Certified Practitioner in Agile Testing (CPAT)

Syllabus

Version 1.00

Released 30-01-2020

Copyright Notice

Thank you to the main authors

- Bas Kruip, Joost Voskuil, Klaartje van Zwoll, Jeroen van Seeters, Huib Schoots

Thank you to the co-authors

- Carlo van Driel, Jayapradeep Jiothis

Thank you to the review committee

Alexis Herrera, Alfonso Fernández, Ana Laura Ochoa Moreno, Arun Janglie, Aurelio Gandarillas, Ángel Rayo Acevedo, Christine Green, Daniel Castillo Garcia, Daniel Leo Lopez Romero, Emilie Potin-Suau, Erik van Veenendaal, Fabiola Mero, Gustavo Márquez Sosa, Héctor Ruvalcaba, Isaac Marcelo Malamud Kobrinsky, Ismael Betancourt, Javier Chávez, Javier Jesús Gutiérrez Rodríguez, Jordi Fernandez, José Antonio Rodriguez, Julian Baars, Julie Gardiner,Julio Córdoba Retana, Jochem Gross, Kaan Sanli, Kyle Alexander Siemens, Laksh Ranganathan, Luisa Morales Gómez-Tejedor, Márcia Araújo Coelho, Marco Fidel Peña Valbuena, Marelis V. Pérez García, Mario Alvarez Gómez, Melissa Pontes, Miaomiao Tang, Miguel Angel De León Trejo, Nadia Soledad Cavalleri, Patricia Osorio Aristizabal, Paul Mowat, Richard Seidl, Rogier Ammerlaan, Ruth Margaret Florian Caipa, Sammy Kolluru, Samuel Ouko, Sebastiaan Vreedenburgh, Sergio von Borries, Shashikumar Singh, Silvia Nane, Søren Wassard, Thomas Cagley, Tim Moore, Valeria Cocco, Wim Decoutere & Yaara Egger.

Revision History

Version	Date	Remarks
0.16	December 2019	Initial Beta release
1.00	January 2020	Initial release after reviews
1.01	January 2020	Small review comments

Table of Content

Business Outcomes

Business objects (BOs) are a brief statement of what you are expected to have learned after the training.

BO-1	Understand the principles of Agile and Scrum in general
BO-2	Understand the principles of Agile and Scrum in relation to testing
BO-3	Understand the role of a tester in an Agile environment
BO-4	Have insight into the people skills that are essential to be part of a multi-disciplinary team and to collaborate effectively with developers, analysts and product owners
BO-5	Be able to discuss and practically map risks using multiple techniques
BO-6	Understand the relation between risks, facts, questions, intuition, exploration and testing
BO-7	Define a test strategy tailored to work in an Agile environment
BO-8	Practice creating a test strategy for a real product
BO-9	Understand and use Exploratory Testing to accelerate your testing
BO-10	Apply Exploratory Testing to real software
BO-11	Understand and use Test automation in a sustainable way (maintainable and re-usable)
BO-12	Be able to tell the Testing Story
BO-13	Be able to do effective reporting about testing to management and other stakeholders
BO-14	To be able to improve specifications to add more value
BO-15	Use your critical thinking skills to improve the (quality of) the product and/or processes
BO-16	Understand and practice feedback scenario´s
BO-17	Understand learning and the role of retrospectives in learning
BO-18	Practice learning by applying retrospectives

Learning Objectives/Cognitive Levels of Knowledge

Learning objectives (LOs) are brief statements that describe what you are expected to know after studying each chapter. The LOs are defined based on Bloom's modified taxonomy as follows:

Definitions	**K1 Remembering**	**K2 Understanding**	**K3 Applying**
Bloom´s definition	Exhibit memory of previously learned material by recalling facts, terms, basic concepts, and answers.	Demonstrate understanding of facts and ideas by organizing, comparing, translating, interpreting, giving descriptions, and stating main ideas.	Solve problems to new situations by applying acquired knowledge, facts, techniques and rules in a different way.
Verbs (examples)	Remember Recall Choose Define Find Match Relate Select	Summarize Generalize Classify Compare Contrast Demonstrate Interpret Rephrase	Implement Execute Use Apply Plan Select

For more details of Bloom's taxonomy please, refer to **[BT1]** and **[BT2]** in References.

Hands-on Objectives

Hands-on Objectives (HOs) are brief statements that describe what you are expected to perform or execute to understand the practical aspect of learning. The HOs are defined as follows:

- HO-0: Live view of an exercise or recorded video.
- HO-1: Guided exercise. The trainees follow the sequence of steps performed by the trainer.
- HO-2: Exercise with hints. Exercise to be solved by the trainee, utilizing hints provided by the trainer.
- HO-3: Unguided exercises without hints.

Prerequisites

Mandatory

- None

Recommended

- Some Agile or Scrum certificate like PSM or CSM or ASF or at least read the Scrum guide.
- Basic knowledge (ISTQB-CTFL or ISTQB-CFTL-Agile Tester) of testing in general.
- Having at least 1 year working experience in Agile and in testing.
- Read a book about Agile Testing like ´Agile Testing´ **[JL1]** and ´More Agile Testing´ **[JL1].**

Chapter 1 - Introduction to Agile Testing

In the introduction, we will define the terminology surrounding testing, risks, Agile, Scrum and Agile Testing.

Keywords

Testing, Product Risk, Project Risk, Quality Risk, Agile, Manifesto, Known knowns, Unknown knowns, Known unknowns, Unknown unknowns, Agile Testing, Scrum

LO-1.1	K1	Baseline the definition of testing
LO-1.2	K1	Recall the structure of various risks
LO-1.3	K1	Recall the Agile Manifesto
LO-1.4	K2	Explain the meaning of the Agile Manifesto and principles
LO-1.5	K2	Rephrase the Agile Manifesto into risks, based on Rumsfeld's "There are known knowns"
LO-1.6	K2	Understand and explain the definition of testing and especially Agile Testing
LO-1.7	K2	Understand the Scrum framework
HO-1.1	HO-3	Share the current definitions of Scrum among the group
LO-1.8	K2	Explain the Scrum framework and the role of a tester in the team
LO-1.9	K1	Recall the roles of Product Owner, Scrum Master and Scrum Team
LO-1.10	K1	Recall the use of story points and estimation and test points vs. including test estimates
LO-1.11	K2	Explain the testing perspective in planning poker and the test role in sprint planning
LO-1-12	K1	Recall velocity driven and commitment driven planning
LO-1.13	K2	Understand test estimates
LO-1.14	K2	Explain the test role in refinement sessions
LO-1.15	K2	Explain the test role in retrospectives
LO-1.16	K2	Understand the test role in a Program Increment / Release planning
LO-1.17	K1	Learn some commonly used Scrum anti-patterns like N+1 test iteration or hardening iterations to resolve technical debt
LO-1.18	K2	Understand the shift of the tester role in an Agile context
LO-1.19	K2	Be able to explain what the testers' role is in an Agile team
LO-1.20	K2	Understand the role of a tester as the key quality driver in an Agile team
LO-1.21	K2	Understand the skills that are essential for a tester in an Agile context
HO-1.2	HO-3	Get the definitions from the students to start a discussion
LO-1.23	K2	Demonstrate the definition of critical thinking
LO-1.24	K2	Demonstrate the definition of a claim or assertion
HO-1.3	HO-1	Practice with the claims, conclusions and constructions to understand the complexity
LO-1.25	K2	Understand and recognize fallacies
LO-1.26	K2	Understand thinking errors and how they influence people´s judgement
LO-1.27	K3	Apply all the definitions that play a role in thinking critical
HO-1.4	HO-2	Practice critical thinking by using argumentation, claims, constructions, fallacies and thinking errors
LO-1.28	K2	Understand the definition of a retrospective
LO-1.29	K2	Explain the purpose of retrospectives
LO-1.30	K2	Demonstrate the need of retrospectives and the role of a tester
LO-1.31	K3	Be able to implement retrospectives in a team

LO-1.32	K1	Be able to select different types of retrospectives
HO-1.5	HO-3	Define how a serious game can be used for a retrospective

1.1 Definition of Testing

LO-1.1	K1	Baseline the definition of testing
LO-1.2	K1	Recall the structure of various risks

Testing is giving **insights** on **risks**, where risk is any threat to the **customer value** of the product to be delivered.

The risks can either be product risks that threaten the value of the product or project risks that threaten either the speed of delivering the product or in the end the product itself.

1.2 What is Agile?

LO-1.3	K1	Recall the Agile Manifesto
LO-1.4	K2	Explain the meaning of the Agile Manifesto and principles

Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment **[AG1]**.

The Agile manifesto **[AM1]** consists of 4 values and 12 principles. Explained and discussed are the 4 values:

"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

"That is, while there is value in the items on the right, we value the items on the left more"

Explained and discussed are the 12 principles as they are evenly important to get people to the right mindset:

"We follow these principles"

1. Our highest priority is to satisfy the customer through the 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 their needs 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. A 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 behavior accordingly.

1.3 Link Agile to Testing and Risks

LO-1.5	K2	Rephrase the Agile Manifesto into risks, based on Rumsfeld's "There are known knowns"

Rumsfeld stated: ´Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns— the ones we don't know we don't know. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones' **[RF1]**

The idea of unknown unknowns was created in 1955 by two American psychologists, Joseph Luft (1916-2014) and Harrington Ingham (1916-1995) in their development of the Johari window. They used it as a technique to help people better understand their relationship with themselves as well as others.

These ideas are linked to important pillars in Agile Testing:

- Facts
- Questions
- Intuition/The unconscious
- Exploration

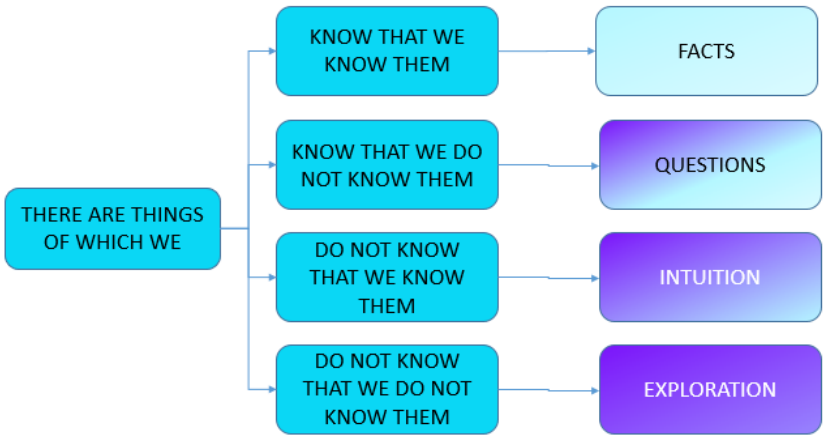

Figure 1

What is valid for agile in general is of course also valid for testing. Here too all areas must be covered.

Risks are (properties of) 'things' that can present a danger / pose a threat to the value of the products to be delivered. A testers job is to identify those dangers and threats. The total field of those ´threats´ consists of 4 subcategories.
These all in a pure sense require a different approach: in a broader sense, those things that we know of require CHECKS. Things we don't know about require TESTS. These approaches can again be divided into many test approaches / techniques or operational choices.

1.4 Definition of Agile Testing

LO-1.6	K2	Understand and explain the definition of testing and especially Agile Testing

There have been many definitions of Agile Testing (C. Kaner, M. Bolton, J. Bach, L. Crispin, E. Hendrickson, J. Gregory and many more) and discussions about these definitions. Most definitions are centered around testing or testers as a separate role in developing software. Two very important missing elements are added to this definition: the team's responsibility in Quality as well as the four types of skills (People skills, Agile skills, Test skills, Technology skills) testers should possess.

The definition consists of 3 parts: **why** do we do what we do, **what** do we do and **who** does it.

Why:

- Help the client in the process of reviewing his/her product by providing as much insight as possible into the workings and the risks involved in the usage of the product.
- Help the team in producing a constantly better product, one that meets the expectations and intent of the client.

