Agile testing and testing in agile software development

This presentation describes agile testing and how it is applied in agile and traditional software development and also about other testing in agile software development.

UPDATE 2010

Matti Vuori, www.mattivuori.net

Contents 1/6

Contents 2/6

Contents 3/6

Contents 4/6

Contents 5/6

Contents 6/6

About this slide set

- This set is about 1) agile testing and 2) testing in agile development.
- It is assumed that the readers know the basics of testing and agile development and know the basics of Scrum.
- This is not the only slide set. More information is available about for example:
  - Scrum.
  - Effective error seeking.
  - Testing unit’s instructions for working in a Scrum project.
  - And of course, about all areas of testing, most of which still apply in agile testing. (For example, the old testing techniques used in functional testing are very relevant and will remain so.)

Agility in software development

- Instead of complete pre-made plans living in the moment.
- Plans are rapidly changed when situations change.
- The thought that software development is dynamic and a continuous learning process.
- It is impossible to know beforehand how the product will develop and what is needed to work with it.
- Understanding of the product changes continuously.
- Change is seen as a positive thing.

Agile testing

- Agile testing can mean many kinds of testing:
  - Any testing that is not based on test case level plans.
  - Exploratory (sometimes called explorative) testing, where the tester proceeds based on his/her observations of the software.
  - Sometimes it means testing is agile software development.
  - The test-based development, used by programmers, is also often classified as agile testing due to its rapid and cyclic nature.
- It is a counter-force to the old ideal of “plan based systematic testing”, in which one at an early stage, based on specifications, decides what is tested and how.

Exploratory testing: How?

- One explores the software by using it and makes observations, tries to learn and understand it.
- Areas are identified that are risk-prone and need testing
- Tests are carried out right away, without formal test cases.
- Or more detailed tests are designed and executed.
- Results are evaluated and iteration is continued.

Exploratory testing: Why? 1/2

- It is good to base planning to what the implementation tells about the software.
  - One is completely realistic. Documents are not relied upon, but what we see that has been produced. What features really are there?
  - Specifications are always incomplete and have errors. Let’s not get stuck with that problem.
  - (Or course we must know how the software should work.)
- Time is not spent on planning the testing, results are gotten faster
  - Faster starting of testing.
  - Fast reaction to changes.
  - Fast feedback to developers.
- It fits to situation where requirements change constantly.
  - Otherwise the plans would be always out of date

Exploratory testing: Why? 2/2

- When we have eyes open, errors are found more effectively that if we just use pre-planned test cases.
  - Too much planning creates too much expectations and closes eyes from observations.
- One can identify characteristics in the software that were not brought up during specification.
- Software is so complex that test case based testing is always insufficient.
  - It is impossible to determine all test cases. There is not enough time.
- The technical risk level of functions is only revealed when the implementation begins, by studying the software.
- It is a way to learn about the system.
- It suits the users' viewpoint.
- Testing is different each time, thus new errors can be found.

Exploratory testing: Problems

- Hard to validate quality of testing.
- Needs skills and guidance – otherwise it is of little use.
  - Badly done it can be *very* bad...
- Little formal proof of testing.
  - Plans, reports are often lacking.
- Methods to use are still not well documented.
- **Usually it is good not to restrict on one testing style – exploratory testing is just one part of the whole testing.**

Exploratory testing: Examples

- Exploring product’s behaviour.
  - How does the just implemented new feature work?
- Agile smoke testing.
- Thorough testing of features.
  - In system testing, acceptance testing.
- Agile regression testing. (In addition to systematic procedures.)
- Analysis of issues behind some observations by further exploration.
  - For example, why did the performance tests produce such results? Execute further tests to understand the behaviour.
- The first phase of usability testing where one most of all tries to understand the new software concept.

Exploratory testing: Application areas

- Agile development.
- Systematic development.
- Customer’s acceptance testing.
  - Some Finnish government agencies have started to use mostly agile testing.

Exploratory testing is based on strategies and knowledge

- Understanding of the software.
- Observation – identification of symptoms of problems and areas that may have errors?
- Strategies – the mentality of finding errors?
  - For example, “breaking the software”.
- Experience – what kind of errors have there been before? What areas of software have previously been problematic?
- Testing is intellectual, challenging work.
- Tester is a detective! Not a test-case typing robot.
- See also the slide set ”Effective error seeking”.

