structure, the content is as shown in Figure 4 below:

Figure 4 – Setup Element

As can be seen, we "call" a Procedure to start our application, with the parameters specifying the name and location of the appropriate batch file (using Jython conventions for variables, in this case specified globally). Why do we use this approach? Well, proceeding like this means that we gain the benefit of reusability of the test "code" we write – the Procedure neatly encapsulates everything needed to securely start any application via a batch file, and thus can be re-used in any testing project.

Within the Start Procedure we use a built-in QF-Test Start SUT node. Once the application is started, a login panel appears and our test design must take account of this. Let's look at what this Procedure does in pseudocode form:

```
Try
        Use Start SUT client
node using supplied parameters
        Use Wait for client node
        Call Procedure to per-
form the login step
        Return true
Catch ClientNotConnected
        Return false
Catch DuplicateClientException
        Return true
```

As you can see the structure is fairly simple, the task of starting the target application is broken down into a number of steps; one that calls a QF-Test Start SUT function which literally starts the executable via its batch file, one that waits for the application to announce that it is fully started, and yet another which then deals with the details of the login task. Here, we may base the design on the strong exception handling capability of QF-Test. The pseudocode above is expressed in the project as shown in Fig 5-.

The procedure that logs into the application with specific (globally defined) credentials is as shown in Figure 6.

In this Procedure, we wait for the login dialog to become *fully established* and then enter the credentials. It should be noted that we take account of any (*unexpected*) blocking Modal Dialogs that arise e.g. user name not valid, password not valid etc.

It should also be noted that we take account of the special characteristics of the `JPasswordField`, when writing the password text by using generic library calls.

The Test Element

The intention of our test is to perform a Static Validation [1] of our main panel. This form of test is a *very important* one in the test automation domain. Why is this? Well, prior to performing any sort of business-related workflow test, we *must* ensure that the state of the application is as expected. In particular, we will need to verify elements such as:

- the main menu – both structure and enablement
- the toolbar – both button enablement as well as tooltip text
- main panel decorations – such things as dates, title text, memory monitoring elements and any user-specific elements

In this example, for simplicity we restrict ourselves to validation of the message displayed and the main panel title.

(See Fig 7)

Some important observations can be made about our Test element:

- we use Procedures to encapsulate the logic of main panel validation (`Test-case Support.ValidateMainPanelText`)
- we use a Procedure to validate the (`JLabel`) textual greeting (`Utility.Label.CheckText`), one parameter of which specifies the expected greetings
- as well as catching exceptions (`UserException`, `UnexpectedClientException`), we throw them too – specifically to signal that the test cannot continue (`UserException`) – a very clean and elegant approach which leaves our test uncluttered by tedious logic, thus improving readability and maintainability.
- we use QF-Test scripts to perform logging

The Cleanup Element

In this element of our test, we simply "kill" the application (it may be in a bad state, any menus may not be working). Figure 8 shows the details of the Stop Procedure, called from our Cleanup element.

Where Are We…

In this relatively simple example of test automation , a number of important themes have been touched upon. These can be summarized as follows:

- using QF-Test we have a genuine capa-

1 www.qfs.de

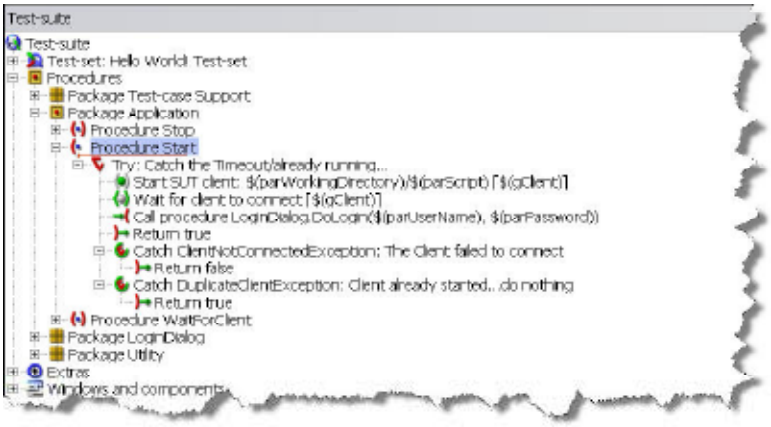


Figure 5 – Start Procedure

Figure 6 – LoginDialog.DoLogin Procedure

Figure 7 – Test Element

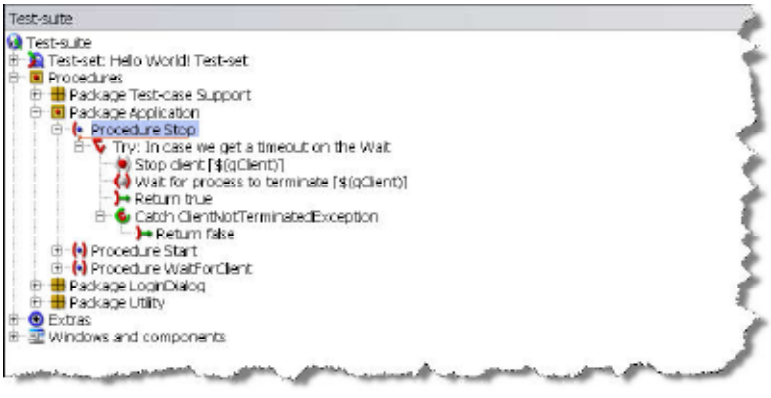


Figure 8 – Stop Element

bility to *design* our tests

- the use of Procedures delivers to us as test automators, the same benefits that they do to the general software development community
- the availability of a first-class exception architecture provides crucial benefits in responding to the SUT state
- although only hinted at in this example, we have the capability of developing a generic automation library of Procedures which allow us to interact with Java-Swing objects such as `JLabel`, `JComboBox`, `JPasswordField`, `JTable` etc. This is a crucial observation for real-world, project-based automation efforts.

Say Hello to Pluto

In real-world projects the automator is very often faced with software that, to one extent or other, contains UI elements that change in relation to such things as:

- The currently "logged in" user
- The current date/time
- The currently open file/document within the application
- The current "pending edit" state
- The machine on which the application is running
- The current number of elements in a collection, e.g. number of open documents, number of results found by searching

A good example of this variation is the way in which most applications which deal with documents, embellish the main application frame with a title that carries the current document title as well as a mark, often '*', that indicates that there are pending edits. When QF-Test responds to object creation, the *default component naming mechanism* takes these specializations into account. So, for example, if we attempt to automate a Search functionality for which the Search panel carries a result `JLabel` like:

```
Hits = 120
```

to inform the user that the Search resulted in 120 result elements, then this represents a case for which the *default* Name Resolving process will produce different values of component ID as differing result counts are found through searching. In such a case, if we recorded the `JLabel` component, we might see a default component ID like:

```
Label_Hits_=_120
```

But in the case where we see an outcome involving 1200 result elements, we would see:

```
Label_Hits_=_1200
```

Given this situation, if we develop our tests using the former component ID, then when we want to address the count `JLabel` again for the situation with a different result count, *the component will not be found*. On first reflection, this situation may seem somewhat catastrophic for successful test automation. Well, QF-Test provides us with a very *elegant*, *practical* and *powerful* solution to this problem – the Name Resolver API.

As a simple example of what all this means, let's revisit Hello World!, our good friend from earlier discussions. By design, with this application we can login as a different user, other than "test/test123", the credentials we used earlier, and see a different set of important visual outcomes. In particular, we now say hello to a different planet! Let's see how this works…

Saying Hello Again

The alternative set of credentials we can use for Hello World! is "pluto/pluto123". In real situations, the application being automated may have a large set of credentials that are permissible when it comes to logging in. This "unboundedness" (for all practical considerations), if it leads to variations in the visual elements of the application, should be the alarm bell that we must enhance the name resolving process.