What:

- Developing the awareness to quality in the team by actively helping to continiously improve quality.
- Performing tests of the criteria that are not clearly established or yet fully determined by experimenting and researching using tests methods such as Exploratory Testing, Pair Testing and Bug Hunting.
- Performing (automated) checks based on established criteria using testing methods such as Unit testing, Integration Testing, Security Testing, Performance Testing and Compliance Testing.

Who

- People who have the right people skills, agile skills, test skills, technology skills, and who have the motivation and credibility and are able to obtain, retain and improve these skills, motivation and credibility.

Detail on the "Who":

- Motivation: a tester needs to be motivated to learn by improving the skills. Without being motivated yourself you can not improve your team.
- Credibility: by delivering value that is not directly a visible product (insight in quality) a tester in an Agile context needs credibility in and outside the team in order to do a good job. If you have no credibility you are constantly defending what you are doing instead of adding value.

- People skills: in a team a tester needs many people skills in order to do a good job as a tester in an Agile context. You have to be able to, for example, give good feedback, ask difficult questions, think critically and tell the testing story. You need to be able to communicate to stakeholders, business, clients, and also to developers and people from operations.
- Agile skills: a tester in an Agile context needs the agile mindset to keep focus on delivering value for your customers. This means you need to understand agile and the software development lifecycle and any methods used to deliver the value (Scrum, XP, Kanban, DevOps, ...).
- Test skills: insights on the status of the product need to be available instantly and constantly. This requires a really different test strategy and different test methods. A tester in an Agile context needs a broad range of methods in her/his toolbox in order to use the most appropriate one in the specific context of the project and product.
- Technology skills: a tester in an Agile context needs to know the technology of the product and development environment. You need to be able to read code, write automated tests on diferent levels and define risks based on used technology.

A visual representation is made by Karen Greaves and Sam Laing **[GA1]**

Figure 2

1.5 Scrum

LO-1.7	K2	Understand the Scrum framework
HO-1.1	HO-3	Share the current definitions of Scrum among the group
LO-1.8	K2	Explain the Scrum framework and the role of a tester in the team
LO-1.9	K1	Recall the roles of Product Owner, Scrum Master and Scrum Team
LO-1.10	K1	Recall the use of planning estimation (like story points) and test points vs including test estimates
LO-1.11	K2	Explain the testing perspective in planning poker and the test role in sprint planning
LO-1-12	K1	Recall velocity driven and commitment driven planning
LO-1.13	K2	Understand test estimates
LO-1.14	K2	Explain the test role in refinement sessions

LO-1.15	K2	Explain the test role in retrospectives
LO-1.16	K2	Understand the test role in a Program Increment / Release planning
LO-1.17	K1	Learn some commonly used Scrum anti-patterns like N+1 test iteration or hardening iterations to resolve technical debt

There are many agile frameworks: XP, Scrum, Kanban, Lean, Crystal and many more. Scrum is used during the training as being the most used framework.

There are many different implementations and variations of the most commonly used Agile framework: Scrum. People often start making changes to the original framework even before they start using it. This often results in non-Agile ways of working and should not be called ´Agile´. This information is helpful and needed to understand the role of a tester in an Agile team.

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk. Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation **[SC1]**.

The different roles in a typical Scrum team are recalled: Product Owner, Scrum Master and Team Member and their responsibilities.

The different events in Scrum are recalled: daily Scrum, product backlog refinement, sprint planning, sprint review, retrospective.

A closer look is taken at work estimation according to the sprint planning sessions. Explained is the role of the most commonly used estimation method using story points versus hour estimation. Also the option to have separate test points and development points and join them to get the estimate is discussed, as well as the option to include the testpoints without explicitely mentioning them. By splitting the roles and points the estimation seems easier; however, it creates less opportunities to learn from each other and discuss about the story (which is the whole point of doing estimation). The role of the team member with testing skills in sprint planning is also to challenge the team and keep them alert as well as asking critical questions about the stories. Another role of this is to get the team to an understanding that making a small program change can sometimes have big consequences for testing and the test estimates. We also mention 2 commonly used ways to plan a sprint: velocity driven and commitment driven. They slightly differ; however, a sprint planning is mainly about consensus, and not about planning.

A tester's mindset is really valuable in refinements to bring in the critical thinking aspect and a different approach to a story (what could potentially go wrong versus how we can make it work). If in an organization it is not common practice to involve a tester in refinements, action should be taken to convince the team to involve all team members including the tester in the refinement.

The retrospective is a very important event in Scrum. It brings the possibility to reflect and learn. This also means a team will need a fail-safe environment as team members share the things that probably failed in order to learn from it and do better next time. This means that people will need the confidence that they can take a vulnerable position and most people find that difficult. As testers, we have no special role in a retrospective; we can help our team though by taking the vulnerable position first and share a (testing) experiment we performed and that failed and what we learned from that.

If a team is in an Agile at scale environment, there will be Product Increment sessions in which a tester can add a lot of value from the testing perspective towards business goals and making sure organizational readiness is there from a test perspective. Also, a tester can give input if the tester is aware of people´s planning bias.

The Scrum part is finished with some commonly used practices that are considered anti-patterns of Agile and Scrum:

- The N+1 Iteration or lagging test iteration: in iteration N, the development is performed and the testing of that development is completed in the N+1 iteration. The fixes are most likely done in the N+2 iteration.
- Hardening iterations, as a result of build-up technical debt: hardening iterations are used to fix low quality-built solutions (technical debt) in previous iterations. If refactoring is part of every sprint iteration, there is no need for hardening iterations.
- Mini-waterfall: instead of real Scrum, the team designs, builds and tests during an iteration according to the iteration length waterfall scenario.

1.6 Tester in an Agile context

LO-1.18	K2	Understand the shift of the tester's role in an Agile context
LO-1.19	K2	Be able to explain what the testers' role is in an Agile team
LO-1.20	K2	Understand the role of a tester as the key quality driver in an Agile team
LO-1.21	K2	Understand the skills that are essential for a tester in an Agile context

The differences between the role of a tester in a more traditional environment and the role of a tester in an Agile context are explained. The role has shifted from a functional tester of requirements (performing mainly checks) to a ´quality engineer´, which partly requires different skills (as mentioned before).

In Agile, quality is a team responsibility and if the team doesn't pick up that responsibility, or doesn't understand it, the tester is responsibile for helping the team in understanding and implementing this responsibility. The tester in an Agile context must be the person who makes the team aware of quality. Examples:

- A tester in an Agile context motivates product owners/customers to be concise about what they want.
- A tester in an Agile context pairs with Product Owner/stakeholders from the start to make user stories as specific as possible.
- A tester in an Agile context coaches developers about the value of good coding practices including unit testing.

A tester in an Agile context is a quality driver/architect in the team as she/he has the natural quality mindset.

In order to be the quality engineer, a tester in an Agile context needs specific skills, such as people/communication skills to bring messages/new ideas to the team and technology skills to help developers with creating good unit tests and test skills to explore the unknown of the application and many more skills of which the most important ones are praticed in this training.

1.7 Critical thinking

HO-1.2	HO-3	Get the definitions from the students to start a discussion
LO-1.23	K2	Demonstrate the definition of critical thinking
LO-1.24	K2	Demonstrate the definition of a claim or assertion
HO-1.3	HO-1	Practice with the claims, conclusions and constructions to understand the complexity
LO-1.25	K2	Understand and recognize fallacies
LO-1.26	K2	Understand thinking errors and how they influence people´s judgement
LO-1.27	K3	Apply all the definitions that play a role in thinking critical
HO-1.4	HO-2	Practice critical thinking by using argumentation, claims, constructions, fallacies and thinking errors

Critical thinking (or Scepticism) is generally a questioning attitude or doubt towards one or more items of putative knowledge or belief or dogma. It is often directed at domains, such as the supernatural, morality (moral scepticism), theism (scepticism about the existence of God), or knowledge (scepticism about the possibility of knowledge, or of certainty).

Claim and assertions are being discussed as being a meaningful statement that is:

- True
- Not true
- Neither ´true´ or ´not true´

Argumentation and reasoning are being discussed as being a construction of claims that cause a conclusion.

A conclusion is valid if the denial of the conclusion is in contradiction with one of the used claims in the construction. An example:

- Claim 1: “Jan is a test analyst.”
- Claim 2: “All test analysts are critical thinkers.”
- Conclusion: “Jan is a critical thinker.”

The conclusion is valid as you can not deny it based on claim 1 and claim 2. The conclusion is however not true because at least 1 claim is neither ´true´ or ´not true´.

Fallacies are defined and demonstrated as people often use fallacies to convince other people to do what someone wants them to do instead of doing the right thing. A tester needs to learn how to recognize fallacies as they play an important role in being critical. Being critical is one of the most important values that a tester can bring to the team.

People keep considering errors **[KM1]** as arising from an irrational judgement. Conclusions are drawn irrationally. People continue on the road to critical thinking although it is a painful and difficult trip. Shared is why not being critical looks way easier (however in the end it is not).

The trip is finished by giving multiple examples of how to become a critical thinker and practice being critical with an exercise by using a serious game.

1.8 Retrospectives

LO-1.28	K2	Understand the definition of a retrospective
LO-1.29	K2	Explain the purpose of retrospectives
LO-1.30	K2	Demonstrate the need of retrospectives and the role of a tester
LO-1.31	K3	Be able to implement retrospectives in a team
LO-1.32	K1	Be able to select different types of retrospectives
HO-1.5	HO-3	How can a serious game be used for a retrospective

The retrospective is a very important event in Scrum. It brings the possibility to reflect and learn. This also means a team will need a fail-safe environment as team members share the things that probably failed in order to learn from it and do better next time. This means people need the confidence that they can take a vulnerable position and most people find that difficult. As testers, we have no special role in a retrospective; we can help our team though by taking the vulnerable position first and share a (testing) experiment we performed and that failed and what we learned from that.

Continuous learning plays a key role in Agile. As a tester, you need to understand the role of a retrospective and be able to perform a retrospective as well.

The definition of a retrospective according to the Scrum guide is: “The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint”.

The definition is discussed in detail to explore the real meaning of the highlights in this definition:

- Opportunity
- The Scrum team inspects itself
- Create a plan

By discussing the meaning of retrospectives, we come to their actual purpose : to be critical of yourself and others in order to learn and do better next time (continuous improvement).

A variety of different retrospectives is shown in order to produce different kinds of improvement focus – it could be either team work, quality, tooling, communication, etc. – and to keep retrospectives interesting.

Chapter 2 – Start Testing – A Case

The hands-on training is built around a case that is used to explain the theory. The case contains real software (and optional hardware) **[SH1]** to be tested.