Understanding the software by trying it an making observations

- Eyes open to everything.
- Flow of use scenarios.
- Identification of various elements in the software.
- Identification of events.
  - How the software behaves, how it reacts.
- Identification of states in software.
- Changes.
  - State transforms.
  - Changes in data.

Observations about behaviour

- What is familiar.
- What is new, unfamiliar?
  - What is the logic behind it?
- Reactions of the software.
  - Speed of use.
  - Different sequences.
  - Different data.
  - First time and afterwards.
- Traditional problems in similar applications and situations.

Mentality in testing

- Everything is allowed.
- We know that there are errors; now we just need to find them.
- Trying to break the software.
  - Being hard on the software.
  - Let the software crash – it doesn't matter.
- Use experience.
- Use own "hunch".

Starting points to a test session 1/2

- Goals for the session.
  - Learning or breaking the software or something else?
- Understanding the software.
  - Purpose and use of the software.
  - Purpose and use of the new features.
  - What thing provided the most value to customer now?
  - What are the issues that have the most risk? (From not succeeding or working wrong.)

Starting points to a test session 2/2

- Understanding the co-operation of user and application.
  - What kind of use scenarios can there be.
  - What do we know about typical use?
  - What kind of exceptions can be conceived?
  - What about deliberate misuse?

Mindset

- Exploratory testing is brain work and requires suitable conditions.
- No time pressure (even though there can be a time limit to testing – it is just a frame).
- Realistic thinking about bugs and readiness to see them anywhere.
- Orientation to what is essential.
- Readiness to change thoughts based on new observations.

