No test levels needed in agile software development!

Leo van der Aalst[1]
Lector “Software Quality and Testing” at Fontys University of Applied Sciences
l.vanderaalst@fontys.nl

Abstract

Testing is not only a vital element of agile projects, it is an integral part of agile software development. In traditional software development environments test levels like system test, user acceptance test and production acceptance test are commonly executed. In an ASDE, all team members have to work together to get the work done. All disciplines have to work together and support each other when necessary. There are no designer, developer, user or test teams in an agile project. In such an environment it does not make much sense to talk about system test and/or acceptance test teams. Instead all team members have to accept the feature (or user story or use case, etc.) with their own acceptance criteria in mind. For instance, an end-user should test the suitability and user-friendliness of the feature. Operations should, for instance, test the performance, the manageability and continuity of the feature. And the designer should test the functionality of the feature. In short, in an ASDE test levels are replaced by combinations of acceptor and quality characteristics important to the acceptor per feature. This approach requires a different mind-set of the team members, a different product risk analysis approach and a different view on establishing the test strategy.

Biography

Leo van der Aalst *has almost 25 years of testing experience and developed amongst others services for the implementation of test organizations, agile testing, test outsourcing, software testing as a service (STaaS), risk based testing, calculation of the added value of testing and test-governance.*

Leo is lector "Software Quality and Testing" at Fontys University of Applied Sciences (Eindhoven, The Netherlands) and he is co-author of the TMap NEXT® for result-driven testing and TMap NEXT® Business Driven Test Management books. He is also a member of the Dutch Innovation Board Testing and of the Research and Development unit of Sogeti Netherlands[2]*.*

Besides all this, Leo is a much sought-after teacher of international test training, a regular speaker at national and international conferences, and he is the author of several articles.

Speaker at conferences: a.o. STARWEST and PNSQC (both USA), Test Congress, Iqnite and Test Expo (all UK), Quality Week Europe (Belgium), SQC/Iqnite and TAV (both Germany), Swiss Testing Day (Switzerland), ExpoQA and QA&TEST (both Spain)

Cecile Davis *is co-author of this paper. She is a test consultant and has been involved in several agile test projects. She is a certified agile tester, RUP-certified, co-author of TPI NEXT® and founder of the SIG on agile testing within Sogeti. She is involved in several (national) SIG’s related to this subject.*

---

[1] Rick D. Anderson thank you for your input. You were a great help!
[2] TMap NEXT and TPI NEXT are registered trademarks of Sogeti Nederland B.V.

1. Introduction

Structured testing can be perfectly integrated in agile development. This paper is intended for everyone with an interest in testing in relation to the agile manifesto and, specifically in how to test without test levels in an agile software development environment (ASDE). This paper does not describe in detail how to test without levels together with a specific agile method like scrum.

1.1 Vision on testing in agile environments

In agile processes a number of aspects pose a significant challenge to the traditional view of professional testing: lack of detailed requirements, reduced multipurpose process documents (test strategy, plans and cases etc.), always delivering working software, nightly integration and builds, user involvement, short time boxed iterations, potential technical requirements for testers, change of culture (self managed teams), distributed or off-shore teams and the fast pace of delivery.

It is important to realize that agile methodologies arose from the software developer's side. Agile is not a prescriptive discipline, hence, specific processes like testing, configuration management, build and release processes are not discussed. However neutral the agile manifesto is towards the test discipline, the methods that embraced the agile philosophy never focused on how to integrate the testing discipline thoroughly in ASDEs, nor did they consider what the consequences for test professionals would be. Therefore many organisations struggle with the implementation of testing in an ASDE and since testing is not only a vital element of agile projects but also an integral part of agile software development, we have defined a *test vision* in addition to the agille manifesto:

1. Use the agile manifesto as a starting point
   A test vision should based on the four values of the agile manifesto together with the twelve principles. This means that each and every proposed test activity must be in line with the agile manifesto and its principles.
2. Integrate the test activities in the ASDE
   a. The test activities must be integrated in both the development process itself and in the teams. Agile teams test continuously and all team members must be prepared to perform test activities. Although a professional tester should be part of the team, this does not mean that all test activities must be carried out by the tester.
   b. Testing should move the project forward. This means a shift from quality gatekeeper solely to collaboration with all team members to provide feedback on an ongoing basis about how well the emerging product is meeting the business needs.
   c. Test tools are increasingly important and should be used to support and improve the performance of agile teams.
   d. Test activities/acceptance criteria must be part of the definition of done.
3. Find balance by making well-considered choices
   The right balanced choices will always be context sensitive, which means that different factors like type of organization, type of project, business goals, available resources and available technology are taken into account.
4. Reuse values of the traditional structured test approaches, e.g. TMap NEXT® [3]
   The traditional test approaches are still very valuable, but sometimes they must be adapted to the agile way of working.

In this paper I'll explain at a high level how to put the above stated test vision into practice and in detail how to put a test approach without test levels into practice (*bullet 2a*).

[3] Aalst, L. van der, Broekman, B., Koomen, T., Vroon, M. (2006), TMap NEXT®, for result-driven testing, 's-Hertogenbosch: Tutein Nolthenius Publishers, ISBN 90-72194-80-2

1.2 Shift to an agile software development environment

In sequential software development lifecycles, the emphasis traditionally has been on defining, reviewing and subsequently validating the initial business requirements in order to produce a full set of high-quality requirements. Further levels of testing, such as system and user acceptance testing, are then planned to achieve coverage of these requirements and their associated risks.

This approach, combined with a full lifecycle testing strategy, utilizing effective document/code evaluation and other levels of development testing, such as unit and unit integration testing, can achieve very high levels of software quality.

However, in reality, projects invariably fail due to a lack of end-user involvement, poor requirements definition, unrealistic schedules, lack of change management, lack of proper testing and inflexible processes.

This traditional approach means that there often is a disconnection between users and testers. As a result, changes to requirements that often surface during the design or coding phase may not be communicated to the test team. This results either in false defects, or in a test strategy being incorrectly aligned with the real product risks. To combat this, traditional projects exert a lot of effort in managing change because in long-term projects, change is inevitable.

The agile software development approach, based on iterative development in which requirements and solutions evolve in combination and change is embraced, would therefore appear to offer a potential solution to the problems in a traditional approach.

Incremental software development processes have been specifically developed to increase both speed and flexibility. The use of highly iterative, frequently repeated and incremental process steps and the focus on customer involvement and interaction theoretically supports early delivery of value to the customer.

Agile software development is not a method in itself. It is derived from a large number of iterative and incremental development methods. Methods such as:

- Rapid Application Development (RAD)
  - 80's, Barry Boehm, Scott Shultz and James Martin
- Scrum - The New New Product Development Game – Harvard Paper
  - 1986, Ikujiro Nonaka and Hirotaka Takeuchi
- Dynamic Systems Development Method (DSDM)
  - 1995, DSDM Consortium
- eXtreme Programming (XP)
  - 1996, Kent Beck, Ken Auer, Ward Cunningham, Martin Fowler and Ron Jeffries
- Feature Driven Development (FDD)
  - 1997, Jeff de Luca

In 2001, in Utah, 17 representatives[4] of the above mentioned and other similar methods, met to discuss the need for lighter alternatives to the traditional heavyweight methodologies. In about three days they drafted the agile manifesto, a statement of the principles that underpin agile software development.

[4] The 17 authors of the manifesto were: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland and Dave Thomas.

1.3 The agile manifesto

Agile software development has many different 'flavours'. However, the foundation of any of these development methods known today can be found in the manifesto for agile software development. The agile manifesto consists of four values and twelve principles.

1.3.1 Four values

At http://agilemanifesto.org the manifesto is stated as:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions	*over*	*processes and tools*
Working software	*over*	*comprehensive documentation*
Customer collaboration	*over*	*contract negotiation*
Responding to change	*over*	*following a plan.*

While there is value in the items on the right, we value the items on the left more!"