If we take a look in the "Windows and components" section of the Hello World! project, and expand the tree view, then the `JLabel` component (when we logged in with "test/test123") can be located, as shown below:

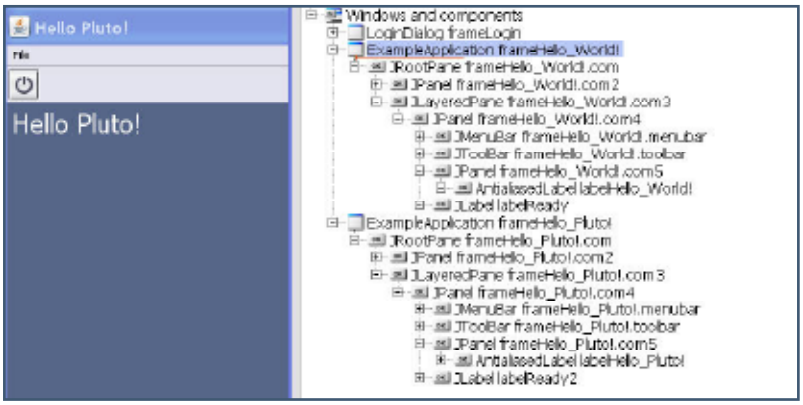


Figure 9 – Hello World Label

In this figure we can see that the main panel of Hello World!, as well as the relevant Main Window JLabel component are highlighted. The component ID of the JLabel is

```
labelHello_World!
```

just the form we described earlier. In addition, we can see that the main application frame is also decorated with the specialized text, e.g. frameHello_World!. In addition, all the components below the main application frame are given names that involve the specialized text. If we now look at the situation when we use the "pluto/pluto123" credentials, and re-record this entire main application frame [8], then the situation as shown below emerges:

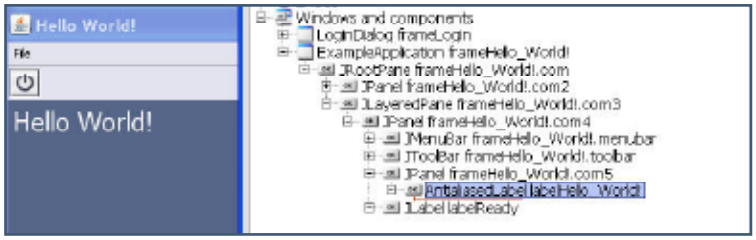


Figure 10 – Hello Pluto! Components

So now we have our "Windows and components" section containing two top-level elements, essentially the same components, but having different names/IDs, only brought about because we have logged in with different credentials. As an alternative strategy to extending the basic QF-Test Name Resolver, we could get the developers to do setName() on the UI controls. When this is done, such controls do not have their name overridden by QF-Test.

This is a great strategy to solve this component naming problem. However, typically on projects, the test automation workstream gets active only later in the overall project life cycle, and this makes sense as the software, particularly in relation to its UI, is becoming stable. At this point, developers are often heavily involved in getting the business functionality integrated, for example, and asking them to go through the code and do a setName() on controls is probably not a practical proposition. Having a strategy for resolving this naming problem that is in the hands of the QA workstream, then, has definite appeal.

Let's set about resolving this conundrum using the Name Resolver extension API within QF-Test.

The two components for which we want to provide a new, neutral name are the main application window (Class: idx.loginapp.ExampleApplication) and the JLabel that ends up being displayed on the main application window area (Class: idx.gui.text.AntialiasedLabel), and you see below how a test is made in our Name Resolver Jython script, using methods of the Resolver Registry object, to determine if the class type of the current object is one of those in which we are interested.

(See Fig 11)

This Name Resolver script is "installed" as part of our Setup phase of each test [2].

In the case where we detect that we have an object of the main application class type, then we return the neutral name "APPLICATION_FRAME", and if the current component is of the label class and if the text property has the prefix "Hello " and the postfix of "!", then we return the neutral name "Hello_PLANET".

Once this Name Reseolver script is installed [8], then as these components are brought into existence within the application, the component Name/ID will be assigned these neutral names. When we record these specific UI objects [8], we will see these names/IDs instead of the default ones assigned by QF-Test.

Now we can, with confidence, develop our automated tests using these neutral names.

The importance of this Name Resolving approach cannot be overstated when considering its effectiveness at dealing with this "component naming" issue – one that bedevils test automation generally.

Now when we login with any alternative credentials, all is well - the elements we need to find can be found and their properties asserted.

Now that's a pretty good outcome for test automation…

Summary

Hopefully, in this article, a number of the crucial steps along the road to successful Java-Swing test automation have been illuminated. The issues highlighted, as well as the tool attributes which help us meet the challenges, cannot be overstated in their importance to a successful project-based approach. Good luck…

References

[1] Project-based Test Automation, Testing Experience, June 2009, www.testingexperience.com

[2] "Automated Functional Testing for Java-Swing; A Pattern of Solution", David Harrison, 2009, ISBN (978-1-4092-9068-1) www.LuLu.com

[3] A Simplified Automation Solution, Using WATIJ, Steven Troy et al, Testing Experience, December 2008, www.testingexperience.com

[4] The Record & Playback Fairytale, Koen Wellens, Testing Experience, December 2008, www.testingexperience.com

[5] Model-based Test Development and Automation, Claus Gittinger, Testing Experience, December 2008, www.testingexperience.com

```
#############################################################################
# Class: NRHelloWorld
# Author : D.Harrison
# Purpose: This class is the name resolver for the HelloWorld application
#
#############################################################################
class HelloWorldResolver(NameResolver):
    #constructor
    def __init__(self):
        self.data = []

#############################################################################
# Method: getName
# Author: David Harrison
# Purpose: overwrites the built-in Name Resolver.
#          If we return a non-None name from here, then the built-in Name Resolver will preserve the assigned name (but replacing WS).
# Parameter:
#       com ... the component to be examined and possibly named
# Returns
#    a replacement name
#############################################################################

    def getName(self,com):
        try:

            if not ResolverRegistry.getElementName( com ):
                #  ---------------------------------------------------------- Application Specic classes
                if ResolverRegistry.isInstance(com, "idx.loginapp.ExampleApplication" ):
                    return "APPLICATION_FRAME"

                elif ResolverRegistry.isInstance(com, "idx.gui.text.AntialiasedLabel"):
                    s = com.getText()
                    if  s.strip().startswith( "Hello " ) == 1:
                        if  s.strip().endswith( "!" ) == 1:
                            return "Hello_PLANET"
                #  ---------------------------------------------------------- Base Java-Swing classes
```

Figure 11 – Hello World! Name Resolver

[6] Robot Framework for Test Automation, Marcin Michalak et al, Testing Experience, December 2008, www.testingexperience.com

[7] Boon and Bane of GUI Test Automation, Mark Michaelis, Testing Experience, December 2008, www.testingexperience.com

[8] QFTest Documentation

Biography

David Harrison works as an independent software QA/Test Manager – currently at SwissRe, Zurich, Switzerland within the tools development group. This group has the mandate to develop and deploy globally reinsurance costing tools to the actuary and underwriter desktop. David can be contacted via email at dharrison_ch@yahoo.co.uk, or via his web site www.dexters-defect-dungeon.com

This article is based on material from his book: “Automated Software Testing for Java-Swing; A Pattern of Solution”, 2009, ISBN (978-1-4092-9068-1) www.LuLu.com

5 reasons why your Agile Testing project will fail

by Rajesh Mathur

In the last couple of months I have had many discussions, and many more arguments about the scope of Agile Testing in projects and the success rate of such projects. From most of these discussions I got the feeling that managers try to go for Agile testing with the half knowledge they have gained from some not-so-authentic websites or on the basis of whatever information they have learned from others.

I have attempted to observe and explore the five top reasons for failure in some Agile Testing projects and the ways in which you can tackle these roadblocks.

1. Implementing Agile via traditional methodologies:

The biggest and most important reason for failure that I have found is that people try to implement Agile Testing with traditional methods in their minds. They follow a waterfall, keep haphazard documentation, perform ad-hoc testing, listen to the project manager and fail in their testing. I have seen people who told me that Agile Testing is nothing other than ad-hoc testing and that they did not have to do any documentation. Well, I hope I know the reason why they failed.