Keywords

Agile, Testing, Test strategy, Questions, Feedback, Critical thinking, Risks, Riskstorm, Exploratory Testing, Headline game, Test automation, Check automation, Oracles, Heuristics, Mnemonics, Product Outline, Project Outline, Issues

HO-2.1	HO-3	Start testing unknown software that optionally runs on specific hardware
LO-2.1	K3	Optimal use of the given situation, documentation and other information at hand
LO-2.2	K1	Recall the need of asking questions before actually doing something in order to understand the specific situation
LO-2.3	K1	Recall the need of asking questions before actually doing something in order to understand the specific needs
LO-2.4	K1	Recall the need of asking questions before actually doing something in order to start your test in a useful direction
HO-2.2	HO-3	Test a specific subject (something that looks like a ball)
LO-2.5	K2	Understand the need of asking questions
LO-2.6	K2	Understand the value of questions in general
LO-2.7	K2	Understand the value of questions in relation to testing
LO-2.8	K2	Understand the definition of feedback
LO-2.9	K2	Understand the meaning and value of feedback
LO-2.10	K2	Rephrase feedback to testing
LO-2.11	K2	Understand the ground rules of feedback
LO-2.12	K3	Apply 4 different methods of feedback
HO-2.3	HO-2	Practice feedback in a real-life testing situation
LO-2.14	K2	Understand what models are and how we use models (Mental, Conceptual and Real Things/World)
LO-2.15	K3	Apply models to software development and testing in specific
HO-2.4	HO-3	Create a project outline and present it to everybody
LO-2.16	K2	Understand the definition of a project outline
LO-2.17	K3	Apply the project outline to your project
LO-2.18	K2	Understand the relation between a project outline and testing
LO-2.19	K1	Remember to ask questions to get the needed information for the project outline
LO-2.20	K3	Apply asking questions to get a good project outline
LO-2.21	K3	Use the mnemonic PROJECT to help us create the project outline
HO-2.5	HO-3	Create a product outline by exploring the product (Case software/hardware)
LO-2.22	K2	Understand the definition of a product outline
LO-2.23	K3	Apply the product outline to your project
LO-2.24	K2	Understand the relation between a product outline and testing
LO-2.25	K1	Remember to ask questions to get the needed information for the product outline
LO-2.26	K3	Apply asking questions to get a good product outline
LO-2.27	K3	Use PRODUCT to help us create the product outline

2.1 Test the case software

HO-2.1	HO-3	Start testing unknown software that optionally runs on specific hardware
LO-2.1	K3	Optimal use of the given situation, documentation and other information at hand

Real life projects are generally ongoing when a new team member gets involved. As new member you need to deal with already existing software and environments, and there is usually little time and no existing documentation, or an outdated one, on how the software is supposed to work.

By means of an actual project testing situation, you, as a tester, learn how to start your testing by using the information at hand and by asking questions to obtain the information you require/need.

2.2 Questions

LO-2.2	K1	Recall the need of asking questions before actually doing something in order to understand the specific situation
LO-2.3	K1	Recall the need of asking questions before actually doing something in order to understand the specific needs
LO-2.4	K1	Recall the need of asking questions before actually doing something in order to start your test in a useful direction
HO-2.2	HO-3	Test a specific subject (something that looks like a ball)
LO-2.5	K2	Understand the need of asking questions
LO-2.6	K2	Understand the value of questions in general
LO-2.7	K2	Understand the value of questions in relation to testing

Before testing something, the tester needs to get as much information about it as possible:

- Who is asking you to test the subject and what is his/her relation to the subject?
- What do you think the subject is and what does that mean to you?
- What is the intended use of the subject?
- What should you test, what is important, where are potential risks?
- Was it already tested before and are there any results of these tests available?
- What has been changed to the subject recently?
- How much time is available?
- Who is/was involved and can be consulted for more information?
- Many more questions can be asked depening on the context and available information.

The need is explained to understand why, who and when questions should be asked:

- To get more information about anything, at any time.
- When things are unclear to you.
- What you think is true does not have to be true.
- What you know does not have to be complete.
- To check whether you understand the real risks.

2.3 Feedback

LO-2.8	K2	Understand the definition of feedback
LO-2.9	K2	Understand the meaning and value of feedback
LO-2.10	K2	Rephrase feedback to testing
LO-2.11	K2	Understand the ground rules of feedback
LO-2.12	K3	Apply 4 different types of feedback
HO-2.3	HO-2	Practice feedback in a real-life testing situation

The definition of feedback is: "(Re)acting on changeable behavior or changeable products and processes".

A tester provides feedback which is the most important result of the work done. The challenge is that this feedback is usually not a positive feedback. Usually, testers find issues on the quality of what someone else has created. The non-positive feedback can feel negative towards the creator of the work.

For that reason, it is important for a tester to be able to give feedback in the most constructive way, in order to obtain the best results. The ground rules of feedback are given and 4 different ways of giving feedback are provided so you can choose the way that, in your specific situation, has the highest change of being successful:

- Corrective feedback; behavior, feelings, consequences, desired behavior.
- Evaluating feedback; the Pendleton method **[PD1]**.
- Activating feedback; the hamburger method **[HB1]**.
- Motivating feedback; a compliment

All 4 ways of providing and receiving feedback are practiced.

2.4 Models

LO-2.14	K2	Understand what models are and how people use models (Mental, Conceptual and Real Things/World)
LO-2.15	K3	Apply models to software development and testing in specific

Models are a simplification of the real world. Models can be made for everything around us; a new product such as a car or a house, but also models from an event that you organize, like a birthday party.

People use and need models because the real world is way too complicated. Just try to imagine a birthday party and write down everything that is related to that birthday party. You can probably come up with people, cake, drinks, etc., however, you will never be sure of who will actually attend until they are really there. Neither do you know for sure what they want to drink....

So, people use models to deal with situations that are too complicated to handle. Creating software and also testing software is such an activity. It is impossible to deal with complexity without using all

kind of models (Mental, Conceptual and Real Things/World) to bring some order in the chaos. Some examples of models are f.e. a database models and user models.

Testers (people in general) have a broad set of preconceived models in their mind. For software development and testing, we can mention mental models, technical models, domain models and in general experiential models.

When testing software, testers link what they see (in the product or project or documentation) to their own models and they draw conclusions from that. The deeper we connect the model to what we see, the more we focus. People can, however, not be focussed all the time, and focussing also narrows down the scale of what can be seen , so we need to be aware that we also need to de-focus to see the bigger picture.

2.5 Test strategy global overview

In order to create the best insights in the status of the product, the team needs an approach on quality. In the syllabus this is referred to as the test strategy. The test strategy consists of different parts of information:

- A project outline to get and share information about the environment.
- A product outline to get and share information about the product itself.
- A risk outline to get and share information about potential risks.
- A test outline to get and share information about the choosen testing approach:
    - What kind of tests are performed.
    - Which tests are done when.
    - Test environments and tooling.
    - Reporting.

Please note that a test strategy is an incremental product. It develops over time. There is no need to be complete from the start. Also note that all parts in a test strategy should not be a duplication of already existing products. In that case, put in a reference.

The purpose of a test strategy is not only to get information, but also to share and discuss any quality and test related subjects within (and potentially also outside) the team. A test strategy is a real-life reflection on the project, the product, the status of the product, the risks and the way the team deals with quality and testing.

It is highly recommended to visualize the test strategy to keep it alive and constantly remind the team of the need for quality and be aware of elements that can influence the quality.

2.6 Project outline

HO-2.4	HO-3	Create a project outline and present it to everybody
LO-2.16	K2	Understand the definition of a project outline
LO-2.17	K3	Apply the project outline to your project
LO-2.18	K2	Understand the relation between a project outline and testing

LO-2.19	K1	Remember to ask questions to get the needed information for the project outline
LO-2.20	K3	Apply asking questions to get a good project outline
LO-2.21	K3	Use the mnemonic PROJECT to help us create the project outline

Each project and everything in its environment can have an impact on the decision to test or not test something. As a tester in an Agile context, you need to be able to create a project outline to get insights in the project in order to make decisions, together with the team, on what to test or what may not need to be tested. A lot of this information is hidden in culture, people, history, customers, implicit knowledge and so forth. Testers needs to be able to ask the right questions to clarify this information so we can deploy the right strategy and approach on testing.

The mnemonic PROJECT helps you to create a project outline and contains the following information:

Purpose Mission and Vision
Relations Scrum-team / Development-team
Objectives Key performance indicators
Jeopardy Chances for the project and threats to it
Environment Technical environment (tools, hardware, software, information (requirements, etc.))
Clients Stakeholders, Sponsors, Tribes
Team rules Team values, Working Agreements

Purpose
- Vision
  - How does the product change the world of users (e.g. less repetitive work)?
  - Which problem is solved / which benefit is provided, and for whom?
- Mission
  - Defines the users of the product to be created.
  - Defines the actions and final results of the team.
  - Defines which product or service the team will deliver.
  - Defines the attributes of the product or service that describes the added value.
  - Defines the generic value of the product for the customer.

Relations
- What does each team member bring?
- What makes every team member happy?

Objectives
- KPIs that are clear to the team
  - External: on July the 1st, 2020 (after iteration 5) the new software will reduce the occupancy of the customer service desk by 20% in two months.
  - Internal: on March the 1st, 2020, team production increased by 10% (N story points per iteration from 100 to 110).

Jeopardy
- Although the entire “project” only takes 2 to 3 days, you can also look at opportunities and risks in such a short period:
  - Is the PO / Customer / Sponsor available during those 3 days?
  - Is there only 1 test object? What if it breaks down?
  - Shall we focus on deal breakers for the product, so that we might be able to finish a lot sooner (and do we want that?)

Environment
- Instructions.
- The software.

- Any specific hardware.
- Any tools you might use or have to use.

Clients

- Stakeholders: the client: what is her/his interest?
- Sponsors: who in this project wants tests to be carried out? Is this someone other than the stakeholder?
- Teams: are other teams or your own team perhaps clients as well?

Team rules

- Team rules
    - Examples
        - Openness: “Speak out about topics that concern yourself or the team.”
        - Courage: “Communicate openly and put everything on the table.”
- Work agreements
    - Examples
        - During meetings, we stick to the subject.
        - We determine in this order: experts first, sociocratic (no objection), consensus (everyone in favor), democratic.
        - Learning: we fail fast, a lot and identify our mistakes as soon as possible.
        - We respect our deadlines. In the event of force majeure, we will inform the whole team and the stakeholders.

2.7 Product outline

HO-2.5	HO-3	Create a product outline by exploring the product (Case software/hardware)
LO-2.22	K2	Understand the definition of a product outline
LO-2.23	K3	Apply the product outline to your project
LO-2.24	K2	Understand the relation between a product outline and testing
LO-2.25	K1	Remember to ask questions to get the needed information for the product outline
LO-2.26	K3	Apply asking questions to get a good product outline
LO-2.27	K3	Use PRODUCT to help us create the product outline

In order to understand what we are testing, we need an overview of the product itself. A tester needs to create a map to help him/her understand the product. If we do not understand the product, we cannot test it properly.

A helpful mnemonic to help us is PRODUCT:

Platform	What the product runs on (technically)
Relations	How we (or another system) interact with the system
Operate	How the product operates
Data	What the product processes
Users	All the elements of the product
Construction	How the product is used
Time	Relationships between the product and time

Platform

- Hardware
- Middleware

- Software

Relations

- User interfaces
- System interfaces
- API interfaces
- Import/Export

Operate

- Application: any function that defines the product.
- Calculation: any mathematical function in the product.
- Security: user rights, data security, encryption, front and back end security, vulnerabilities in subsystems.
- Transformations: font settings, inserting clip art, money transfers, etc.
- And more

Data

- How does the product handle data?
  - Input / output
  - Persistent
  - Invalid / Noise
  - Lifecycle (CRUD)
  - And more

Users

- Users
- Environment (physical, light, distractions, noise)
- Common use (patterns, sequences or input)
- Disfavored use (mistakes, ignorance, malicious, stupid use)
- Extreme use

Construction

- Code: from executables to individual routines.
- Hardware: every hardware component integral to the project.
- Non-executables: all files other than multimedia or programs, such as sample data, help files.
- Collateral: everything beyond the above: web links and content, packaging, licenses, etc.

Time

- Input output delays and intervals.
- Fast / Slow: input
- Changing rates: spikes, bursts, hangs, bottlenecks, interruptions.
- Concurrency: multi-user, time sharing, threads, shared data.
- Time-out settings, periodicals, time zones, company holidays, guarantee periods, chrono functions.

Chapter 3 – Risks

"Risk" is the keyword for any test strategy. However, in an Agile environment, the risks are changed and coming from more different sources than before. This requires a different strategy of dealing with risks.