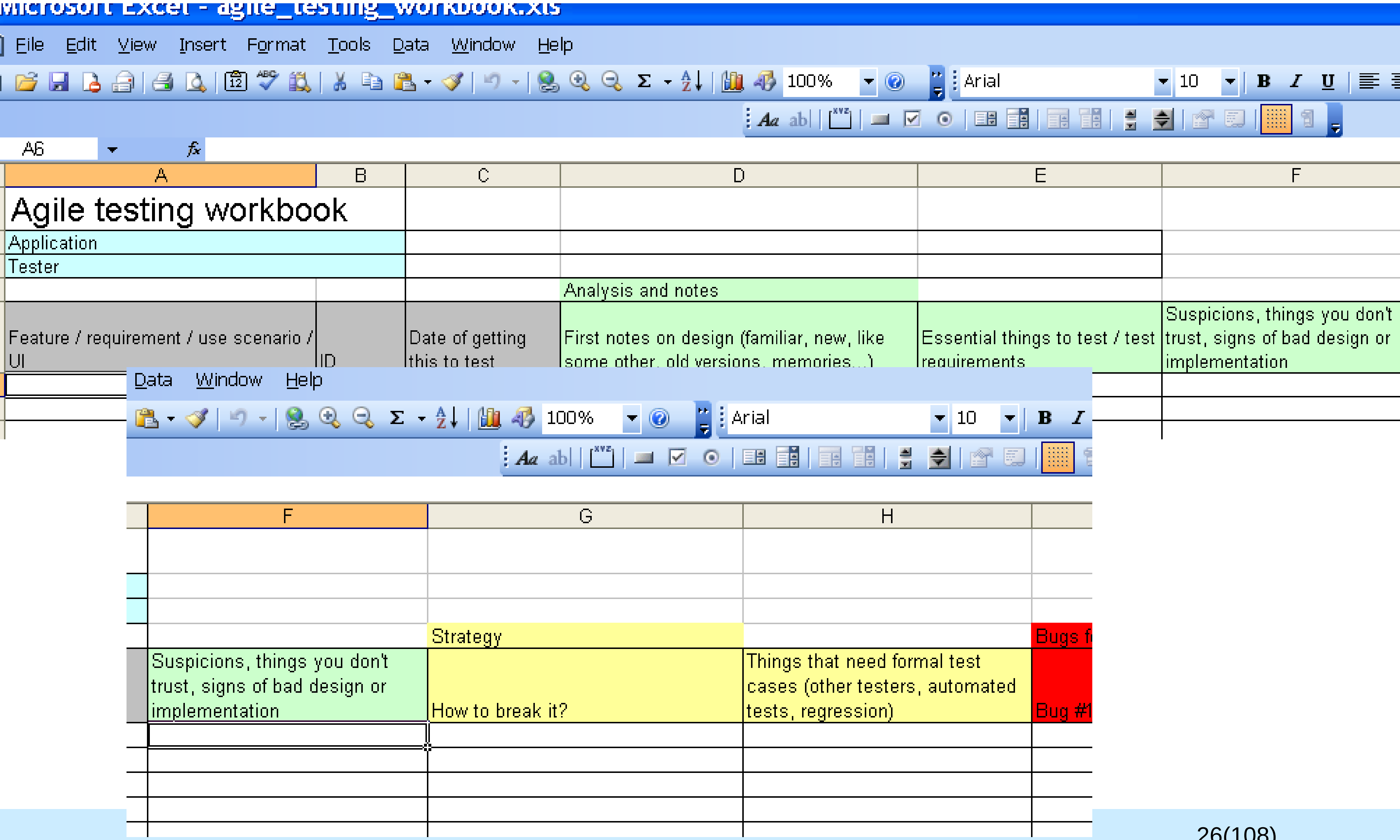
Flow of test session

- Use scenarios, use cases are a good starting point.
- The action patterns of different kinds of users.
- No guidance from strict descriptions – just a framework.
  - Strict following of task descriptions would be systematic testing...
- Making observations, letting the experience guide testing.

Documenting of the test execution?

- Logs of testing are always needed.
- Externalisation of ones thoughts by even writing makes thinking better – and makes for better testing.
- On the following slide is one example of a test log.

Documenting log of testing


Microsoft Excel - agile_testing_workbook.xls
File Edit View Insert Format Tools Data Window Help
100%
Arial
10
A6

A	B	C	D	E	F
Agile testing workbook					
Application					
Tester					
			Analysis and notes		
Feature / requirement / use scenario / UI	ID	Date of getting this to test	First notes on design (familiar, new, like some other, old versions, memories...)	Essential things to test / test requirements	Suspicions, things you don't trust, signs of bad design or implementation

F	G	H	
	Strategy		Bugs f
Suspicions, things you don't trust, signs of bad design or implementation	How to break it?	Things that need formal test cases (other testers, automated tests, regression)	Bug #1

Exploratory testing in everyday work 1/2

- The practices of agile testing are very important.
- In real-life software industry one cannot act based on just one set of ideals – even if American book authors or researches would suggest so.
- Exploratory testing has for a long time been seen as an important part of a well-planned testing project.
  - It has just been called ad-hoc testing.
- One principle of good testing practice is that ways of testing are changed when new observations of the product are made and old ways do not reveal errors any more.

Exploratory testing in everyday work 2/2

- It is essential that exploration also produces new test case that can be included in updated test specifications.
- New information can this way be shared between all testers.
- This reduces risks.
- Choice of terms is important, because exploration is easier to accept in systematic software development than any “ad-hoc” activity.
- In order to be most efficient, this way of testing needs to be accepted as and important testing method and time to do it must be allocated to the most skilled testers.
- *But good testing consists of various styles and practices and one must not let just one style dominate – at least before it is known to work excellently.*

Rapid test design for new features

- Interaction level:
  - Based on user stories, use cases.
  - They act directly as a basis for exploratory testing.
- Logic level:
  - Traditional test techniques – equivalence partitioning, boundary value analysis, decision tables, state diagram based testing.
- Physical level:
  - Monitoring of events programmatically as part of explorative testing.

Being prepared

- Agile testing can use some pre-understanding of what needs to be tested.
  - Lists of things that are usually tested with some kind of applications and functionalities (like list, ”Ohjelmistojen yleiset testattavat asiat”.)
  - With those it is possible to immediately get on understanding of what should be tested.
  - All new systems are in some way an implementation of old ones, of old concepts and ideas.
  - Lists of typical errors are very useful.