Further, there are Test Managers who believe in no-strategy and no-plan for their Agile project. Again, they are mistaken that an Agile project does not require any strategy or plan. One must plan for an Agile Testing project. Extreme Programming projects, for example, keep their planning, monitoring & tracking simple so that next steps can be defined and project completion can be forecast. The planning may include test-driven development or a test-first approach, automated acceptance testing, testing early and iteratively, testing often and whenever required, the use of methods like exploratory testing, and finally following a context-driven approach to testing.

2. There are separate development & testing teams:

...and they are rivals, as they are in traditional software development projects.

…and testers should not spare developers by finding and reporting each & every defect.

..and testers should ask for all the details from developers to test the builds, or should otherwise tell them that the entry criteria for testing have not been met.

With these assumptions you cannot be sure of success in your Agile project because agility demands a high level of collaboration and integration between developers and the testers. In some Agile projects you may not find any testers, as the development team prepares the automated unit tests and the client is responsible for the acceptance tests. As much of the testing is done before and during the construction phase, less testing is required in the end phase.

Another important point that our traditional, individualistic managers tend to forget is that Agile is best performed in pairs. Whether it is pair programming or pair testing; agile groups work in pairs with the single aim of delivering a quality product faster with their simple designs.

3. All we have to do is ad-hoc testing:

I was talking to one of the test leads and he told me about his projects. When I asked about the test strategy he was using for the project, he told me that it was an Agile project and all they have to do is ad-hoc testing.

How mistaken he was! I did not dare to explain the methodology to him, as he was convinced of what he and his project manager were doing. It was not just him who thought that Agile testing is just performing ad-hoc testing. I have seen many experienced managers talking along these lines.

So what kind of tests do we do in Agile Testing projects? In his article in Dr.Dobb's journal, Scott Ambler suggested many testing types that can be performed throughout the Agile life cycle. Some of these are confirmatory testing, investigative testing, system testing and acceptance testing. We should not forget that automated unit testing that is an integral part of any Agile Testing project.

One may refer to "Testing Extreme Programming" authored by Lisa Crispin and Tip House to become acquainted with testing in extreme programming environments.

4. It does not matter who works in my team:

Here I am talking about the (over)-confident managers, who claim that they can get the work done by anyone.

Indeed, Agile projects involve pairing up and integration, and they require team spirit from all team members. That is, your team members should not only be technically very sound, they should be socially sound as well. An Agile Testing project will fail if your team members are not socially communicative and have poor social skills.

Further, communication also plays an impor-

tant role in Agile projects, as you do not have to deal with heavy documentation as you do in traditional development methodologies. Here, all you have to do is communicate, complete & deliver. Some Extreme Programming experts discourage having people scattered across different floors, cities or countries. They believe that face-to-face communication is highly important to make the project successful.

5. All I need is proper requirement documents:

First of all, you are not delivering your project through some traditional development method, where all you need are bundles of written documentation of customer requirements thrown over the wall for the testing team to write their test plans and scripts.

Agile projects are different. They do not get you loads of requirement documents. You have an on-site customer who has his acceptance tests ready and is clear about his requirements. He is available to give feedback and clarify acceptance criteria, so that frequently delivered pieces of software remain in sync with what he wants.

As I said earlier, one of the fundamental principles of Agile projects is that they need collaboration and integration. And this does not spare the customer too, because he is the one driving the project and defining the requirements. If your customer is bad, you will not be able to deliver good software.

Basically, any project will fail if you do not follow the basic principles of the methodology you are using. All flavors of Agile emphasize collaboration, communication, integration, simplicity, respect, feedback and the ability to adapt to changing business requirements. If you miss any of these, your project may fail. The reasons for failure that I have described in this article are not the only ones, there may be more reasons that can cause you to fail. However, my experience is that by avoiding even these ones, you can save your project.

Biography

Rajesh is a test management professional with more than 12 years of comprehensive experience in Project Test Management, Information Technology, Software Testing and Quality Assurance. He is a Bachelor of Science, Diploma in Computer Applications, MCSD, ISEB/ISTQB Certified and a Six-Sigma Green Belt.

He has extensive experience in test management & testing of Web-based products, Client-Server applications and Mobile Applications with domain expertise in Auto & Medical Insurance, Mobile and E-learning. He has worked with companies like Steria, Nokia, NIIT & Infogain serving clients in multiple geographies. Currently, Rajesh is working with Steria India Limited as a Test Manager.

ISSECO®

Certified Professional for Secure Software Engineering

TRAINING

There is no doubt that application security is a vital issue for any modern application. Unfortunately there is no plug that you can press to turn it on. Application security should be implemented throughout the product life-cycle.

The ISSECO course will teach you the how and when. The course will introduce the technology but never the less the methodology. Furthermore you will learn the security logic and how it can be achieved without breaking the budget.

Contents

- View Of The Attacker
- Explain The Definition Of The Terms Hacker, Cracker And Ethical Hacker
- Show Examples Of Famous Hackers And Their Attacks
- Point Out The Different Skill Levels
- Modes Of Hacking
- Hacker Motive
- Illustrate The Process Of Hacking
- Outline Common Hacking Tools
- View Of The Customer
- Explain Why Customers Expect Secure Software
- Trust & Threat Models
- Methodologies
- Requirements Engineering
- Secure Design, Secure Coding, Secure Testing, Secure Deployment
- Hands-On Workshop
- Security Response
- Explain The Difficulties Of Fixing Security Issues Via Standard Maintenance Processes
- Code & Resource Protection

Tutor: Manu Cohen-Yashar

01.03.10-03.03.10 **in Berlin, Germany** **Price: 1899,00 € (plus VAT)**

Register at http://training.diazhilterscheid.com

ISSECO® SECURE SOFTWARE ENGINEERING

SELA GROUP

Díaz Hilterscheid

Optimizing our Regression – Are we Ready?

by Alon Linetzki

"Our regression execution effort takes too long..." – is a sentence I hear much too often from clients when trying to assist them in optimizing their regression pool. It is a real challenge to reduce regression testing, as the whole aim of this major effort is not to find any defects while covering most of the product's functionality. From my experience, companies spend around 40-60% of their test execution effort on regression testing, in order to make sure that their client's experience will stay the same as the last release was, and even better.

But, we all know the pesticide paradox, which states that our systems are getting more and more resilient to our regression test cases, and so as test managers we may ask ourselves why we should invest such a huge effort when the chances or probability for having defects found are getting closer to none? Understanding this dilemma, I have come to the conclusion that something must be done in order to make our regression test execution more effective and efficient.

This article will describe the basic first steps on the road to having a better regression test execution and optimization of the regression pool.

What is regression testing?

Regression testing is a selective retesting approach of the system or product with the objective of assuring that the bug fixes work correctly and that those defect fixes have not caused any undesirable effect(s) on other parts of the system which have not been changed.

Regression testing also uses a selective set of tests from the regression test cases pool, and executes these to assure that the new system does not unintentionally cause the creation of new defects in the part that were not touched in the new release, thus hurting the current clients.

The first relates to running tests in the same release, the second is about running tests that relate to the previous release.

Motivation for optimization

1. Each new release of the system or product could potentially cause existing features to fail.
2. Clients rely on the fact that the current installed system or product will work properly (at least as well as the one installed at their servers).
3. A failure in the new system or product release would lower the confidence of clients in the system or product, and would harm the branding company.
4. At the end of a release, or big cycle, regression tests are carried out to make sure that no defects were introduced through the fixes. Such defects can degrade the system quality and customer experience.

A few factors influence regression testing along the life cycle, and if we take care while performing those activities, we shall have better chances to optimize our regression test cases pool:

1. Test planning aspects exercised
2. Test design-techniques used, coverage investigated
3. Test case management approach
4. Test assets maintenance

If we invest a lot in regression test execution, this can come up to 12-15% of the whole system/product development budget; so we must make sure that we perform proactively to optimize that pool of test cases.