Keywords

Agile, Testing, Test strategy, Critical thinking, Risks, Riskstorm, Headline game

LO-3.1	K2	Understand what is important in a product and what is not in relation to risks
LO-3.2	K2	Understand that a risk is only valuable if it is specific enough
LO-3.3	K2	Understand the definition of a risk
LO-3.4	K2	Understand the challenges when it comes to defining risks
LO-3.5	K2	Be able to explain the different types of risks
LO-3.6	K2	Be able to separate the different types of risks
LO-3.7	K2	Understand how risk of type A can influence risks of type B
LO-3.8	K2	Understand the role of a tester in relation to risks and the product
LO-3.9	K2	Understand what an oracle is
LO-3.10	K2	Be aware of oracles
LO-3.11	K1	Understand the value of oracles in testing
LO-3.12	K1	Understand the risk of oracles based on invalid principles or mechanisms
LO-3.13	K2	Understand the 2 different approaches towards risk analysis
LO-3.14	K3	Apply the best approach (combination as well) to your situation
HO-3.1	HO-2	Perform a risk analysis on the Case product based on the given methods
HO-3.2	HO-2	Do a risk nightmare headline game on the Case product
HO-3.3	HO-2	Present your risk analysis headline game to the group
LO-3.15	K3	Be able to execute a headline game
HO-3.4	HO-2	Execute a Testsphere riskstorm with hints by the trainer
HO-3.5	HO-2	Present your risk analysis riskstorm to the group
LO-3.16	K1	Recall the different approach between Agile and waterfall
LO-3.17	K1	Relate Agile to risk assessment
LO-3.18	K2	Demonstrate the continuous risk assessment cycle

3.1 Risks

3.1 Definition of risk

LO-3.1	K2	Understand what is important in a product and what is not in relation to risks
LO-3.2	K2	Understand that a risk is only valuable if it is specific enough
LO-3.3	K2	Understand the definition of a risk
LO-3.4	K2	Understand the challenges when it comes to defining risks

To identify risks, a team needs to know what is important and what is not. Something that is not important is not a risk.

Anything that is important can be a risk, however not everything important can or will or must be tested to mitigate a risk.

A risk is only a valuable risk if it is specific enough. For example: it is a risk if the app is too slow. This is a risk; however, it is not very useful for testing. What is "too slow – is it the entire app, or just specific parts?

This means there are a few difficult questions to be asked while testing a risk:

- What is a threat to the value of the product?
- Who are the persons who matter?
- How do we determine the actual probability?
- How do we determine its impact?

In order to speak the same language in a team, we need to start with a clear definition of the used terminology. The next definition of a risk is used:

"Risk is anything that threatens the value of a product to a person that matters**[JW1]**.

Risk is a combination of the probability that the risk may occur and the impact it will have if becoming a reality".

3.2 Type and area of risks

LO-3.5	K2	Be able to explain the different types of risks
LO-3.6	K2	Be able to separate the different types of risks
LO-3.7	K2	Understand how project risks can influence product risks

There are different types of risks:

- Project risks for which we can use PROJECT (project outline) as a base.
- Product risks for which we can use PRODUCT (product outline) as a base.

Risks can influence other risks, also across different types of risks. Example: having too many inexperienced developers is a project risk. This risk is likely to become a product risk as well. Inexperienced developers are more likely to make mistakes due to their lack of experience. This threatens the value of the product and probably requires more testing effort.

3.3 Risk Coverage

LO-3.8	K2	Understand the role of a tester in relation to risks and the product

There are 3 important statuses of a product that relate to risks:

- What we already know.
- We know everything there is to know.
- We know enough to make an informed decision about the product.

Teams, and testers especially, have to ask questions, trust their intuition, and explore. Why? Because of the RISK GAP: it is the task of every team member (and the tester being the driver) to ensure that we know as much as possible about the (to be built) product (and project) so that we can obtain sufficient insight into the associated risks. So that ultimately, based on the resulting test measures, an 'informed decision' can be made about the follow-up.

Figure 3

3.4 Oracles

LO-3.9	K2	Understand what an oracle is
LO-3.10	K1	Be aware of oracles
LO-3.11	K1	Understand the value of oracles in testing
LO-3.12	K1	Understand the risk of oracles based on invalid principles or mechanisms

An oracle is a principle or mechanism by which people recognize a problem **[BO1]**. In testing that is usually our expected result **[TE1]**.

Oracles are very powerful for testers, if we are aware of them, as they help us recognize potential issues that threaten the value of the product: risks!

Oracles can also be ´dangerous´ if your used principles or mechanisms are invalid. This will result in many ´problems´ which are not a real problem.

3.5 Inside out versus outside in risk analysis

LO-3.13	K2	Understand the 2 different approaches towards risk analysis
LO-3.14	K3	Apply the best approach (combination as well) to your situation

Risk analysis can be approached in 2 different ways. Usually, both ways need to be combined to be complete in your risk assessments:

- Inside out: we start with the knowledge available from the people involved
- Outside in: we start from generic quality attributes and risk checklists

Inside out:

Experts and stakeholders make a list of potential vulnerabilities on the component level and what could go wrong on each component. They also think about which input situation could lead to the component failing and finally they think about who the victims of this failure could be.

Teams need to have specific (technical) knowledge about the system to take the inside out route.

Outside in:

Instead of opting for the in-detail level, you keep the detail to much more generic quality attributes and generic risk lists or user stories. This will result in a different list of potential risks. If you do not (yet) have more specific technical knowledge available, this method is the starting point.

3.6 Risk analysis: headline game

HO-3.1	HO-2	Perform a risk analysis on the Case product based on the given methods
HO-3.2	HO-2	Do a risk headline game on the Case product
HO-3.3	HO-2	Present your risk analysis nightmare headline game to the group

An outside-in risk analysis is performed based on the case by using the Nightmare Headline Game **[EH1]**. This method is based on defining risks on what the team/organization does not want to be in the newspapers in the morning about the software the team just released yesterday.

Step 1: Brainstorm a list of serious failures
Step 2: Choose a risk to work on
Ask the group to scan the list looking for a nightmare that stands out as:
- Plausible
- Software-related
- Interesting

Step 3: Brainstorm contributing causes
Step 4: Refine causes into test cases
Step 5: Lather, rinse, repeat

3.7 Risk analysis: riskstorm using Testsphere©

HO-3.4	HO-2	Execute a Testsphere riskstorm with hints by the trainer
HO-3.5	HO-2	Present your risk analysis riskstorm to the group

After the headline game, a completely different way to determine risks is used by using (inside-out) riskstorm. We use the Testsphere **[TS1]** deck of cards. This serious game will help you identify risks. We introduce the light blue (quality criteria) cards and the pink (heuristics) cards in order to determine and later zoom in on the risks.

Figure 4

3.8 Risks in Agile: the continuous risk assessment cycle

LO-3.16	K1	Recall the different approach between Agile and waterfall
LO-3.17	K1	Relate Agile to risk assessment
LO-3.18	K2	Demonstrate the continuous risk assessment cycle

In an Agile environment, assessing risks is a continuous process. Teams cannot do just 1 product risk analysis and use that throughout the whole project. Each time you create or modify or test something, you will get new information about the product and about potential risks. These are the actual risks. Based on the results, the team can ship the product or do some fixing and/or do some more testing. This cycle will run forever while the team is delivering software.

The level on which the team goes through the cycle usually matches the cycle of delivering the product to production. However, any cycle time is ok: from release or iteration until user story or feature or...

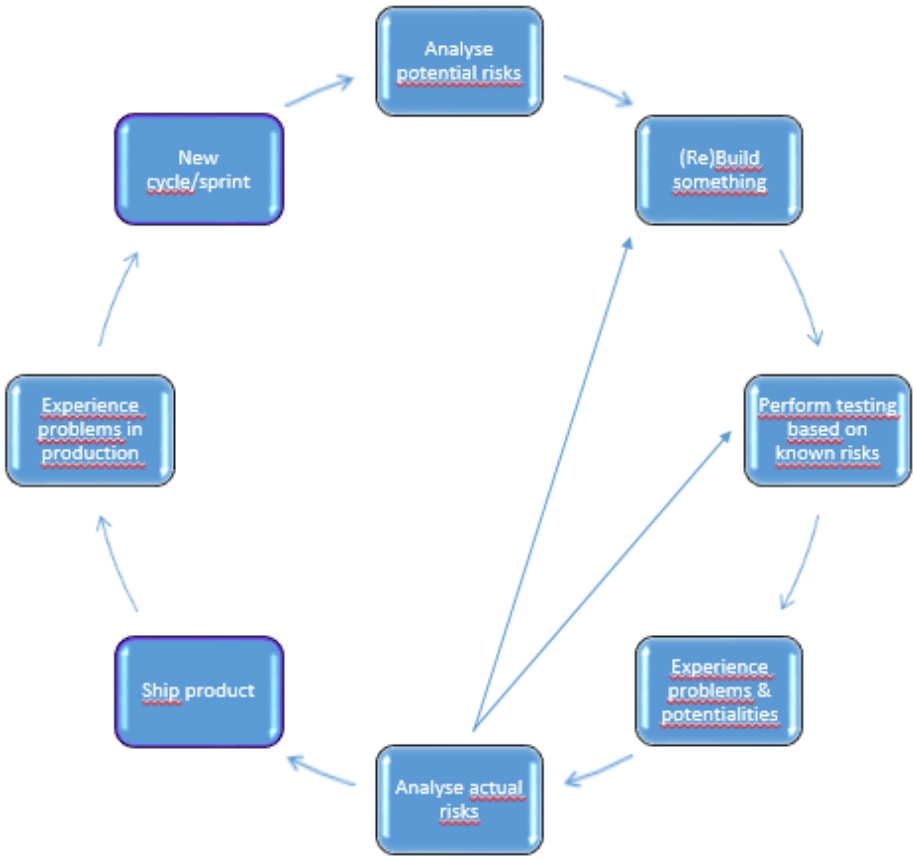

Figure 5

- New cycle/iteration: start of the (new) cycle. Level can be any (release, iteration, feature, ...).
- Analyse potential risks: use a risk analysis tool of your choice to get an insight in any potential risks.
- (Re)build something: team members with a developer role will create the software.
- As soon as possible the new component is delivered to perform testing (any test that fits the assigned risks).
- Most likely the team will discover (potential) problems. Problems are fixed and tested again. During testing you can also find new risks that need more testing.
- When the risk gap is closed (and when the PO/stakeholders decide) the product can be shipped by the team.
- After shipping problems (and risks) could be found and the cycle starts again.

Chapter 4 – User Stories

User stories are the most commonly used way of reporting/documenting specifications. Although user stories are less complex than a fully functional and technical design, they could create issues within the Agile team. Examples are: user stories can be unclear, incomplete, too big, untestable or way too dependent on other stories.

This is a threat to the quality of the end result and thus it makes it very relevant for a tester to know the details about good user stories.

Keywords

Value, feedback, format (WHO, WHAT, WHY), specific, acceptation criteria (GIVEN WHEN THEN format, boundary, consensus, test base, base for planning), horror plots, story splitting, (Workflow, CRUD (Create Read Update Delete), Roles), INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable).

LO-4.1	K2	Understand the values of user stories
LO-4.2	K1	Recall the elements of a user story
LO-4.3	K2	Demonstrate the format with an example
LO-4.4	K2	Understanding the pitfalls while using the format
LO-4.5	K1	Define an acceptance criterion
LO-4.6	K2	Understand the testers' role in defining and elaborating acceptance criteria
LO-4.7	K2	Understand the pitfalls of defining acceptance criteria
LO-4.8	K2	Understand horror plots as a medium to find acceptance criteria
LO-4.9	K2	Understand story splitting
LO-4.10	K2	Understand how to split stories
LO-4.11	K1	Recall when to split a story
LO-4.12	K2	Understand when a story is well split
HO-4.1	HO-3	Analyse and redesign the given story

4.1 Why do we use user stories?

LO-4.1	K2	Understand the values of user stories

User stories are used for at least two reasons: they create value for the customer and they are a medium to deliver feedback on the activities of a team **[US1]**.