Some ways wherein the first value expresses itself are the frequent face-to-face communication between individuals, the self-regulation of the multidisciplinary teams and the team's responsibility. The second value is followed by working in short iterations with working deliverables at the end of each iteration, by prototyping and by efficiency in documentation. The third value is supported by the involvement of customers in the team, frequent feedback and demonstrations. The last value can only be true if all disciplines are involved early, and the process is implemented based on a strategy that considers changes being inevitable.

1.3.2 Twelve principles

The twelve principles behind the agile manifesto are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity, the art of maximizing the amount of work not done, is essential.
11. The best architectures, requirements and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

2. How to put the test vision into practice

In this chapter the vision on testing in an ASDE will be discussed in more detail (see section 1.1).
In addition, some tips and hints will be given on how to put the test vision into practice.

The four, high level, test vision values are:
1. Use the agile manifesto as a starting point.
2. Integrate the test activities in the ASDE.
3. Find balance by making well-considered choices.
4. Reuse values of the traditional structured test approaches.

The above mentioned values are described in the following sections.

2.1 Use the agile manifesto as a starting point

When implementing an agile method, a lot of organizations pick one or two appealing values or principles from the agile manifesto and start to implement these favourite values. However, wise and sensible it may sometimes be to start with small steps, implementing only part of the values of the manifesto will not work in the end. It often happens that organizations stop implementing halfway or just do not spend as much attention to the other values as to the first. The manifesto is not a checklist you can pick from. The way these values are implemented is open, which is why there are different agile methods, but all values need to be implemented. And if an agile method is implemented partly, it will also have consequences on how to deal with test activities. Moving to agile is similar to any process change/improvement. That is, everybody needs to be clear on why the change, and the steps to be taken to change the process, is necessary.

First of all, to solve the issue of a test process that is not based on the agile manifesto as a whole, it is necessary that all stakeholders involved are familiar with the manifesto and the agile method that will be implemented. This seems very simple: just give them a book to read. However, it is very important that people really understand the values and their consequences. The manifesto and its principles stand for a mindset; it is not a checklist.

Furthermore, it is most important that people have the same understanding.
If everybody has to learn the values on their own, for certain, different interpretations will arise. That will lead to miscommunication and misunderstanding. The best and most complete solution is therefore, to start by appointing an expert as coach, who can give training and support. When everybody is used to the new way of working, this coach will no longer be necessary.

2.2 Integrate the test activities in the agile software development environment

In this section four aspects with respect to the integration of testing into an ASDE are discussed:
- Integrate in development processes and teams.
- Testing moves the project forward.
- Test tools are necessary.
- Testing is part of done.

2.2.1 Integrate in development processes and teams

In traditional development environments testing is often inserted at a later stage, although most test methods advise an early test involvement. In agile environments testing is, by nature, incorporated from the beginning and throughout the whole process. So, organizations need to think on how to deal with traditional test levels like system test, user acceptance test, end-to-end testing, and production acceptance test in an ASDE.

In this section three aspects with regard to continuous testing are discussed:

- Integrate testing activities with development activities.
- Test levels in an ASDE.
- The testers' role changes.

Integrate testing activities with development activities

Testing in agile projects cannot be seen as a separate activity but needs to be integrated in the entire process. Testing is not only a vital element of agile projects, it is an integral part of agile software development.

The underlying thought of the agile philosophy is to deliver business value as early as possible and in the smallest workable piece of functionality. To prove that the developed software works as desired *and* to prove business value, the user story and product need to be tested. To make agile work well, testing cannot be seen as a separate activity but needs to be integrated in the entire process.

Testers can add value in different areas by acting as a spider in the agile web. Besides being involved in testing, they will be involved in formulating the definition of done, in planning, in unit testing, in gathering all information necessary to form the test basis, in risk management, in retrospection, in test automation, etc. Therefore, the role of the tester is an important one. This early and high degree of involvement of testing in the entire project has a lot of benefits. For instance, including testing activities/acceptance criteria in the definition of done ensures that the software is considered 'done' only when the testing is fully 'done'.