Process (and other) assumptions

Before we can start optimizing our regression pool, certain characteristics of that regression pool and of our process must be in place. Mainly it is to 'guard' our regression pool from being 'damaged' in the process of being built. Those preventive actions must be carried out during the activities mentioned before throughout the life cycle.

In order to make sure that I will be able to optimize the regression pool wisely and effectively, I will make the assumptions described below. While these are essential for that purpose, most organizations have not yet realized nor proactively worked to put these in place. This is why they are struggling to optimize their biggest effort of regression testing.

Assumption – 1

Regression test cases do not normally need to test bound, invalid data, etc. – normally they will be designed and focused to test the system and how it was designed.

It is assumed that the above were tested prior to the regression test effort in previous releases.

Note: A good system test execution phase, according to some industry statistics, may reach only 25% code coverage, and still be very robust.

Assumption – 2

Test cases of each release are designed to cover most (if not all) of the system or product functionality. Even if we perform risk-based testing, that criteria is still part of our goals.

Non-functional testing is also conducted to the maximum coverage if possible and whenever relevant.

Hence, the regression pool of test cases has the potential of covering the whole product capabilities.

Assumption – 3

The regression pool of test cases has minimal scenario redundancy.

This is being kept across the different versions that went into production in the past.

Note: According to some industry statistics, the redundancy of test cases within the regression test cases pool may reach 10-30%. This is mostly a waste of execution time; I refer to 'ghosts' inside the regression pool.

Assumption – 4

Test cases for a release are managed according to risks, and a risk index is attached to each test case that was planned to run in order to cover a certain area of functionality of the system or product.

Hence, regression test cases have the same risk index attached as in previous releases.

Note: Risk-based test management has the potential of reducing test cases by 30%, while substantially increasing the DDP and added value to the customers.

Regression test approaches

There are a few regression test execution approaches which are straightforward:

1. Run all test cases from the previous release again.
    a. Applicable to some systems or products, mainly for safety-critical ones, whenever strict regulations and standards must be applied (GE Health use such an approach on some of their products, and also some other companies).
2. Run most of the regression test case pool.
    a. Happens in companies that have automated most of their regression pool, and probably most of their test cases
3. Run a subset of the regression pool.
    a. This is the most common approach.
    b. It is mostly derived from the risks attached to certain areas of the system or product.

The impact of regression testing is shown in the figure below.

We may use other aspects to determine which test cases from the regression pool will be run: size of the changes, area of the changes, complexity of the changes, past bug logs.

Conclusion so far: There is no single great regression test approach, and those that exist are context-dependent.

But we can learn from them and find out what the decision-making process is based on. Understanding that will give use the ability to enhance and leverage our decision- making process, and optimize our regression pool.

Regression test strategy

We may classify regression testing on any dimension:

- Regression at any test level: unit, integration, system, acceptance, etc.
- Regression for any test type: functionality, usability, performance, reliability, etc.
- Regression on any configuration supported by our system or product.

Investigating these strategies will have the same characteristics and assumptions that must be kept in order to maintain a 'clean' regression pool of test cases, and must be integrated into the process throughout the life cycle.

Summary

Regression test execution, demands that we be proactive along the testing life cycle, keeping in mind that the test cases that we develop today, in this release, will be integrated into the regression pool of test cases of tomorrow, for future releases. Bearing this in mind and analyzing the above, the regression strategy should rely on at least 3 major factors:

- Risk assessment results and data
- Coverage information
- Past defects logging

Those 3 elements encapsulate most of the data we need to perform optimization of our regression pool. This, however, is only after we have made sure that we have the ability to clearly say that the above-mentioned assumptions have been implemented.

Clearly, this is only the 'opening shot', and we still have to investigate the objectivity of these assumptions, and whether they fit many cases, i.e. many types of product and system. Also, there are open questions regarding the improvement of the process of implementing and integrating those factors into a pragmatic way of establishing and maintaining the regression pool effectively and efficiently as a clear methodology or guidelines for the industry.

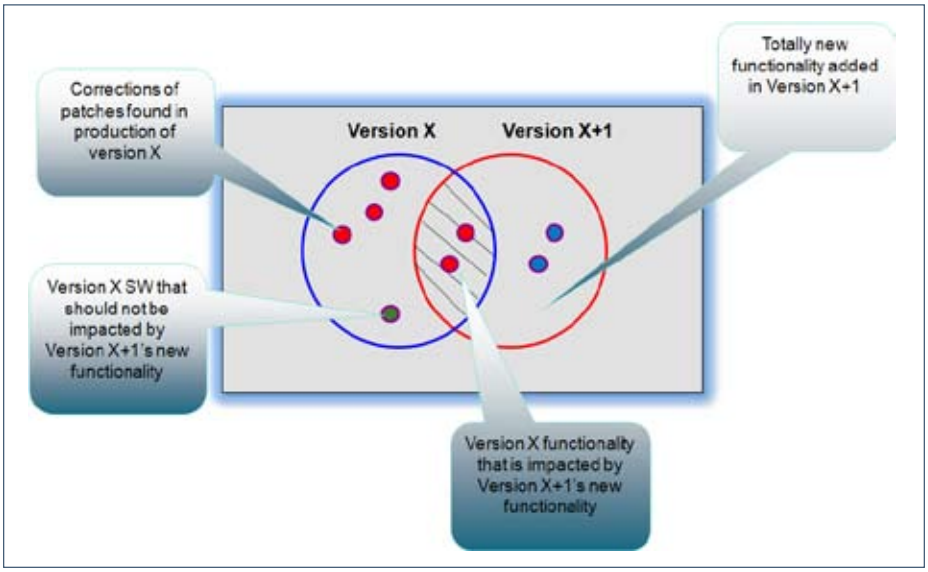


Figure 1: The impact of regression testing

Biography

Mr. Alon Linetzki is the founder and managing director of Best-Testing (www.best-testing.com), an innovative testing services company that brings effective coaching and mentoring, training and consulting to many organizations.

Mr. Linetzki has 16 years in testing and over 26 years in IT, and he is the author of several testing courses, including: TPI in a day, Risk-Based Testing, Personal Communication and Presentation Skills, Adding Business Value and Increasing ROI in Testing, Introduction to Test Automation, and more.
Part of Mr. Linetzki's experience includes: Establishing the IBM Testing Services group in Israel, where he was in charge of all the testing and QA activities for the SI projects; serving as R&D test manager of Amdocs, where he was in charge of testing all Amdocs offerings, products worldwide; leading a large development project of several hundred people in Canada for CMM level 3; and director testing at SELA, in charge of business development and serving as the principal consultant of testing training and consulting services.
During his professional career, he has consulted and trained over 900 professionals, in ca. 20 different companies.
Mr. Linetzki has been a popular speaker at international testing conferences since 1995, he is co-founder of the Israeli Testing Certification Board (www.itcb.org.il), and the founder and chair of SIGiST Israel (www.sigist.org.il). He is also a member of the marketing working party of the ISTQB.

XML Fuzzing Tool: Testing XML on Multiple Levels

by Rauli Kaksonen & Ari Takanen

During the past years, XML adoption has spread rapidly reaching almost every area of business and society. It is hardware and software independent, thus it enables communication across organizational boundaries. Due to its diversity, XML is widely used in modern protocols, file formats and applications. There is hardly an industry where XML is not used. Here also lie the biggest security risks: versatile technologies tend to create complexity, which is a breeding ground for security vulnerabilities. In addition, an increasing number of businesses and organizations utilize XML in business-critical applications. Indeed, the fast adoption of XML and the diversity of use cases have overwhelmed traditional security testing and few or no security testing solutions for XML have been available to customers.