Agile testing in non-agile project 1/4

- Modern testing always has agile approaches.
- It is understood that:
  - The project will never happen completely as planned.
  - Testing produces new information that needs to be reacted to.

Agile testing in non-agile project 2/4

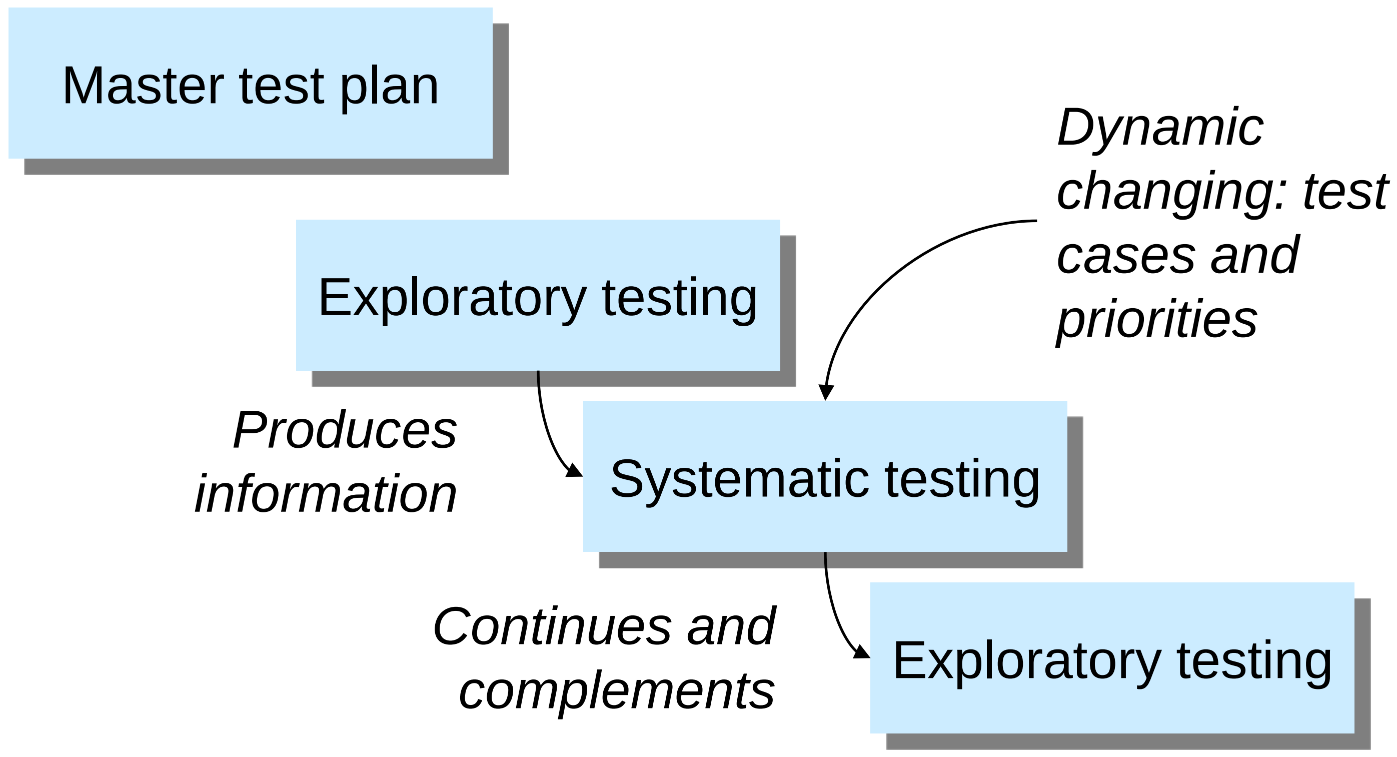
- Test planning
  - W model of testing where the basic approach to testing is planned when the project starts but more detailed testing is planned when the product begins to be in a testable shape.
- Dynamic steering
  - Even is the product has a build plan, testing is adjusted in an agile way to the actual order of implementation.
  - Emphasis of testing of areas of the software are dynamically changed based on how many errors are found and what is the risk level of various areas.
  - The test set used on each test round is really a case by case choice.
  - Test sets are updated based on information gotten from customers and other interest groups.