A high degree of involvement at an early stage is essential for testing in agile environments. All disciplines, testing included, must be involved as soon as possible in the process. This is an essential part of agility, required, for instance, by the value in the manifesto of *responding to change.* In agile environments, changing the requirements can be done more easily than in a non-agile environment. If the change is discussed and agreed upon within the team, which would also include the customer, then no one can really oppose to the change. Testing, especially, is of importance in an early stage of changing requirements, because of the value it can add to impact analysis and risk analysis. The sooner testing can respond to changes and take actions, the better.

Also, responding to change raises a need for test automation, efficient and effective documentation and more face-to-face communication. These differences require a different approach and different skills from testers (see also section "the testers' role changes").

Test levels in an ASDE

In traditional software development environments, test levels like system test, user acceptance test, end-to-end testing and production acceptance test are commonly used. However, in ASDEs, all team members have to work together to get the work done. In such an environment it is not relevant to talk about system test and acceptance test levels/teams with managers and budgets of their own.

If there is a need for some distinction, instead of test levels, quality characteristics can be used per user story. This way, certain test activities can be grouped together. Quality characteristics can also be used to SMART-en up acceptance criteria in the definition of done. All team members have to accept the feature with their own acceptance criteria in mind. For instance, an end-user should test the feature suitability and user-friendliness. Operations should test the feature manageability, performance and continuity for instance. And the designer should test the functionality. In short, in an ASDE test levels are replaced by combinations of acceptor and quality characteristics important to the acceptor per feature. See figure 1.

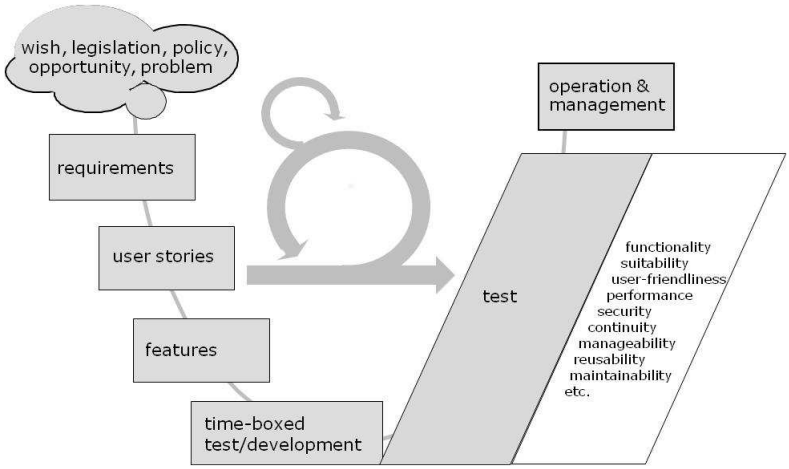

Figure 1: All test activities carried out by the agile team.

The characteristics for such an approach are:

- No seperate test levels, not in the iterations nor afterwards.
- All acceptors of the product must be present in the, relevant, iterations and prepare/execute their own tests (with or without help of the other project members)
- In the product risk analysis, the risk per combination of feature and quality characteristic should be determined by and per acceptor.
- After the last iteration the product is explicitely accepted by all acceptors.

However it is still possible to group certain test activities to a specific test level. One could say that all tests aimed at, for instance, testing the functionality of a feature is called the system test. And that all tests aimed at, for instance, testing the suitability and the user-friendliness of a feature is called the acceptance test.

Depending on for instance the availability of people, environments or for other reasons, it is also possible that certain test levels are not carried out by the agile project team itself, but it could be a test phase after the release is delivered or in parallel with the iterations. In this situation it's possible to make a distinction between iteration tests and release tests. See figure 2.