Vulnerability of XML applications

Parsers do not protect applications

A common misconception about XML is that storing data in XML is an excellent way to avoid data corruption. Even some academics believe that XML formats are more resilient, because XML parsers assume nothing about the input. The parsers are also supposed to ensure that the mark-up of the data is syntactically correct and that each tag is associated with appropriate values. Thus, the parser would protect your application by only allowing inputs which the application logic can handle. In reality, XML parsers can be broken and malicious code can pass through them and attack the business logic of your application. XML does not protect you from parser errors, nor does it guarantee the security and robustness of your application. In fact, using XML actually makes your systems more vulnerable due to its complexity.

XML is prone to vulnerabilities

There are several factors, which heighten the importance of XML security testing. Not only is XML used in business-critical applications, but its complexity also makes it prone to vulnerabilities. In contrast to other mark-up languages, XML tags are not predefined. Instead, users can create tags and information structures which best suit their purpose. In addition, due to the nature of XML applications, the interfaces have to allow a large number of users to feed information into the interface. In the case of internet applications, all internet users have access to at least the login page. Indeed, most attacks are made through public interfaces, because this is the easiest way. Furthermore, most applications rely on a few XML libraries, e.g., Expat, libxml or Xerces, to process XML. While the business logic or schema is usually the company's own proprietary code, very few companies develop their own parsers or platforms. As a result, vulnerabilities in XML libraries can have serious widespread consequences: attackers might just know a few flaws, but if these are from the most widely used libraries, then they can attack just about any system. However, while parser errors can result in Denial-of-Service (DoS) situations or hostile code can be executed on a vulnerable host, flaws in application logic can be even more damaging as they may lead to the incorrect or partial execution of the organization's core operations. Just think of an online billing system, which charges the customer twice or a patient database, which gives false information.

Fuzzing XML applications

Representing real threats

Robustness testing is a software testing technique, in which unexpected data is fed to the inputs of a system, and the behavior of the system is then monitored for stability, security and reliability. If the system fails, e.g., by crashing or by failing built-in code assertions, then there is a bug in the software. Fuzzing is a form of robustness testing, which focuses on communication interfaces and the discovery of security related issues, such as overflows and boundary value conditions. It is a very representative method of testing software robustness, because it targets the problems attackers would also find. In robustness testing, thousands and even millions of misuse-cases are created for each use-case, thus most robustness testing solutions contain at least some degree of automation.

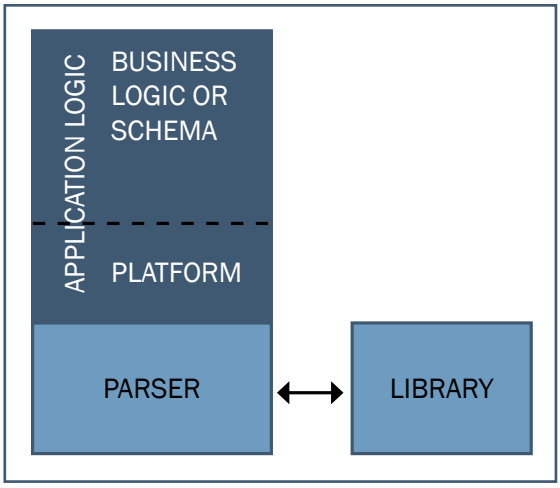


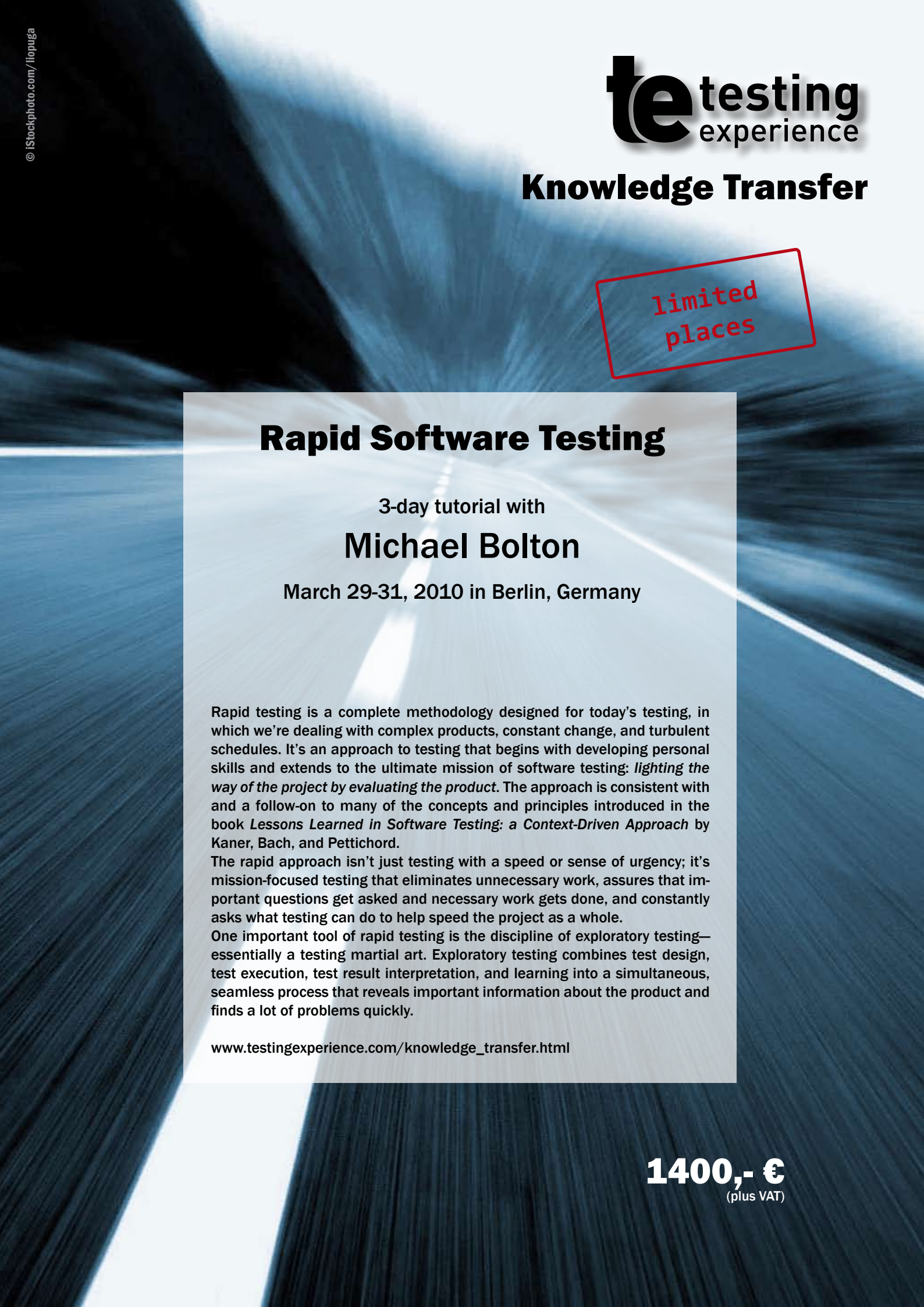
Figure 1: XML Application structure

Mutation-based vs. model-based Fuzzing

There are two popular ways to automate Fuzzing: mutation-based and model-based Fuzzing. Mutation-based Fuzzers are created by breaking down structures used in message exchanges, and by manually tagging these building blocks with metadata, which then enables the automation of the mutation process. In model-based Fuzzing, the process of data element identification is already automated by using specifications, which also provide the model with other protocol or file format specific information, e.g., on the boundary limits of the data elements. Thus, model-based test generation is often more systematic than mutation-based test generation. Model-based test case generation also significantly reduces the number of test cases needed, because the protocol and file format specific information from the specifications can be used to target the test cases.

Comprehensive testing

There is an infinite number of ways to manipulate XML messages, e.g., characters can be added or removed, special characters added, or elements repeated. Fuzzing differs from penetration testing in that Fuzzing enables tests, which are performed manually in penetration testing, to be automated and tested



testing experience

Knowledge Transfer

limited places

Rapid Software Testing

3-day tutorial with

Michael Bolton

March 29-31, 2010 in Berlin, Germany

Rapid testing is a complete methodology designed for today's testing, in which we're dealing with complex products, constant change, and turbulent schedules. It's an approach to testing that begins with developing personal skills and extends to the ultimate mission of software testing: *lighting the way of the project by evaluating the product*. The approach is consistent with and a follow-on to many of the concepts and principles introduced in the book *Lessons Learned in Software Testing: a Context-Driven Approach* by Kaner, Bach, and Pettichord.