Agile testing in non-agile project 3/4

- Exploratory testing
  - Testing always includes a free-form part.
  - Getting familiar with a new version is done using exploratory testing. It produces understanding about the targets of systematic testing and is thus also a part of test planning and test design.
  - After systematic testing no longer reveals effectively new errors, exploratory testing is again used more strongly.
  - Test sets of systematic testing are again updated.

Agile testing in non-agile project 4/4

- The whole is a combination of pre-planning and agile reaction.



Fast verification of risk-prone areas

- Ability to rapidly tackle risky things is essential in agility.
- If for example the project implements a new protocol, its functioning is not only verified used by normal testing, but separating the implementation and testing it separately as soon as possible.
  - This way it can be shown that the protocol can be used.
  - The risk associated with it can be closed.

A two-phase model of exploratory testing

1. Testing done to new features in the first test rounds, by which we learn about the product’s behaviour.
2. Agile testing during later test rounds that continues error seeking after systematic tests don’t reveal errors effectively any more. At the same time the test environment is switched to a more “dirty” one.

- Between those, more systematic testing is used.

PET: General and application situation

- Testing of new features in the first testing rounds: First round ad-hoc testing for new functionality – Preliminary Exploration Testing = PET testing
- Application situation:
  - A new build has not been systematically tested. It is not know how it works and behaves.
  - It may have passed automated smoke tests.

PET: Goals

- Learning of new features.
- Studying and understanding of the application
- Making observations of how it behaves.
- Making observation of possible / know problem areas.
- Finding errors!

PET: Tactic and procedures

- Execute full use cases in an “experimenting state of mind”.
- Try everything at least once.
- Test environment can be a clean one.
- Make notes about problems, slowness, state of the system etc…
- Try the areas of the software that have had errors before.
- Identify and report errors.
- Based on the notes, check that the systematic test specification cover all susceptible areas.

FAT: General and application situation

- Exploratory testing of late test rounds: Freeform Adaptive Testing = FAT testing
- Application situation
  - A new build has been systematically tested.

FAT: Goals

- Use experience and error guessing to find new errors.

FAT: Tactic and procedures 1/2

- Use exploratory techniques to find errors.
- Test areas that have had many errors.
- Test “around” official test cases, with a goal of breaking the software.
- Execute full use cases.
- Use a “dirty” test environment. This helps in finding errors.
- Testing is adjusted based on how the product behaves. Concentrate on areas that are slow and have unexpected phenomena etc.

FAT: Tactic and procedures 2/2

- Use imagination in the complex areas or the software.
- Set yourself in a novice’s state of mind (one who doesn’t understand anything about the software) and then to an expert’s state of mind (one who tries everything because there is a right to it). Act like a child.
- Do random things. To things simultaneously. Interrupt things (process in a mobile device, process in PC, interruption done by a colleague!)
- Make mistakes!
- Use other principles of the slide se “Effective error seeking”.
- Identify and report errors.
- Check the test specification and suggest new test cases.

FAT: Example of a rough level test plan

1. Choose a role in which to act (simulate some user group which is essential for the features under test).
2. Select a suitable test environment.
3. Select use cases. Prioritise them.
4. Recognise priorities
   - New features, changes.
   - Areas that have had errors.
   - Priorities of the functions (based on the product and chosen user group).
5. Do an experimental test round
6. Do a test round where you make plenty of mistakes
7. Do a test round where you overload things
8. Adapt to observations and improvise!

Agile software development

- There are many process models for agile development.
- They can be based on strict principles, like the Agile Manifesto, but in real life the models change into more realistic and evolve.
  - The needs of big project are different that the needs of small projects.
  - For example CMMI, ISO 9001 and other process requirements are not the opposite of agile activity, but a set of sensible things that need to be considered in agile processes too.
- Therefore one needs to be careful of the things that are associated with agile development.
  - They need to be adapted and added to in order to make them meet the requirements of given software development situation.

Characteristics of agile software development 1/4

- Incrementality.
  - Development is done in short increments, perhaps in steps of two weeks or 30 days. (The terms used for the steps vary; Scrum methods talks about “sprints”)
- Cyclic nature.
  - The increments are carried out in similar, repeating process.
- Short time-span of planning.
  - Concrete plans are made only for the next sprint.
- A well-planned, simple software process.