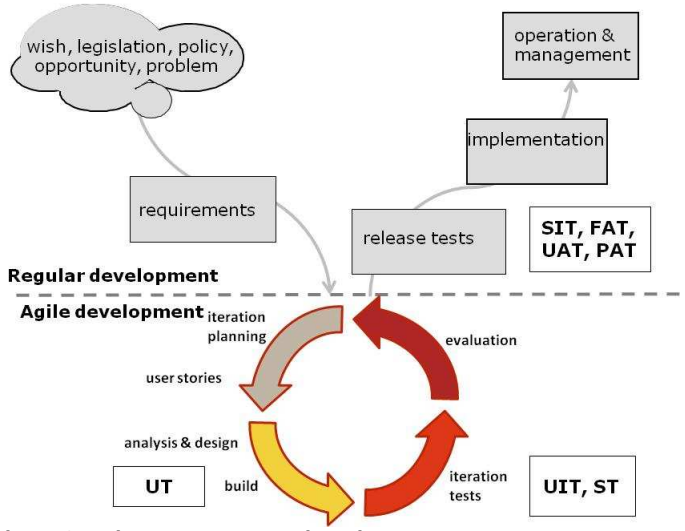

Figure 2: Release tests versus iteration tests.

And what about end-to-end testing? Is it possible to incorporate this test level in the agile process? This depends on the specific situation. Theoretically, end-to-end testing would benefit enormously from agile

software development. However, it will usually be difficult to realize the end-to-end test within the agile process. Therefore, it could be wise to implement end-to-end testing as a separate phase after the agile project is completed.

The testers' role changes
Agile project teams have to work together closely and the team should contain the people required to make the project successful. In agile methods, involvement of all disciplines, including testing, is part of the philosophy. Therefore, working with multi-disciplinary teams is very important. So a tester must be part of the agile team.

Working in multi-disciplinary teams also means that all team members must be prepared to fill in other roles if necessary. For instance, a developer must be willing to fill in a testing role if the need arises. Especially, of course, when there is time pressure on the test activities. On the other hand, testers must be willing to assist other team members in their activities where they can according to the projects needs. This can demand more technical skills and knowledge from a tester. Working together means two-way traffic.

The nature of the testers' role changes in iterative projects. Testers are no longer the high-profile victims, they are no longer the lonely advocates of quality. They are competent service providers, working in a team that wants to achieve high quality. The testers' role becomes richer and more influential. Agile methods require testers to be involved in the development project continuously and right from the start.

Testers need to have technical knowledge of the software they are testing and need
to understand the impact on automation as well as the functional implications. Some iterations may be development heavy, some automation heavy, some test heavy; the agile tester needs to be adding value in all instances.

The personal characteristics that are especially important for a tester working in an agile team are:
- Communicative
- A retrospective attitude
- Flexible
- Pro-active
- Creative but practical
- Thinking in solutions
- Customer-oriented
- Open-minded
- A team player
- Interfering with everything like a spider in a web

Professional testers should adapt to fit this different role and so provide additional value by not only focusing on finding defects but also fulfilling a team role.

2.2.2 Testing moves the project forward

On traditional projects, testing is usually treated as a quality gate, and the test team often serves as the quality gatekeeper. The result of this approach exist of long, drawn out bug scrub meetings in which different parties argue about the priority of the bugs found in test and whether or not they are sufficiently important to delay a release.

In agile teams, the product is built well from the beginning, using testing to provide feedback on an ongoing basis about how well the emerging product is meeting the business needs.

This sounds like a small shift, but it has profound implications. The adversarial relationship that some organizations foster between testers and developers must be replaced with a spirit of collaboration. This is a completely different mindset.

Testers must be pro-active and work with the business stakeholders to understand their real needs and concerns. In traditional environments, this is usually called ‘requirements elicitation’. In the context of agile development, the purpose of this discussion is not to gather a huge list of requirements but rather to understand what the business stakeholder needs from one particular user story.

During these discussions, the tester must ask questions designed to uncover assumptions, understand expectations around non-functional needs such as performance, reliability, security, etc., and explore the results the business stakeholder is requesting. This will improve the quality of the software product.