The rapid approach isn't just testing with a speed or sense of urgency; it's mission-focused testing that eliminates unnecessary work, assures that important questions get asked and necessary work gets done, and constantly asks what testing can do to help speed the project as a whole.

One important tool of rapid testing is the discipline of exploratory testing—essentially a testing martial art. Exploratory testing combines test design, test execution, test result interpretation, and learning into a simultaneous, seamless process that reveals important information about the product and finds a lot of problems quickly.

www.testingexperience.com/knowledge_transfer.html

1400,- €
(plus VAT)

more thoroughly. Penetration testing requires substantial knowledge of protocols and systems from the testers, whereas in Fuzzing the expertise can be built into the tools, thus the tests can be performed by relatively inexperienced testers. In addition, Fuzzing improves test coverage, because model-based tests also cover features, which are rarely used. Malicious input in rarely used features tends to cause havoc in systems, because these features are not subject to rigorous testing or heavy day-to-day usage. As a systematic testing method, Fuzzing is also a suitable method for automating security audits. The freely available PROTOS is already a standard Fuzzing tool used by many security assessment professionals.

Testing on multiple levels

An XML application can be divided into three levels: parser, platform and business logic. The parser and the platform enable the application to process XML, and the business logic or the schema provides the actual service. Due to the level-like structure, XML must be tested using feeds, which target the specific levels individually. Otherwise, the parser will notice the lower-level anomalies and the higher-level anomalies will not be able to pass through the parser to test the business logic. Both application levels need to be tested: lower-level attacks are more common, because they are quicker and take place before any authentication is required, but vulnerabilities in the business logic can have more devastating consequences, because they affect the organization's ability to perform its core operations. This type of stateful or level-specific testing requires genuine interoperability between the testing tools and the system under test (SUT). The tools need to ensure that the inputs are processed, and they need to adjust their behavior according to SUT responses, thus providing thorough input coverage. The attack interface depends on where XML is used. If the library is used to process files, then the flaw is most probably only local. However, if the library is used in communications software, the bug is automatically remote.

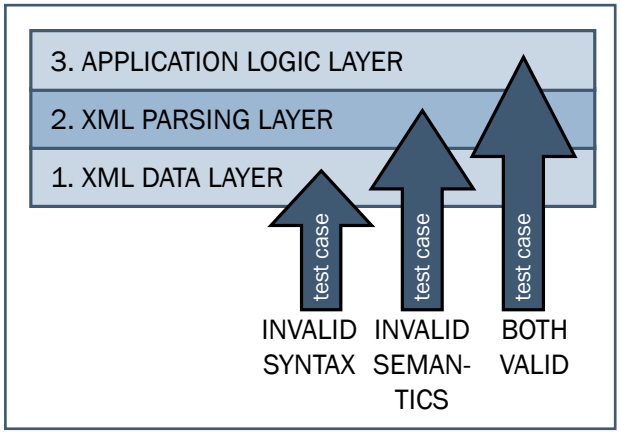


Figure 2: Stateful Fuzz Testing

Importance of security and robustness testing

Mitigating unknown threats

All unpatched software flaws are security threats. Basically, any bug can be exploited to attack a system or an application. Bugs are weaknesses in the system, which can be exploited. Attackers send feeds to a system, and if they can get an abnormal response from the system, they continue to edit their feeds until they get the system to behave the way they want. However, bugs do not need to be exploited for criminal purposes to cause problems. Sometimes the bugs can be triggered by events like heavier than normal use or maintenance. An unknown bug is a ticking time bomb, it does not really matter what eventually sets it off. The important thing is to get rid of them.

Building security into systems

The best way to guarantee security is to ensure that the code is well written. Using authentication, signing or encryption merely narrows down the number of users able to exploit the vulnerability. In addition, a lot of traffic already takes place before authentication. Good-quality coding cannot be substituted by adding security. External security features merely add to the complexity of the system, and complexity is always a threat, because it increases the attack surface of the system. The same applies to firewalls and anti-virus programs. Furthermore, application service providers, like online banks and web stores, cannot build fortresses around their application, because their customers need to be able to use their services. Fuzzing enables testers to accurately simulate potential attacks, and to patch the found vulnerabilities, before somebody else finds them and exploits them. Fuzzers do what attackers do, but before them. By integrating fuzz testing into the software development process, flaws can be discovered at the earliest possible moment. The earlier the bugs are discovered, the cheaper and easier it is to fix them.

Benefits of preemptive testing

Fuzzing can be used to discover flaws and create patches for them proactively, before any problems occur. Fuzz testing can be used to find bugs, which are only exposed in overload situations or which are slowly aging the system and causing its decay. Through proactive testing, these vulnerabilities can be discovered in advance, instead of testers being oblivious to the problems until the system finally breaks down. Thus, Fuzzing leaves the testers more time to patch the vulnerabilities. Furthermore, if vulnerabilities are fixed early on during the software development process, then there simply will not be any time for anyone to devise an attack. The costs of bad Quality of Service (QoS) and downtime can be considerable to company reputation and sales. By ensuring the quality of their software companies can also avoid other liability problems. And, most importantly, by building the security into the system, one can be sure that it is really secure and not just coated with security: bugs should be fixed and not protected.

This article is based on interviews with Rauli Kaksonen, Founder and Fellow of Codenomicon Ltd., and Ari Takanen, Founder and Chief Technology Officer at Codenomicon Ltd.

Biography

Mr. **Rauli Kaksonen** is a founder and fellow of Codenomicon Ltd. He has developed security and robustness testing methods in academic and commercial settings since 1999. Before co-founding Codenomicon, he worked as a Researcher at VTT Electronics, Technical Research Centre of Finland. Mr. Kaksonen is a Master of Science and Licentiate of Technology from the Department of Electrical Engineering at Oulu University.

Mr. **Ari Takanen** is a founder and the CTO of Codenomicon Ltd. He is a distinctive member of the global security testing community and a regular speaker at various testing and security conferences. His professional background is on academic software security testing research, where he has been active since 1998. Codenomicon Ltd. is a spin-off of the widely acclaimed PROTOS project of the Oulu University Secure Programming Group (OUSPG).

ISTQB® Certified Tester-Foundation Level

in Paris, in French

Test&Planet : votre avenir au présent.

Venez suivre le Cours « Testeur Certifié ISTQB – Niveau fondamental » à Paris, en français !

Test&Planet, centre de formation agréé DIF, vous propose ce cours D&H en exclusivité.

Apprenez à mieux choisir Quoi, Quand et Comment tester vos applications. Découvrez les meilleurs techniques et pratiques du test logiciel, pour aboutir à une meilleure qualité logicielle à moindre coût.

Possibilités de cours sur mesure.

Pour vous inscrire ou pour plus d'information : www.testplanet.fr

Díaz Hilterscheid

Test&Planet

Functional tests with the FEST framework

by Dominik Dary

This article describes an easy and quick way to write robust and compact tests for Swing-rich applications using the open source framework FEST.

In most projects a lot of time is used for testing the application with unit tests and functional Graphical User Interface (GUI) tests, which makes the entire system safer and more robust. Most of these functional tests are often done manually, which is very time consuming.

Automated GUI tests can be used for verifying the main scenarios of the application. GUI testing is also essential during application maintenance; with automated tests, the impact of refactorings can be tested within minutes. The aspect of visualizing the GUI components is not the main intent of this test approach.

Test frameworks should provide a solid way of finding GUI components, to get robust tests that are also able to run after layout changes. An intuitive and well-documented test framework is important for the readability of the written tests and allows the tester a more rapid learning.