4.2 The elements of a user story

LO-4.2	K1	Recall the elements of a user story

Explain that user stories consist of the actual conversation about the customers needs and some form of writing is down with a short unique title, a WHO, WHAT and WHY **[AA1]** of the story, and one or more acceptance criteria.

According to the 3C concept **[JF1]**, a user story is the conjunction of three elements:

- Card: The card is the physical media describing a user story. It identifies the requirement, its criticality, expected development and test duration, and the acceptance criteria for that story. The description has to be accurate, as it will be used in the product backlog.
- Conversation: The conversation explains how the software will be used. The conversation can be documented or verbal. Testers, having a different point of view than developers and business representatives [ISTQB_FL_SYL], bring valuable input to the exchange of thoughts, opinions, and experiences. Conversation begins during the release-planning phase and continues when the story is scheduled.
- Confirmation: The acceptance criteria, discussed in the conversation, are used to confirm that the story is done. These acceptance criteria may span multiple user stories. Both positive and negative tests should be used to cover the criteria. During confirmation, various participants play the role of a tester. These can include developers as well as specialists focused on performance, security, interoperability, and other quality characteristics. To confirm a story as done, the defined acceptance criteria should be tested and shown to be satisfied.

4.3 Example and challenges

LO-4.3	K2	Demonstrate the format with an example
LO-4.4	K2	Understanding the pitfalls while using the format

An example is provided. Once the students have understood this example, they are confronted with the fact that the example is flawed and why: a user story needs to be as specific as possible to be of some value. This specification is demonstrated.

4.4 What is an acceptance criterion?

LO-4.5	K1	Define an acceptance criterion
LO-4.6	K2	Understand the testers role in defining and elaborating acceptance criteria

Acceptance criterion is explained in terms of the format (we use: given, when, then (however there are no rules, the team decides)) and goal (creating boundaries, consensus on what (and what not) to do, and is a base for testing and estimating effort and planning). To be testable, acceptance criteria should address the following topics where relevant **[WG1]**:

- Functional behavior: The externally observable behavior with user actions as input operating under certain configurations.
- Quality characteristics: How the system performs the specified behavior. The characteristics may also be referred to as quality attributes or non-functional requirements. Common quality characteristics are performance, reliability, usability, etc.
- Scenarios (use cases): A sequence of actions between an external actor (often a user) and the system, in order to accomplish a specific goal or business task.
- Business rules: Activities that can only be performed in the system under certain conditions defined by outside procedures and constraints (e.g., the procedures used by an insurance company to handle insurance claims).

- External interfaces: Descriptions of the connections between the system to be developed and the outside world. External interfaces can be divided into different types (user interface, interface to other systems, etc.).
- Constraints: Any design and implementation constraint that will restrict the options for the developer. Devices with embedded software must often respect physical constraints such as size, weight, and interface connections.
- Data definitions: The customer may describe the format, data type, allowed values, and default values for a data item in the composition of a complex business data structure (e.g., the ZIP code in a U.S. mail address).

The role of a tester is to use all his/her knowledge and experience to get awareness in the team to have acceptance criteria with clear business value.

Acceptance criteria are difficult to describe in a non-ambiguous way and the help of a tester is needed to do that. Thinking in acceptance criteria improve user stories.

4.5 Horror plots

LO-4.7	K2	Understand the pitfalls of defining acceptance criteria
LO-4.8	K2	Understand horror plots as a medium to find acceptance criteria

Horror plots are introduced as a medium to overcome the pitfall of the usual way to find acceptance criteria (being a non-conclusive list).

Horror plots are for approaching userstories from a specific test point of view, where the main question is: "what can go wrong?".

The definition of a horror plot: ´the outcome of a user story that you definitely do not want your customers to find out or see´. An example: i can see the bankaccount data from somebody else, as a horror plot of login to my personal bank environment.

Horror plots are shown based on the user story, as introduced earlier on.

4.6 Story Splitting

LO-4.9	K2	Understand story splitting
LO-4.10	K2	Understand how to split stories
LO-4.11	K1	Recall when to split a story
LO-4.12	K2	Understand when a story is well split.
HO-4.1	HO-3	Analyse and redesign the given story

This chapter starts with a story which is too big to build / understand / plan. This is followed by an explanation of three ways how we can split a story (workflow, data-actions, user roles). Then it is explained when to split stories and we end with the criteria used to check whether a story is of the

right quality (the INVEST acronym (Independent, Negotiable, Valuable, Estimable, Small and Testable) is explained in detail).

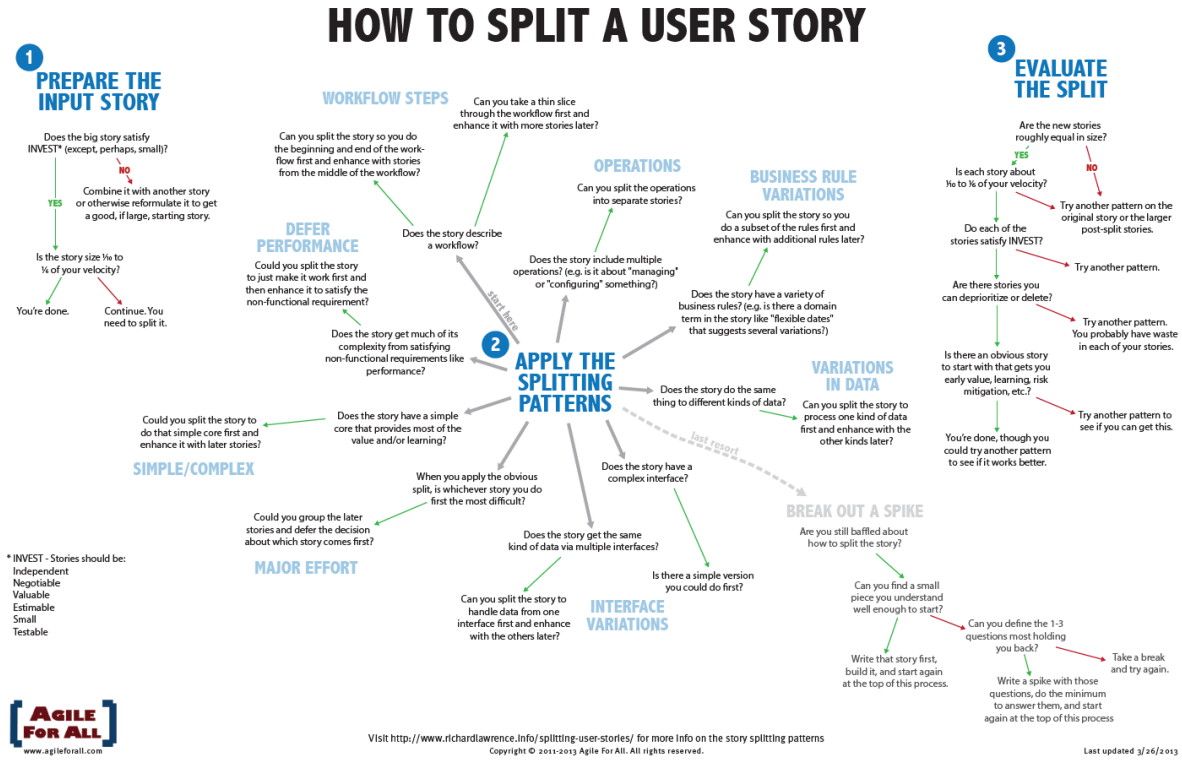

Figure 6 ***[AF1]***

Finally, students are confronted with a user story that requires them to apply the knowledge obtained in this module. They need to split the story, define new stories with the correct format; passively invited to find new acceptance criteria (using Horror plots), and also to check their own defined smaller user stories with the INVEST acronym.

Chapter 5 – Test Strategy

Based on the risks, the tester (and team) need to decide to choose an appropriate effective way of testing – a way that fits the risks, the project, the company and its goals, the team and the product. The strategy needs to make clear what we are going to do, why we are doing it, how we are doing it, who is doing it and what the results of our actions have been. This gives the owner of the product (Stakeholders/PO) the best possible insights in the software and, based on these insights, they can decide what to do next. The first parts are covered in Chapter 2 and 3. This chapter contains the final parts of the test strategy objects.

Keywords

Agile, Testing, Test strategy, Exploratory testing, Checking, Test charter, Test automation, Test area´s

LO-5.1	K2	Understand that a test strategy is about more than just the think it and build it phase in development. Testing is a constant activity, continuous quality.
LO-5.2	K2	Understand that valuable information about the product will come from the production situation and can help you create a better understanding of risks and can help you adjust your test strategy.
LO-5.3	K2	Understand the difference between different types of testing; checking versus intuition and exploration
LO-5.4	K2	Be able to demonstrate the levels of testing mapped on Rumsfeld´s "There are known knowns"
LO-5.5	K2	Understand the definition of BDD (Behaviour Driven Development)
LO-5.6	K2	Understand the definition of SBE (Specification By Example)
LO-5.7	K2	Understand the definition of (A)TDD ((Acceptance) Test Driven Development)
LO-5.8	K2	Understand the relations between BDD/(A)TDD/SBE
LO-5.9	K2	Understand the specification syntax Gherkin
LO-5.10	K2	Understand the relationship between BDD/(A)TDD/SBE, Gherkin and testautomation
LO-5.11	K2	Understand 3-amigo sessions and example mapping
HO-5.1	HO-2	Be able to use SBE and Example mapping
LO-5.12	K1	Be able to recall the definition of exploratory testing according to Cem Kaner
LO-5.13	K3	Use the definition of exploratory testing to execute an exploratory test
LO-5.14	K2	Understand the different sources and possibilities to setup exploratory testing
LO-5.15	K1	Be able to select appropriate sources to execute an exploratory test
LO-5.16	K2	Understand what a test charter is
LO-5.17	K2	Understand how to setup a exploratory test session
LO-5.18	K2	Understand how to document your exploratory test sessions
LO-5.19	K2	Understand the definition of test automation
LO-5.20	K1	Recall checking versus intuition and exploration
LO-5.21	K2	Understand the value of test automation
LO-5.22	K2	Understand the limitations of test automation
LO-5.23	K2	Classify different types of test in test automation
LO-5.24	K3	Learn to apply the test automation pyramid
LO-5.25	K2	Demonstrate the difference between the test automation pyramid and the test automation ice cone anti-pattern
LO-5.26	K2	Understand the importance to be familiar to a broad range of tools that can make your testing more efficient
LO-5.27	K2	Understand the value and need of automation in an Agile context

5.1 General overview

LO-5.1	K2	Understand that a test strategy is about more than just the think it and build it phase in development. Testing is a constant activity, continuous quality.
LO-5.2	K2	Understand that valuable information about the product will come from the production situation and can help you create a better understanding of risks and can help you adjust your test strategy.
LO-5.3	K2	Understand the difference between different types of testing; checking versus intuition and exploration
LO-5.4	K2	Be able to demonstrate the levels of testing mapped on Rumsfeld´s "There are known knowns"

First the domain of different types of tests is explored. There are many types of tests and each have their own objective and value. In the past, testers mainly focussed on the types of tests that are related to checks; tests that tell us something about facts we know from the system (by documentation). We check whether the system does what it is supposed to do – does it work?

Since the introduction of Agile, our customers become central in everything we do. Teams focus on whether the system/product does what a customer expects. This is a fundamentally different approach that is only partly covered when we check if the system functions correctly. Also, customers nowadays expect a system to be good quality. Failures could result in customers moving to a competitor who can deliver the same product that does not fail.

Testing in Agile starts in the ´think it´/ ´design´ phase by turning you into a critical thinker who questions and challenges requirements and comes up with horror plots. This is usually called “shift left”.

Also, during the shipping and especially during production, we perform “tests” by gathering data from monitoring and logging to gain information about the use of the product. We can also use tests such as A/B tests to get information about our new features. Usually this is called “shift right”.

Teams constantly learn and adapt, based on the results of all the tests. We adjust the risks and our strategy all the time.