One, very effective, way of involving testers early is to introduce test driven development. This way, testers, programmers and business representatives can all learn from each other, while interpreting the requirements together.

2.2.3 Test tools are necessary

Over time, test tools have become increasingly important to the performance of agile teams. This is not just because teams sometimes have to be technically oriented, but because the right tools can help a team to become more efficient and agile.

If agile is about speed, efficiency and flexibility, then the role of automation in agile is to support this and remove mechanical, routinely and time consuming tasks. Due to the required speed of agile, management of test data and environments needs to be very efficient and effective with little if any room for unnecessary manual effort. Although this seems logical, it is still very hard for people working in an ASDE to decide upon the tools to be used and how these tools should be implemented and used. Tasks that can normally be automated within agile teams include:

- Build and integration process.
  Usually in agile teams, this process happens on a very regular basis (every night), resulting in a new build every day, with almost zero manual effort being put into this task. This requires good configuration management and build tools.
- Unit (integration) Test.
  These are part of the nightly build and integration process, allowing the development team to get instant feedback on the quality of their code. The execution of these unit tests requires no manual intervention.
- Static Analysis Tools.
  Instead of doing manual code reviews, the analysis tools review the code against coding standards to uncover defects. The manual reviews can be kept for particular types of defects or more complex code.
- Test data and environment management.
  Available tools can generate data to manage the test environment.
- Regression Testing.
  To be able to follow the quick pace of agile software development, agile teams cannot do without automated regression testing.
- Functional Testing.
  Until now, functional automation testing has focused on regression testing. However agile teams are pushing for functional testing to be automated earlier and earlier in the development lifecycle, so that it is the design of test cases rather than the execution that is important. The use of Model Based Testing (MBT) tools in combination with the automation of test case execution is a (almost) perfect solution for functional testing in agile environments. Test execution should be automated as much as possible.

2.2.4 Testing is part of done

In traditional software development environments with strict boundaries between development and test, it often happens that user stories are declared ‘done’ before it is tested properly. Of course, agile teams only count something as ‘done,’ when it has been implemented, tested and bugs have been fixed.

Incorporating testing in the definition of done gives more control on iteration planning, on testing risks, more early involvement for testers and therefore more grip on the test process.

2.3 Find balance by making well-considered choices

The most important consideration when using an agile software development method, is finding the right balance in the choices that have to be made. When you look at the four values of the agile manifesto, you could think about a balanced choice for each value between the left and right item.

In practice, however, we see agile project teams struggle mostly with finding balance between:
- Working software and comprehensive documentation.
- Covering the risk and the available time and money.

Therefore these two balances will be discussed in the following sections.

2.3.1 Find the balance between working software and comprehensive documentation

Does agile mean the end of all documentation? To those looking at adopting agile practices, especially if they come from more traditional project management backgrounds, agile can seem quite loose, especially in the area of documentation.

The agile manifesto values working software over comprehensive documentation. And in agile projects working software is perhaps the ultimate quantification of your projects status. This may take some time to get used to.

Agile methods are all in favor of documenting only what is necessary. They simply value working software over comprehensive documentation. However, what level of documentation is necessary may vary per project. Besides that, just deciding on what to document and what not, is not enough. Whenever decisions are made on, for instance, the extent of documenting requirements, it must be thought over how the information about these requirements that is not in the documentation is communicated between the different “requirements stakeholders”. There should be a balance between documentation and communication so that important information does not get lost.

In addition, there is a number of principles behind the manifesto that elaborate on this value:
- Working software is the primary measure of progress.
- The teams’ highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Sometimes end-user documentation is used as one of the criteria for ‘done’. It can also be used to document decisions for team continuity. And documentation may even be needed for regulatory compliance.

With regard to the quantity of documentation to be produced, the team could be asked to answer two simple questions:
- Does the documentation add value?
- Is the team better off writing it?