Characteristics of agile software development 2/4

- Team's internal dynamics.
  - Teamwork and often pair work (for example a software developer and a tester).
  - Shared ownership of program code and other things.
- Keeping the product in shape so we have a stable version that is easy to redirect and change.
  - After each increment it is tested, documented etc…
  - Test-based development is often used. (For example: Make unit test first, then the implementation code.)
- Communication intensity
  - Situation is known all the time.
  - Plenty of discussion.
  - Wall tables of the team and other information system.
- Customer participates in development – even daily.

Characteristics of agile software development 3/4

- Values (from Agile Manifesto, www.agilemanifesto.org)
  - Individuals and communication over processes and tools (to be noted that in the implementation and testing the tools have a critical role!)
  - Working software over complete documentation.
  - Collaboration with customer over contract negotiations.
  - Responding to changes over following a plan.

Characteristics of agile software development 4/4

- Principles picked from Agile Manifesto:
  - Goal of starting customer deliveries soon and making them regularly. The goal is to keep the customer satisfied.
  - Changes are not frowned upon, but they are welcome.
  - Daily co-operation between business people and developers.
  - The software is the main meter of progress.
  - Developers' motivation. Good working environment and tools.
  - Face to face communication is most effective.
  - Technical excellence and good planning improve agility.
  - It is essential to keep things simple.
  - Self-organising team produce the best results.
  - The teams need to evaluate their activity and develop it regularly.

Quality assurance of an agile software project 1/2

- The developers have the main responsibility.
  - Quality assurance integrated to development.
  - Developers only pass forward quality code.
- Creation of quality in small parts.
- Technical quality is in good shape all the time.
  - Test-driven development.
  - Continuous integration (& integration testing).
  - Other disciplines.
  - Automated “acceptance tests” (verification in the team that customer requirements have been filled).

Quality assurance of an agile software project 2/2

- Monitoring of quality in a tight rhythm.
  - “Pathological” problems can not grow.
  - Quality is visible all the time.
  - Risk level is low and visible all the time.
- Tests measure progress.
- Realistic product understanding.
  - Oral communication ensures that erroneous documents are not relied on.
  - By discussion it is found out, what things mean.

Scrum?

- Scrum has become on de-facto standard of agile software development.
- It is often thought that some practices, like continuous integration and automated acceptance tests are part of Scrum, but this is not true.
- Scrum is above all just a project management method.
- It doesn’t contain software development techniques or even direct requirements to those.
- In each context it needs to be decided what needs to be done during the Scrum sprints.
  - From testing perspective: How can we do all that testing that is actually needed, in a Scrum-based project?

Scrum provides plenty of opportunities to testing

- Scrum is excellent from testing's perspective:
  - Testing can be made to happen in a managed way during the whole project, starting immediately.
  - Scrum culture supports testing where it is most critical: in unit and integration level.
  - System testing has plenty of opportunities because of the rhythmic activity.
  - There are many logical places for user testing (like acceptance testing and evaluation of releases).
- Yet, there are still work in adapting multi-level testing processes to Scrum. (We'll tackle the issues later.)

Good features of agile development models

- Emphasis on lower test levels.
  - Solid code is produced. Regression risks managed in low test levels.
- Test-driven.
  - Tests always exist.
  - Tests are expanded continuously.
  - Test are executed continuously.
- Test automation.
  - Automated unit, integration and acceptance tests are efficient.
  - Regression testing works.
- Testing integrates viewpoints in a natural way: implementation, integration, customer, regression.

Usual pathological problems of testing in agile development models 1/2

- Bad test planning.
  - Co-operation between parties does not represent best testing competence, but a bad compromise.
- Automation drives test planning and design.
  - What can be automated, will be tested.
- The levels of testing and the whole process is neglected.
  - Proper testing in high testing levels.
  - Testing using different time spans.
  - Testing required by deliveries and product management.
- Weak utilisation of professional testers.