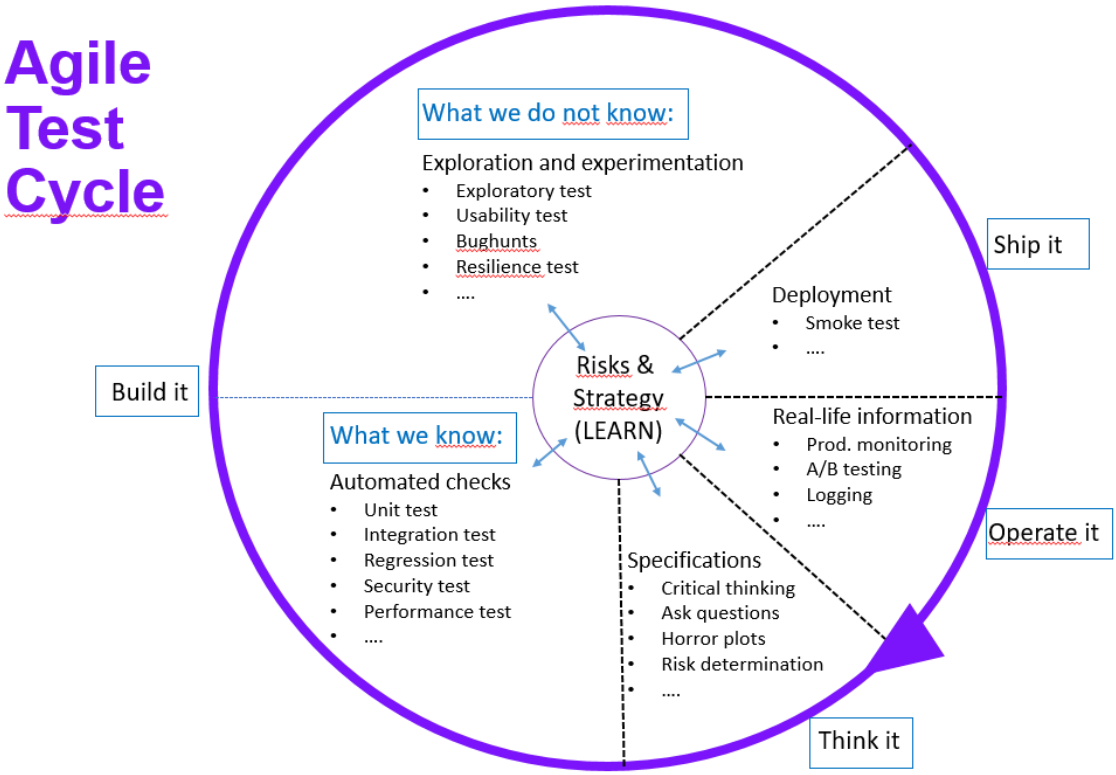

Figure 7

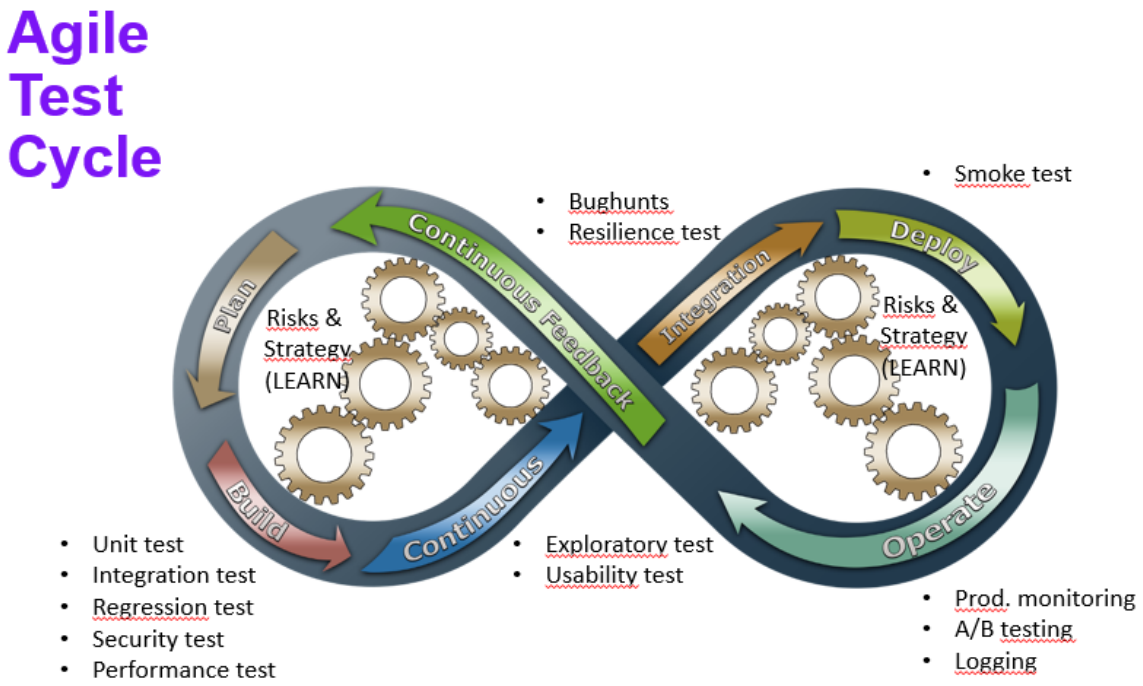

Figure 8

Testing has become a constant activity that starts with the Think it / Design and runs through to production. The purpose of the Agile Test Cycle is that you plot your own different types of test. So which tests are you going to do and when exactly? This is context dependent and the examples

above are an interpretation. The purpose of the model is that you yourself start thinking about which tests you should do and when the most convenient time is to do them.

5.2 Behaviour Driven Development

LO-5.5	K2	Understand the definition of BDD (Behaviour Driven Development)
LO-5.6	K2	Understand the definition of SBE (Specification By Example)
LO-5.7	K2	Understand the definition of (A)TDD ((Acceptance) Test Driven Development)
LO-5.8	K2	Understand the relations between BDD/(A)TDD/SBE
LO-5.9	K2	Understand the specification syntax Gherkin
LO-5.10	K2	Understand the relationship between BDD/(A)TDD/SBE, Gherkin and testautomation
LO-5.11	K2	Understand 3-amigo sessions and example mapping
HO-5.1	HO-2	Be able to use SBE and Example mapping

In software engineering, behavior-driven development (BDD) is an Agile software development process that encourages collaboration among developers, QA and non-technical or business participants in a software project **[DN1]**. It encourages teams to use conversation and concrete examples to formalize a shared understanding of how the application should behave. It emerged from test-driven development (TDD) **[KB1]**. Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design **[EE1]** and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development.

TDD is a software-development methodology which essentially states that for each unit of software, a software developer must:

- define a test set for the unit first.
- make the tests fail.
- then implement the unit.
- finally verify that the implementation of the unit makes the tests succeed.

This definition is rather non-specific in that it allows tests in terms of high-level software requirements, low-level technical details or anything in between. One way of looking at BDD therefore, is that it is a continued development of TDD which makes more specific choices than TDD.

BDD specifies that tests of any unit of software should be specified in terms of the desired behavior of the unit. Borrowing from agile software development the "desired behavior" in this case consists of the requirements set by the business, that is, the desired behavior that has business value for whatever entity commissioned the software unit under construction.
Following this fundamental choice, a second choice made by BDD relates to how the desired behavior should be specified. In this area BDD chooses to use a semi-formal format for behavioral specification which is borrowed from user story specifications from the field of object-oriented analysis and design.
BDD specifies that business analysts and developers should collaborate in this area and should specify behavior in terms of user stories. Each user story should, in some way, follow the following structure:

- Title
  An explicit title.
- Narrative
  A short introductory section with the following structure:

As a: the person or role who will benefit from the feature;
I want: the feature;
so that: the benefit or value of the feature.

- Acceptance criteria
  A description of each specific scenario of the narrative with the following structure:
  Given: the initial context at the beginning of the scenario, in one or more clauses;
  When: the event that triggers the scenario;
  Then: the expected outcome, in one or more clauses.

BDD does not have any formal requirements for exactly how these user stories must be written down, but it does insist that each team using BDD come up with a simple, standardized format for writing down the user stories which includes the elements listed above.

Specification by example (SBE) **[GA1]** is a collaborative approach to defining requirements and business-oriented functional tests for software products based on capturing and illustrating requirements using realistic examples instead of abstract statements. It is applied in the context of agile software development methods, in particular BDD. This approach is particularly successful for managing requirements and functional tests on large-scale projects of significant domain and organisational complexity.

A key aspect of SBE is creating a single source of truth about required changes from all perspectives. When business analysts work on their own documents, software developers maintain their own documentation and testers maintain a separate set of functional tests, software delivery effectiveness is significantly reduced by the need to constantly coordinate and synchronise those different versions of truth. With SBE, different roles participate in creating a single source of truth that captures everyone's understanding. Examples are used to provide clarity and precision, so that the same information can be used both as a specification and a business-oriented functional test. Any additional information discovered during development or delivery, such as clarification of functional gaps, missing or incomplete requirements or additional tests, is added to this single source of truth. As there is only one source of truth about the functionality, there is no need for coordination, translation and interpretation of knowledge inside the delivery cycle.

When applied to required changes, a refined set of examples is effectively a specification and a business-oriented test for acceptance of software functionality. After the change is implemented, specification with examples becomes a document explaining existing functionality. As the validation of such documents is automated, when they are validated frequently, such documents are a reliable source of information on business functionality of underlying software.

The Three Amigos **[AA2]**, also referred to as a "Specification Workshop", is a meeting where the Product Owner discusses the requirement in the form of SBE by Example with different stakeholders like the team members. The key goal for this discussion is to trigger conversation and identify any missing specifications. The discussion also gives a platform for the team and Product owner to converge and hear out each other's perspective to enrich the requirement and also make sure if they are building the right product.

The three Amigos are

- Business - Role of the Business user is to define the problem only (and not venture into suggesting any solution)
- Development - Role of the Developers involve to suggest ways to fix the problem
- Testing - Role of testers is to question the solution, bring up as many as different possibilities for brain storming through What-If scenarios and help make the solution more precise to fix the problem.

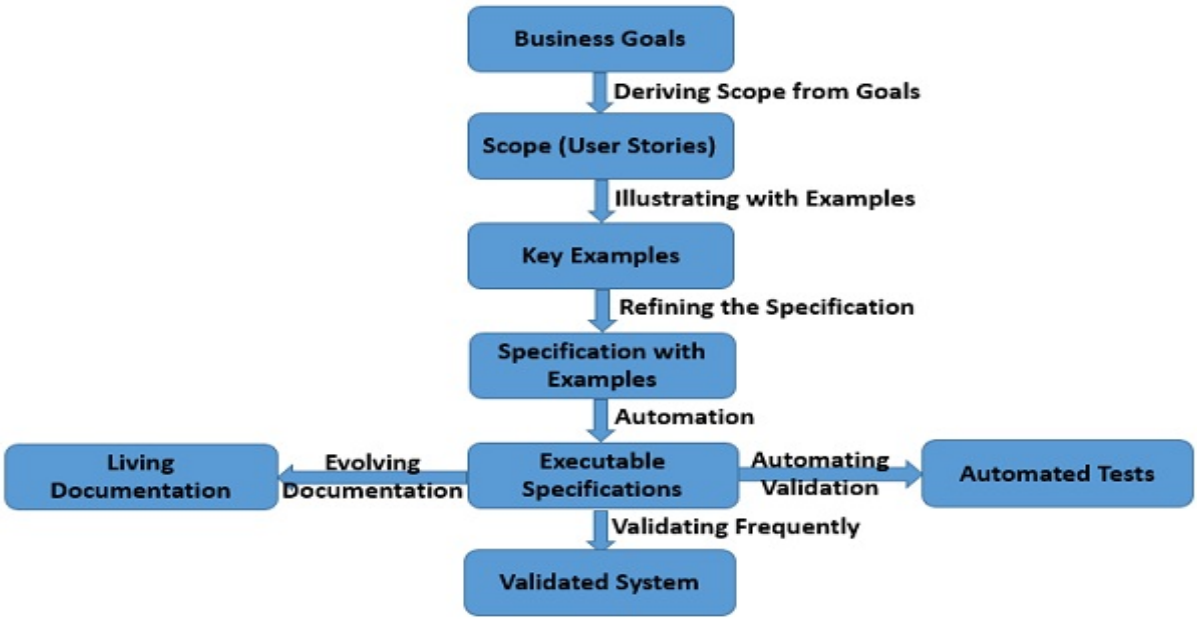

Figure 9

Refining the specification can be done with Example Mapping.**[MW1]** Example Mapping is a technique that can steer the conversation and derive Acceptance criteria within 30 minutes .The process involves breaking each stories into Rules and Examples and documented in the form of Specification by examples.