2.3.2 Find the balance between risks versus time and money

An agile project, like any other project, never has unlimited time and money for testing purposes. Such constraints in terms of time and money represent constraints on the test result to be achieved and

therefore reduced coverage of the product risks. As such it is important to make well-considered choices in relation to the optimum division of the available time and money across the user stories that require testing. In fact we need to determine product risks and test strategy in an *agile business driven test management way*.

Based on the insight resulting from the product risk analysis, high risk user stories can be tested more thoroughly than those representing a lower risk. A product risk is the chance that the product (user story) will fail in relation to the expected damage if it does.

Product risk = Chance of failure * Damage

When performing the product risk analysis, all stakeholders should be member of the agile team and participate in the analysis.

In an agile environment it is difficult to perform a product risk analysis for the release as a whole. There is just not enough detail available of individual user stories. So, in practice a product risk analysis can be performed at the start of each iteration. Since an iteration will deliver a few user stories only, it does not take much time to perform the analysis (it could be done on a whiteboard).

The easiest way to proceed is the following:
1. Gather all team members.
2. List all user stories of the current iteration (on the whiteboard for example).
3. Ask all participants which quality characteristic(s) per user story is (are) important to the participant.
4. Determine per user story and quality characteristic what the “Chance of failure” and the “Damage” are. If figures are used. The risk is simply the product of the figures.

At this moment the product risk analysis is done and the whiteboard could look like this (see figure 3):

User Story	Characteristic	Stakeholder	Damage	Chance of Failure	Risk Class
US 1	Functionality	Person A	3	3	9
	User-friendliness	Person B	2	1	2
US 2	Functionality	Person A	2	2	4
	Security	Person C	3	2	6
US 3	Functionality	Person A	2	1	2
US 4	Performance	Person C	2	1	2
US 5	Performance	Person C	1	1	1
US 6	Functionality	Person A	2	2	4
	Suitability	Person B	2	2	4
..	..		..	..	..

Figure 3: Risk table.

The next step is to determine the testing strategy. In this activity, the outcome of the product risk analysis is used to determine which combinations of quality characteristic and user story are to (must) be tested with what test intensity. In order to make the table more concrete it is possible to extend the table with a column “Test Design Technique”. This will help the team members with testing roles, preferably the stakeholders themselves, in creating the test cases (see figure 4).

In this example all user story and quality characteristic combinatios are carried out by the agile project team members. However, as stated before, it also possible to group certain combinations to a test level which could be executed in parallel to the iteration or after the delivery of the release. For instance, all combinations with the quality characteristic “functionality” in it, could be grouped to a system test level.

User Story	Characteristic	Stakeholder	Damage	Chance of Failure	Risk Class	Intensity	Test Design Technique
US 1	Functionality	Person A	3	3	9	●●●	ECT-MCC
	User-friendliness	Person B	2	1	2	●	SYN
US 2	Functionality	Person A	2	2	4	●●	ECT-MCDC
	Security	Person C	3	2	6	●●	SEM-MCDC
US 3	Functionality	Person A	2	1	2	●	DCoT-EQ
US 4	Performance	Person C	2	1	2	●	EG
US 5	Performance	Person C	1	1	1	●	EG
US 6	Functionality	Person A	2	2	4	●●	ECT-MCDC
	Suitability	Person B	2	2	4	●●	PCT-TDL2
..	..		..	..	..		

Figure 4: Test strategy table[5]**.**

When using these tables one can add other columns to it. Like “Test Cases created” (Y/N), “Test Cases executed” (Y/N) and “Tests Passed” (Y/N). And everone has only to look at the whiteboard to see the (test) status (see figure 5).