Those functional tests describe scenarios. Inside a task manager application, a test scenario can be e.g. the entering of new to do items. This process includes several steps, such as opening the new task form, entering some values into the text fields and clicking the "create" button. Afterwards the created task must be verified. This can be done using the GUI, e.g. by loading the task form, or directly in the database to assert the entered values are saved correctly.

The tools employed in GUI test automation can be categorized as follows:

- **Capture and Replay**: Testers click on the test scenarios and thereafter the test framework captures all GUI interactions, which are stored in a script that can be replayed automatically.
- **Keyword-driven testing**: This approach (e.g. FitNesse) is used for user acceptance tests in which the tests are written using simple tables in html files. Each line represents an interaction with the GUI through a command like "enter text". The columns are used for commands, context, selector, arguments and comments. GUI components are identified inside the GUI-Dialog by their unique name. This simple approach can be used by testers without IT background and adds a layer of abstraction onto writing automated tests.
- **Technical Frameworks**: These tests are written in traditional programming languages like Java or Ruby. Components are also identified by the name of the component or more complex component lookups. The biggest disadvantage of this approach is the required basic know-how about software development and the time needed to learn the – in some cases cryptic – technical framework Application Programming Interface (API).

A very important criteria for selecting a test framework is its maintainability and the duration of the training period. Capture and replay tools are at first glance an easy and fast way to create test scripts. But these scripts are error-prone, especially if the layout of the application has changed.

Keyword-driven testing enables domain experts to use this approach, because of the simplicity of the test case specification. These tests can be created with editors like Word, and there's a basic concept of modularization. Below these frameworks, technical frameworks like JFCUnit are used.

If keyword-driven testing simplifies the usage of technical frameworks, why should a team decide to use a technical framework?

This question can be answered from a domain view and a technical view:

- From a domain perspective, it is important for whom the application is designed. If the application is developed for experts with a lot of specialized views and with a number of extended and self-developed GUI components, a technical framework is much more flexible.
- From a technical perspective, the usage of a technical framework with a well-defined API offers the possibility to use code editors, in which the syntax is highlighted and typing errors are displayed. These editors offer code completion, refactoring, debugging and integration of various version control systems, so that the history of the test file can be compared to each version. Another advantage is the power of the programming language (e.g. all features of the Java language can be used) and the fact that existing utility classes can be reused. The biggest disadvantage is the necessary Java know-how of the testers.

But what does a well-defined API mean? Eric Evans and Martin Fowler invented the term of "fluent interface", which is a design principle for an object-oriented API with the objective to offer a more readable code that is only used within a limited scope (e.g. functional GUI testing)[1]. A special characteristic of this design is the return value of a method call, which enables to build a method chain of a class.

A sample code can look like this:

```
dialog.botton("login").click();
```

In this code sample, the login button is resolved by its name and afterwards a click event is dispatched to it. This type of API is a so-called fluent interface, which offers the developers the possibility to enhance an API to a domain-specific language (DSL). A DSL is commonly described as a computer language targeted at a particular kind of problem and not intended to

1 http://www.martinfowler.com/bliki/FluentInterface.html

solve problems outside of its domain.[2]

FEST Framework

Open Source Library for GUI Testing

- Supports functional Swing GUI testing
- Website: http://fest.easytesting.org
- Its API provides a fluent interface
- Open Source project (Apache 2.0 license)
- Supports both TestNG and JUnit
- Simplifies troubleshooting GUI test failures
- Hosted at CodeHaus

FEST is a Java-based framework for automated functional GUI testing, which provides a fluent interface. FEST is an acronym and stands for "Fixtures for Easy Software Testing". Test cases are implemented as unit tests, and applications based on Swing and SwingFX can be tested. This framework is further developed actively and the javadoc coverage of the source code is high. It has an active community; questions on the mailing list are answered very fast and efficiently. The existing wiki pages also inform about advanced topics. Based on the fluent interface offered and the good javadoc descriptions, intuitive learning is possible.

FEST is divided into four modules:

- Swing

  Simulation of user-generated events and solid GUI component lookup

- Assertion[3]

  Flexible and fluent assertions

- Reflection[4]

  'Fluent interface' for simplifying usage of reflection.

- Mock[5]

  Eliminates code duplication and clearly separates mock expectations from code to test, improving code readability

The most important module for automated functional testing is the Swing module. The other modules can be optionally used and are not described in this article.

(See Figure 1)

The Swing module is divided into different layers. "`Basic robot`" is the basic layer of the FEST-Swing module. The class `java.awt.Robot`[6], which is part of the Java Development Kit (JDK), is the main component of FEST-Swing. This class is used to generate native system input events for the applications, for which input through the mouse and keyboard is needed. FEST has created an abstraction interface and the implementation delegates the calls to the robot class of the JDK.

The "`component driver`" layer contains driver classes for all Swing GUI components. The driver knows in detail which types of interaction are possible and how the state can be checked. The `JTextFieldDriver` interface, for example, contains methods like:

- enterText(..)
- setText(…)
- requireText(…)
- requireEditable(…)

The component "`fixture layer`" is located on top of the driver layer and provides a DSL-oriented API for the tester. The main difference between the driver and fixture layers is the different style of programming. The driver layer is designed in a classic object-oriented way and is used by the fixture layer. The fixture layer is designed in a completely different way: every method returns the GUI component fixture class itself to provide the ability of method chaining. This fluent interface makes the API much easier to write and read.

This small example enters a text into two text fields, selects a radio button and clicks on the calculation button. The last step is the validation of the calculation. Isn't this code easy to read and understand?

Sample test code using the FEST API

```
//Fill in parameters
window.textBox("firstNumberInput").enterText("22");
window.textBox("secondNumberInput").enterText("2");

//Select action
window.radioButton("sumRadioButton").check();
//Calculate
window.button("calculateButton").click();

//Validate the result
window.textBox("resultOutput").requireText("24");
```

By default, all GUI components can be identified by name, as in the example, or by type. However, there are also more sophisticated finders that can be used.

```
window.robot.finder().
findByLabel(label, type)
```

This finder will find a component by its referenced label.

After selecting a component, it is possible to interact with this component. As shown in the example, the input of keyboards can be simulated, just as mouse click events can be sent to the components.

The components used in the example are only simple GUI components. However, how easy is the selection of a special node of a `JTree`? Imagine a folder structure that is displayed in a tree:

- My Documents
  - FEST
    - Article
    - Sample
  - FitNesse

The leaf node "Sample" can be selected using the `JTreeFixture`:

```
JTreeFixture myDocument-
sTree = JTreeFixture(Robot,
"folderTree");
myDocumentsTree.
selectPath("My Documents/
FEST/Sample");
```

Another more complex GUI component is a `JTable`, which differentiates between the rendering and the editing of cells, e.g. a cell displays only a text, but when you double-click on the cell, a drop-down list is displayed.

The following Java code shows an interaction with the To-Do-List table of the demo application:

These short examples demonstrate the elegance and the possibilities of the FEST framework testing Swing applications – the new rich internet application technology SwingFX is also testable. For readers who are interested in more examples, a good starting point is the sample application of this article with more FEST samples[7].