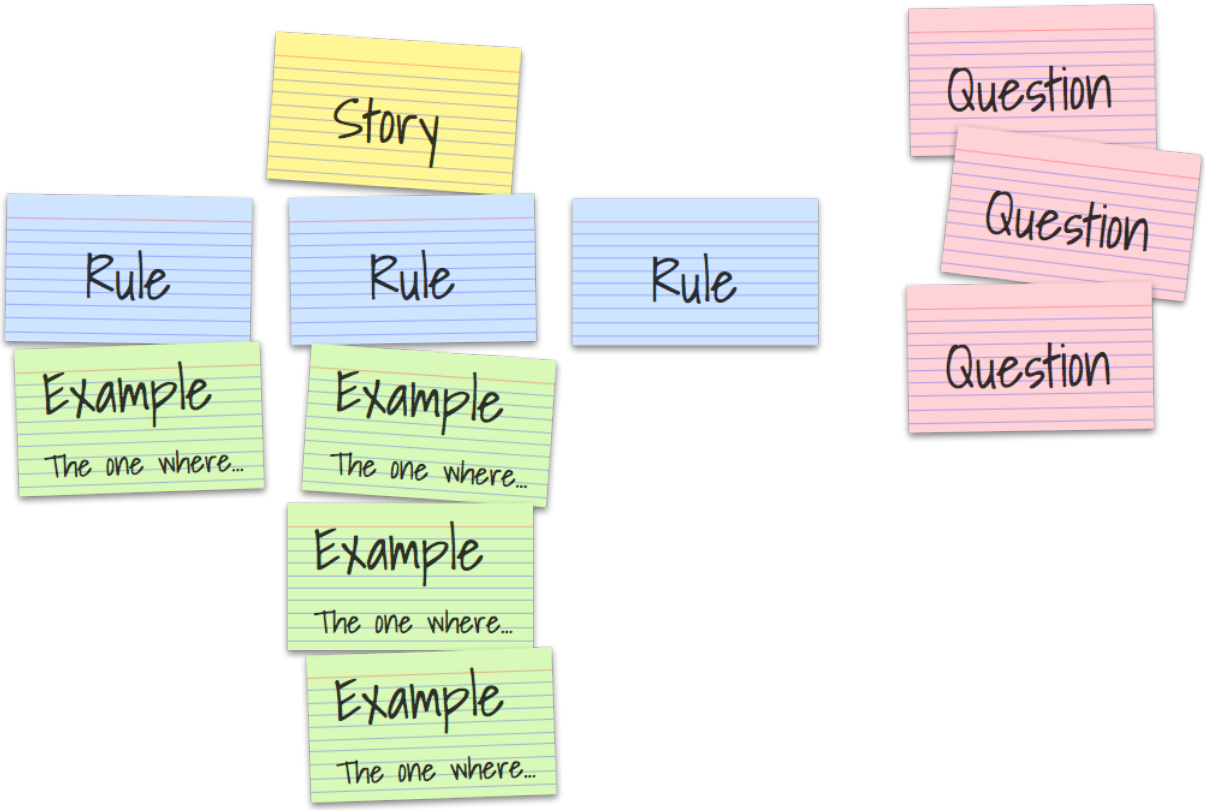

Figure 10

In order to make automation possible (which by itself is not the goal of SBE) you need to write down the specifications in an executable way. Gherkin **[GH1]** is the language that Cucumber uses to define

test cases. It is designed to be non-technical and human readable, and collectively describes use cases relating to a software system. The purpose behind Gherkin's syntax is to promote BDD practices across an entire development team, including business analysts and managers. It seeks to enforce firm, unambiguous requirements starting in the initial phases of requirements definition by business management and in other stages of the development lifecycle.

In addition to providing a script for automated testing, Gherkin's natural language syntax is designed to provide simple documentation of the code under test.

Cucumber tests are divided into individual Features. These Features are subdivided into Scenarios, which are sequences of Steps.

A Feature is a Use Case that describes a specific function of the software being tested. There are three parts to a Feature

- The Feature: keyword
- The Feature name (on the same line as the keyword)
- An optional description on the following lines

Example Feature definition:

```
Feature: Withdraw Money from ATM
A user with an account at a bank would like to withdraw money from an ATM.

Provided he has a valid account and debit or credit card, he should be
allowed to make the transaction. The ATM will tend the requested amount of
money, return his card, and subtract amount of the withdrawal from the
user's account.

    Scenario: Scenario 1
        Given preconditions
        When actions
        Then results

    Scenario: Scenario 2
        ...
```

Each Feature is made of a collection of scenarios. A single scenario is a flow of events through the Feature being described and maps 1:1 with an executable test case for the system. Keeping with the example ATM withdrawal feature, a scenario might describe how a user requests money and what happens to their account.

```
Scenario: Eric wants to withdraw money from his bank account at an ATM
    Given Eric has a valid Credit or Debit card
    And his account balance is $100
    When he inserts his card
    And withdraws $45
    Then the ATM should return $45
    And his account balance is $55
```

The crux of a Scenario is defined by a sequence of Steps outlining the preconditions and flow of events that will take place. The first word of a step is a keyword, typically one of

- Given - Describes the preconditions and initial state before the start of a test and allows for any pre-test setup that may occur
- When - Describes actions taken by a user during a test
- Then - Describes the outcome resulting from actions taken in the When clause

Occasionally, the combination of Given-When-Then uses other keywords to define conjunctions

- And - Logical and
- But - Logically the same as And, but used in the negative form

```
Scenario: A user attempts to withdraw more money than they have in their
account
    Given John has a valid Credit or Debit card
    And his account balance is $20
    When he inserts his card
    And withdraws $40
    Then the ATM displays an error
    And returns his card
    But his balance remains $20
```

5.3 Exploratory Testing

LO-5.12	K1	Be able to recall the definition of exploratory testing according to Cem Kaner
LO-5.13	K3	Use the definition of exploratory testing to execute an exploratory test
LO-5.14	K2	Understand the different sources and possibilities to setup exploratory testing
LO-5.15	K1	Be able to select appropriate sources to execute an exploratory test
LO-5.16	K2	Understand what a test charter is
LO-5.17	K2	Understand how to setup a exploratory test session
LO-5.18	K2	Understand how to document your exploratory test sessions

With exploratory testing, a first way of customer-faced testing is introduced. The product is explored based on a customer facing risks and, while doing that, areas of the application are touched that are unknown as we could not document it because we did not know about it.

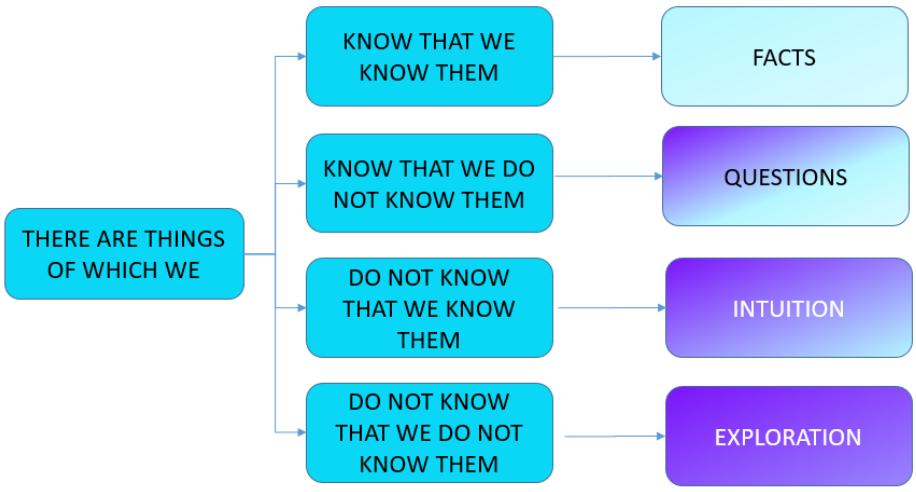

Figure 11

Exploratory testing is explained according to the definition of Cem Kaner:

"Exploratory software testing is a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project." ***[CK1]***

We explain advantages of Exploratory Testing in terms of:

- Getting insights in the unknowns
- Learn about the product very fast
- Learn about non-specified features and use of the system
- Test much faster and more effective
- Bring back unknowns to the knowns domain

After the definition the first approach towards exploratory testing is introduced. Which elements play a role in exploratory testing and how a tester can start doing valuable exploratory tests.

Typical elements that can play a role in exploratory testing are:

- Project Outline
- Product Outline
- Risks
- Quality attributes
- Specifications
- Technical impact
- Feelings/Intuition/Experience
- Mnemonics
- Heuristics
- Checklists
- Historical patterns of failure

The test charter is introduced as a way to structure and guide exploratory tests and the session sheet is added to it to write down the results of our tests. Test charter and session sheet consist of:

- Testcharter
    - Mission of our test (what we are trying to achieve).
    - Purpose of our test (how we are going to achieve that, what we are looking for, what we expect to find, where we expect to find it).
- Session sheet
    - Time box
    - Notes
        - What we are testing
        - Test results
        - Questions that are raised
        - Issues found
        - Relevant test data we have used/needed
    - Debriefing information BRIEF
        - **B**ehaviour: how did the software behave during our session

- **R**esults: what did we archive
- **I**mpediments: what was in our way
- **E**xpectations: what still has to be done
- **F**eelings: how do i feel about the product

5.4 Test automation & tools

LO-5.19	K2	Understand the definition of test automation
LO-5.20	K1	Recall checking versus intuition and exploration
LO-5.21	K2	Understand the value of test automation
LO-5.22	K2	Understand the limitations of test automation
LO-5.23	K2	Classify different types of test in test automation
LO-5.24	K3	Learn to apply the test automation pyramid
LO-5.25	K2	Demonstrate the difference between the test automation pyramid and the test automation ice cone anti-pattern
LO-5.26	K2	Understand the importance to be familiar to a broad range of tools that can make your testing more efficient
LO-5.27	K2	Understand the value and need of automation in an Agile context

The definition of test automation: “a way to automatically repeat the execution of test cases including the checks on the pre-defined results (automated validations)”.

This also gives the limitations of test automation:

- We need pre-defined results that do not change
- We need to create test scripts upfront, so it has to be clear what the system will do
- We only cover the facts (checks), not the intuition and exploration
- Is there enough value in repeating the same tests over and over again?
- Return on Investment on test automation should be taken into account

The different kinds of levels are shown on which test automation can play a role by using the test automation pyramid introduced by Mike Cohn **[MC1]** that introduces 3 layers of test automation:

- UI level
- API or service level
- Unit level

The amount of value automated tests deliver are:

- An automated unit test delivers most value as the feedback loop is very short and many checks can be easily covered.
- API or service level tests (contract testing is a common term nowadays) are valuable as well as an API has very clear pre-defined content and that makes it suitable for automated checking.
- The last level that should contain the least tests is the UI level. This level is changing a lot and automated tests are flaky and hard to maintain and tools are expensive to buy or setup and maintain.