User Story	Characteristic	Stakeholder	Damage	Chance of Failure	Risk Class	Intensity	Test Design Technique	Test Cases created (Y/N)	Test Cases executed (Y/N)	Tests Passed (Y/N)
US 1	Functionality	Person A	3	3	9	●●●	ECT-MCC			
	User-friendliness	Person B	2	1	2	●	SYN			
US 2	Functionality	Person A	2	2	4	●●	ECT-MCDC			
	Security	Person C	3	2	6	●●	SEM-MCDC			
US 3	Functionality	Person A	2	1	2	●	DCoT-EQ			
US 4	Performance	Person C	2	1	2	●	EG			
US 5	Performance	Person C	1	1	1	●	EG			
US 6	Functionality	Person A	2	2	4	●●	ECT-MCDC			
	Suitability	Person B	2	2	4	●●	PCT-TDL2			
..	..		..	..	..					

Figure 5: Test progress table.

The test strategy is the foundation for every test action, activity, process or project. It holds the justification of what will be tested and how this will be done. Proper risk analysis drives this justification: “No Risk, No Test”. The test strategy describes the test goals, how they will be approached and how (product) risks are covered. Risks influence priorities in agile methodologies. Areas that carry the greatest risks need to be prioritized highly. Furthermore, risks will trigger the noted flexibility of agile environments in deciding what to do and what not to do in terms of time, costs and quality. The test strategy thus facilitates the whole agile team.

[5] ECT-MCC: Elementary Comparison Test-Multiple Condition Coverage, SYN: Syntactic Test, ECT-MCDC: Elementary Comparison Test-Modified Condition/Decision Coverage, SEM-MCDC: Semantic Test-Modified Condition/Decision Coverage, DCoT-EQ: Data Combination Test-Equivalence Classes, EG: Error Guessing, PCT-TDL2: Process Cycle Test-Test Depth Level 2 Want to know more? Take a glimpse at TMap NEXT (ISBN 90-72194-80-2).

2.4 Reuse and adapt the values of structured test approaches

Many people think that agile projects are chaotic, unorganized and uncontrolled. On the contrary. Agile projects will not be successful when they lack discipline or structure. Because of this prejudice, however, structure is often thrown out as is the baby with the bathwater.

A structured testing approach offers the following advantages:

- It delivers insight into, and advice on, any risks in respect of the quality of the tested system.
- It finds defects at an early stage.
- It prevents defects.
- The testing is on the critical path of the total development as briefly as possible, so that the total lead time of the development is shortened.
- The test products (e.g. test cases) are reusable.
- The test process is comprehensible and manageable.

When implementing an agile method, organizations often find it difficult to decide on what practices to keep and what practices to throw out. Usually, organizations find it easier to start afresh without much consideration for the consequences of throwing out old structures, and very often the baby is thrown out with the bathwater.

When organizations refrain from this decision process, they might save time at the beginning, but in the end the result can be loss of insight into the quality of the tested system, lacking reusability, lacking maintainability, more defects and delays in testing due to an unmanageable test process and most of all loss of all the advantages of an efficient running agile project.

To be able to decide upon which test structures and activities to keep and which to discard, it is necessary to have a good knowledge about the agile method to be implemented. An expert as coach, who can give training and support and who is aware of all the issues coming up during a transition from one method to another is almost indispensable. Usually, many practices and procedures, whether or not adjusted, can be reused.

3. Conclusion

Testing in agile software development environments ensues a different way of working. Moreover, a different mind-set is necessary. It will not be sufficient to follow a list of rules, applicable in an ASDE. A professional agile tester will need a thorough understanding of the agile approach along with a readiness to step out of the test compartment. Doing whatever is necessary in an effective and efficient way, in order to be a valuable agile team member, must be the goal of an agile tester. In general a structured test approach can support the agile tester to achieve this. In this paper the test approach without test levels is discussed in more detail. The characteristics for this approach are:

- No separate test levels, not in the iterations nor afterwards.
- All acceptors must be present in the, relevant, iterations and prepare/execute their own tests (with or without help of the other project members).
- In the product risk analysis the risk per combination of feature and quality characteristic should be determined by and per acceptor.
- The test strategy (= test intensity/test design techniques) is also determined per combination of feature and quality characteristic.
- No more (many pages) test plans needed, but just one table on the whiteboard.
- After the last iteration the product is explicitly accepted by all acceptors.