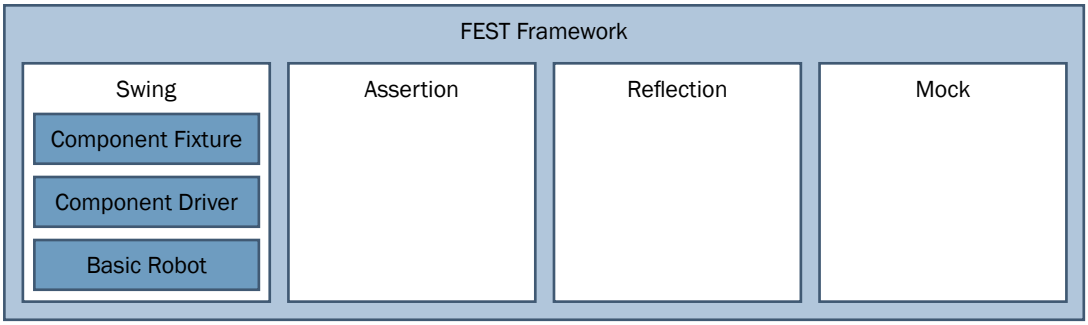


Figure 1: Overview of the modules of the FEST framework

2 http://www.infoq.com/articles/internal-dsls-java

3 http://docs.codehaus.org/display/FEST/FEST-Assert

4 http://docs.codehaus.org/display/FEST/FEST-Reflect

5 http://docs.codehaus.org/display/FEST/FEST-Mocks

6 http://java.sun.com/javase/6/docs/api/java/awt/Robot.html

7 http://github.com/DominikDary/ToDo-

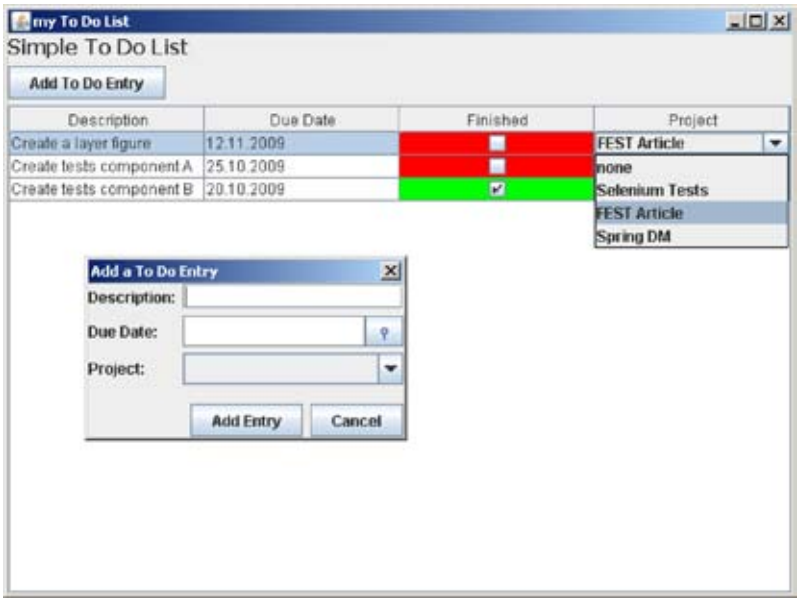


Figure 2: Screenshot of the sample application

```
//import static org.fest.swing.data.TableCell.row;

//Find table GUI component
JTableFixture taskTable = window.
table("toDoListTable");

//Selecting the finish cell
JTableCellFixture finishedCell = taskTable.
cell(row(0).column(2));

// Selecting the checkbox
finishedCell.enterValue("true");
finishedCell.background().requireEqualTo(Color.
GREEN);

//Selecting the project cell
JTableCellFixture projectCell = taskTable.
cell(row(0).column(3));

// Select a value in the comboBox
projectCell.enterValue("Spring DM");

// Assert the selected value is displayed afterwards
projectCell.requireValue("Spring DM");
```

FEST tests are written as unit tests; both popular frameworks - JUnit and TestNG – are supported. If TestNG is used as unit test framework, this offers the possibility to create your own reports. During testing the application, screenshots can be taken using the ScreenshotTaker manually or automatically if the test fails. Those functional tests can be integrated[8] into continuous integration environments like Hudson[9] or TeamCity[10].

For self-developed GUI components, it is helpful to create driver and fixture classes as they exist for the Swing components. Another helpful approach is to create fixture classes for GUI dialogs to offer methods for easy component selection and some simple flows, e.g. choosing a file with JFileChooser which can be centralized. This helps to minimize code duplication, simplifies the tests and increases once more the readability of the tests. Newly designed and developed Swing desktop applications are much easier to test if code conventions contain a naming concept to guarantee that each GUI component has a unique name.

Tests of complex desktop applications need a flexible, well-documented framework to create maintainable tests; check out the sample application and get started writing functional tests with FEST!

List

8 http://docs.codehaus.org/display/FEST/Continuous+Integration
9 https://hudson.dev.java.net/
10 http://www.jetbrains.com/teamcity/

Biography

Dominik Dary is a Software Engineer working for Capgemini sd&m AG in Germany. Dominik has 5 years practical experience in specification and realization of object-oriented software in Java / Spring / JEE as well as in quality assurance and in business intelligence. In 2005 he wrote his diploma thesis about "Quality Assurance in JEE Projects by Test-Driven Development". The practical part of this thesis is based on the "Cactus" and the "Canoo webtest" frameworks.

Dominik's first contact with FEST was in 2008 developing a Java Swing document management / workflow application. His further activities in this project included test specification and organization of acceptance tests.

In his current project, Dominik works for an e-commerce company doing test management, performance testing and automated testing with the selenium framework.

ISTQB® Certified Tester Training in France

www.testplanet.fr

a Díaz & Hilterscheid partner company

Testing Services

www.puretesting.com

PureTesting

Testing Thought Leadership

Training by

SELA GROUP

www.sela.co.il/en

ISTQB® Certified Tester Training in Spain

www.certified-tester.es

Díaz Hilterscheid

IREB

Certified Professional for Requirements Engineering - Foundation Level

http://training.diazhilterscheid.com/

training@diazhilterscheid.com

24.02.10-26.02.10	Berlin
21.04.10-23.04.10	Berlin
09.06.10-11.06.10	Berlin

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

ISTQB® Certified Tester Advanced Level Test Manager

www.testingexperience.learntesting.com

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

Masthead

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin, Germany

Phone: +49 (0)30 74 76 28-0 Fax: +49 (0)30 74 76 28-99 E-Mail: info@diazhilterscheid.de

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.testingexperience.com

ARTICLES & AUTHORS
editorial@testingexperience.com

350.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
online version: free of charge
print version: 8,00 € (plus shipping)

ISSN 1866-5705

Index Of Advertisers

your great escape

GranCanaria.com

Training with a View

Díaz Hilterscheid

also onsite training worldwide in German, English, Spanish, French at
http://training.diazhilterscheid.com/
training@diazhilterscheid.com

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

- subject to modifications -

05.01.10-07.01.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
18.01.10-20.01.10	Certified Tester Foundation Level - Kompaktkurs	Stuttgart
18.01.10-22.01.10	Certified Tester Advanced Level - TEST ANALYST	Frankfurt am Mein
25.01.10-29.01.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
08.02.10-10.02.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
15.02.10-19.02.10	Certified Tester - TECHNICAL TEST ANALYST	Berlin
22.02.10-25.02.10	Certified Tester Foundation Level	Frankfurt am Man
22.02.10-26.02.10	Certified Tester Advanced Level - TESTMANAGER	Düsseldorf
24.02.10-26.02.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
01.03.10-03.03.10	ISSECO® - Certified Professional for Secure Software Engineering	Berlin
08.03.10-10.03.10	Certified Tester Foundation Level - Kompaktkurs	München
15.03.10-17.03.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
15.03.10-19.03.10	Certified Tester Advanced Level - TEST ANALYST	Düsseldorf
22.03.10-26.03.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
12.04.10-15.04.10	Certified Tester Foundation Level	Berlin
19.04.10-21.04.10	Certified Tester Foundation Level - Kompaktkurs	Hamburg
21.04.10-23.04.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
28.04.10-30.04.10	Certified Tester Foundation Level - Kompaktkurs	Düsseldorf
03.05.10-07.05.10	Certified Tester Advanced Level - TESTMANAGER	Frankfurt am Main
03.05.10-07.05.10	Certified Tester - TECHNICAL TEST ANALYST	Berlin
10.05.10-12.05.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
17.05.10-21.05.10	Certified Tester Advanced Level - TEST ANALYST	Berlin
07.06.10-09.06.10	Certified Tester Foundation Level - Kompaktkurs	Hannover
09.06.10-11.06.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
14.06.10-18.06.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
21.06.10-24.06.10	Certified Tester Foundation Level	Dresden