Unfortunately, as the UI level is the most visible level, a lot of organisations choose to mainly create UI-level automated tests. This is what is called the “ice cone” anti-pattern of test automation. Having

this information gives the tester the opportunity to explain and show the benefit of a good test automation strategy.

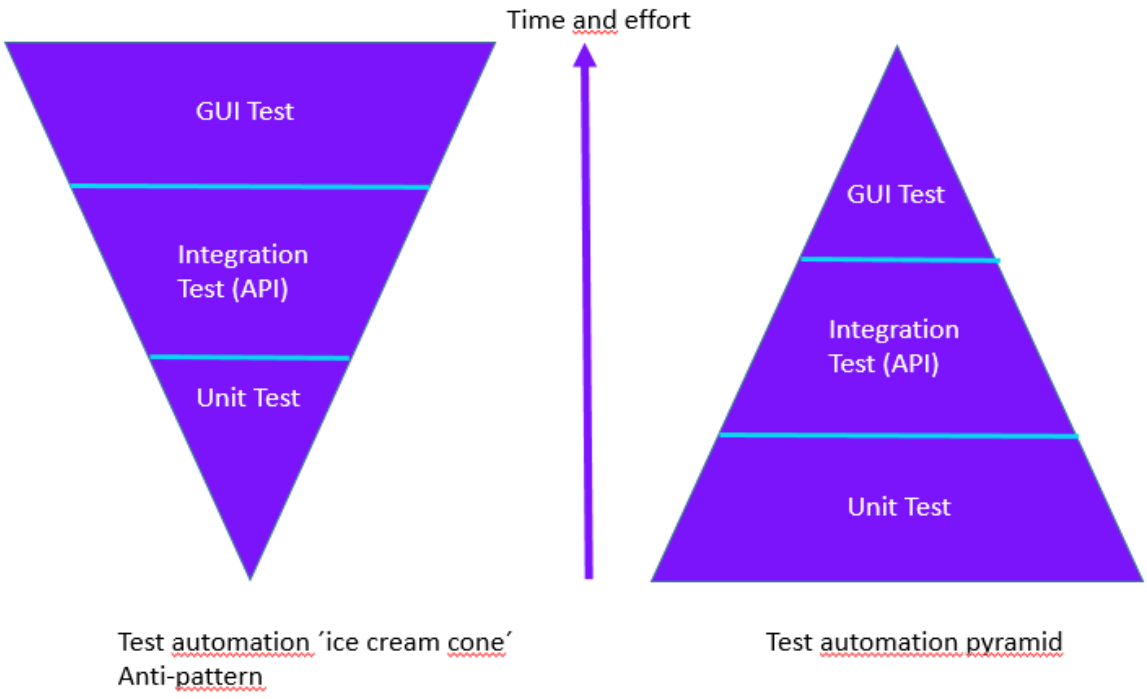

Figure 12

People all know testing tools, however we mainly think of them as test automation tools such as Selenium, SoapUI, Citrus, Cypress....

There are way more tools that can be very helpful at times, to help testers to:

- Documenting our tests, like screen recorders or even Word or a wiki or Confluence
- Finding potential bugs more easily, like Firebug or Bug Magnet
- Running some first basic security checks, like ZAP
- Checking for first potential performance issues, like Yslow
- Checking on potential usability issues, like A-checker WCAAG

Chapter 6 – Test reporting

Reporting needs to be in place on what is done in relation to quality and testing. What was done and with what results? The reporting needs to be lightweight, as we value working software over comprehensive documentation. Reporting is usually also needed for governance, risk and compliance (GRC) purposes. What reporting is needed is highly depending on the context of the organization, the specific project and the product. One rule always applies: reporting needs to fullfill a need. The amount and way of reporting is depending on this need.

3 different ways of reporting are introduced: the one-page status overview, story telling and issue management.

Keywords

Agile, Testing, Reporting

LO-6.1	K2	Understand the value of reporting about testing and quality in Agile
LO-6.2	K3	Implement A3 reporting on test and quality aspects
HO-6.1	HO-2	Create actual valuable reports on the tested subject
LO-6.3	K1	Remember why people need story telling. As a child we learn from stories, as adults we have forgotten this highly effective way of learning and reporting
LO-6.4	K2	Understand what story telling actually is and be able to explain it to others
LO-6.5	K2	Demonstrate the minimal ingredients of a good story
LO-6.6	K3	Apply story telling to testing and be able to tell a testing story
HO-6.2	HO-1	Learn to tell a story guided by the trainers' steps
LO-6.7	K1	Be able to recall the definition of an issue/bug
LO-6.8	K2	Understand when to report an issue and when not
LO-6.9	K2	Understand how to report an issue in order for it to be of value
HO-6.3	HO-2	Create an issue on tested software according to the principles learned

6.1 The one page test status overview

LO-6.1	K2	Understand the value of reporting about testing and quality in Agile
LO-6.2	K3	Implement A3 reporting on test and quality aspects
HO-6.1	HO-2	Create actual valuable reports on the tested subject

Being Agile means teams focus less on documentation. This does not mean, however, that there is no need for any documentation. We will still need to be able to report on our progress regularly and instantly. The elements are identified that are important and give some visual reporting hints on how to use them in order to deliver the needed value for our customers.

An exercise to actually create an A3 test report about the tests just performed finalizes the chapter.

Typical information to be reported about:

- What was tested and what was not tested (yet) and why not.
- What were the results of the tests, which information has the test uncovered.
- Who performed the tests and when were they preformed it.

- What needs more investigation.
- In what way regulations are incorporated (if applicable)

6.2 The Test Story / Story telling

LO-6.3	K1	Remember why people need story telling. As a child we learn from stories, as adults we have forgotten this highly effective way of learning and reporting
LO-6.4	K2	Understand what story telling actually is and be able to explain it to others
LO-6.5	K2	Demonstrate the minimal ingredients of a good story
LO-6.6	K3	Apply story telling to testing and be able to tell a testing story
HO-6.2	HO-1	Learn to tell a story guided by the trainers' steps

In the pre-Agile era, testers created "visible value" by creating test reports about the created test plans, the number of tests created and executed, the number of bugs found. A lot of documentation was created and that convinced most management that a tester added real value.

In the Agile era, we create less documentation. We focus much more on giving valuable insights on the quality of the application, on preventing bugs from happening at all and on team collaboration when bugs are found to actually solve them instantly instead of just reporting them.

Some neuroscience is added to story telling **[ST1]** so you understand why some stories are more interesting listening to than others:

- Dopamine: makes people more motivated and activates the memory.
- Oxytocin: makes people more generous and triggers the need to connect.
- Endorphins: makes people more creative and relaxed.

How do you ensure that your listener creates them, so your story is interesting to listen to and your message will be remembered?

- Dopamine: tell an exciting story with a cliffhanger.
- Oxycotine: tell a story that evokes empathy (or pity).
- Endorphins: tell a story that makes people laugh.

This makes it much harder to show the actual value of testing to stakeholders outside the team. Development shows a build product in a demo. How can the added value of testing and quality be shown?

This can be done by telling the story about testing. What testing have been done, why and with which results. Key points in the test story are the customer/client and what value testing brings him/her. This way, testers can also tell about the "non-visible" value that is created, like helping create good user stories, help developers make unit tests, the bugs found and solved directly together with the developer/PO.

6.3 Issues and issue management

LO-6.7	K1	Be able to recall the definition of an issue/bug
LO-6.8	K2	Understand when to report an issue and when not
LO-6.9	K2	Understand how to report an issue in order for it to be of value

HO-6.3	HO-2	Create an issue on tested software according to the principles learned

The main goal of testing is not to find as many bugs as possible. The main goal of testing is to give insights on the quality of the product and to help make the product better. By documenting bugs, the product will not get any better. A tester needs to test as much as possible to get valuable insights. As soon as a bug is found, a tester should talk to the developer and/or product owner about the risk posed by the bug found and, as a team, decide together what to do with the bug. If it has to be solved, then team solves it immediately and does not document it other than a mentioning in a test log. If it cannot be solved immediately, the team can decide to document the bug for later investigation and ask the PO to add an bug fixing item to the product backlog.

In that case provide the next information:

- steps to reproduce
- expected outcome and actual outcome
- test data used
- test environment used
- screen prints, video, log files
- priority (progress of the iteration)
- severity (how much damage would it be if we went live)
- what you as a team find useful (and strictly necessary)

By using this way of handling issues a way of working is introduced that fits Agile. Preventing issues over finding issues and solving issues over documenting issues. A tester in an Agile context could use the retrospective as means to work towards this issue mindset in the team.

References

General references

Some of the test strategy elements are inspired by: The heuristic test strategy model by James Bach https://www.satisfice.com/download/heuristic-test-strategy-model

Some of the content is inspired by: Rapid Software Testing by James Back and Michael Bolton https://rapid-software-testing.com/

[SH1] Software will be made available to training partners and trainers. Details will be in the trainer manual. Hardware can be ordered by searching for ´LED 3D Hologram Fan´. The fans are widely available, however are optional to use.

Specific references

Reference	Source
[BT1]	A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives - By L. Anderson, P. W. Airasian, and D. R. Krathwohl (Allyn & Bacon 2001)
[BT2]	https://www.apu.edu/live_data/files/333/blooms_taxonomy_action_verbs.pdf
[JL1]	Agile Testing, A practical guide for testers and agile teams https://agiletester.ca/ More Agile Testing, Learning Journeys for the Whole Team https://agiletester.ca/
[AG1]	https://www.agilealliance.org/agile101/
[AM1]	http://agilemanifesto.org/
[SC1]	https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf
[RF1]	https://en.wikipedia.org/wiki/There_are_known_knowns **DoD News Briefing - Secretary Rumsfeld and Gen. Myers** **Presenter: Secretary of Defense Donald H. Rumsfeld** **February 12, 2002 11:30 AM ED** https://archive.defense.gov/Transcripts/Transcript.aspx?TranscriptID=2636
[GA1]	https://www.growingagile.co.za/
[BO1]	http://www.developsense.com/resources/Oracles.pdf
[JW1]	Jerry Weinberg http://geraldmweinberg.com/
[TE1]	https://www.testingexcellence.com/test-oracles-test-heuristics/
[TS1]	https://www.ministryoftesting.com/testsphere
[EH1]	http://testobsessed.com/2006/12/the-nightmare-headline-game-planning-for-the-unexpected/
[CK1]	http://kaner.com/?p=46
[DN1]	Dan North https://dannorth.net/introducing-bdd/
[KB1]	Beck, Kent (2002-11-08). *Test-Driven Development by Example*. Vaseem: Addison Wesley. ISBN 978-0-321-14653-3.
[EE1]	Evans, Eric (2004). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley. ISBN 978-032-112521-7.
[GA1]	Adzic, Gojko (2009). *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*. Neuri. ISBN 0-9556836-1-0.
[MW1]	Matt Wyne https://www.youtube.com/watch?v=VwvrGfWmG_U
[AA2]	2009: George Dinwiddie https://www.agilealliance.org/glossary/three-amigos/
[GH1]	https://cucumber.io/docs/guides/overview/#what-is-gherkin

[PD1]	The new consultation: developing doctor–patient communication. David Pendleton, Theo Schofield, Peter Tate, Peter Havelock. Oxford University Press, 2003. ISBN 0-19-263288
[HB1]	https://projectaccess.uoregon.edu/teachers/socialemotional/socialskills/SW%20E%20Hamburger%20Method%20of%20Construct%20Criticism%20DONE.pdf
[MC1]	https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid
[ST1]	https://en.wikipedia.org/wiki/Storytelling
[US1]	https://www.scrum.org/resources/blog/10-tips-product-owners-business-value
[AA1]	https://www.agilealliance.org/glossary/user-story-template/
[KM1]	Thinking, Fast and Slow, D. Kahneman, ISBN: 9780141033570
[JF1]	Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
[WG1]	Karl Wiegers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.
[AF1]	https://agileforall.com/resources/how-to-split-a-user-story/