Usual pathological problems of testing in agile development models 2/2

- The whole of customer’s acceptance is not understood.
  - Automated acceptance tests are just a small part of acceptance.
  - Sometimes the “product owner” or one customer’s representative is in the role of the customer. In that case the development process may regress to the traditional representative-based development (where the boss will tell how the software should be).
- Assuring of usability not done properly.
  - Testing done by the customer is relied on, but that is not enough!

Agile testing in agile software development

- Now, let’s go into the more challenging world of agile testing…
- In the following slides, agile processes are talked about in general, but still the Scrum process provides some kind of a framework.
- Scrum is not dealt with as it is because the author is part of the school of thought which thinks that all process models need to be adapted to the context.
  - Each company and unit may need a tailored agile process.

Attitude change in testing 1/3

- Testing is in principle against change, because:
  - it break plans.
  - It causes “unnecessary” testing work
  - Changes are always badly documented.
- But in this way, testing is against improvement…
- => In agile development changes need to be accepted and understood that (good) changes are welcome.
- The cycle of increments is quite fast.
  - We need to do software design, implementation, testing, error correction and re-testing during that time.
- => We need to be more rapid in planning, implementation, execution and communication of testing.

Attitude change in testing 2/3

- Ways of communication need to be re-thought.
  - If the agile project emphasises oral communication, testing needs to get to be a part of it.
- => A separately located test team can be impossible (depends on the team’s task). Testers need to live in the team. This causes changes to the testers’ style of action.
- Yet, part of testing is done separately from the development team.
- The development team has strong interaction with the customer.
  - Perhaps daily.
- => Testers also need to be able to work with the customer.

Attitude change in testing 3/3

- Terms and concepts change.
  - Agile development process may have different understanding of what for example “acceptance testing” means.
- => One needs to accept that the development process and project define all concepts and terms used.
- But agile processes are not ideal!
  - They are often in some way lacking and short-sighted.
  - Masters of testing have not been developing the processes.
- => Testers can help in developing the processes better!

V model still essential

- Agile software development is by nature different that waterfall based development of traditional incremental development.
- It is also developer-driven and emphasises creation of robust code and continuous component level integration.
- All testing levels presented by a V model are still essential, like:
  - Unit testing, integration testing, system testing, system integration testing, acceptance testing.
  - They can be implemented in a new cyclic style, but the existence of levels and their requirements is important to understand.

Unit testing in agile development

- **The common consensus is that good unit testing, in a test-driven way, is a requirement for success of agile development.**
- Agile development models are mostly test-driven.
- Before writing code tests are written for it in a test program.
  - Often a test class that corresponds a class in production code.
- Execution of tests is integrated to a part of the developer' personal build process.
- Testing starts before coding.
  - At the first time the tests fail.
  - As the implementation proceeds, the tests will start to pass.
  - Pass rate of the tests show progress of implementation.
- The test application produces automatically reports of testing.

Integration testing in agile development

- Agile development processes always use either continuous integration or integration that is done very often (daily or so).
- Otherwise the software can not be managed in the fast cycles.
- A strategy to manage regression risk: the software is all the time in working order during development.
- Integration tests are usually automated.
- They mainly consists of the developers’ unit test sets.
- The idea is that the integrated code works and will not cause integration test errors. The tests in integration test environment have already been simulated in the developer’s workstation.

System testing in agile development 1/4

- (Here it is discussed in general and from the viewpoint of functional testing.)
- In agile development, system testing should be quite similar as in non-agile development.
- The time-span and scope of test planning is essential.
  - Plans are made for the next release – perhaps already in two weeks.
  - Because the specification for the next release is clear, test planning is done immediately.
  - Test planning is done in co-operation with the developers. Perhaps they can automate many of the tests?
  - The plan is small and focused, as it only applies to the new features.

System testing in agile development 2/4

- Because of the time span, testing starts as soon as possible.
  - As soon as the feature has been integrated.
  - A new build is not produced for system testing – there is a working build available all the time.
- Because of the dynamic nature of the process, testers are mostly integrated in the development team.
  - As soon as the team has produced something, the testing begins, usually with agile testing.
  - Direct interaction with designers in error correction.
  - However, an error database is important too, to measure the product’s status.
- Testing is a mixture of agile and non-agile testing.

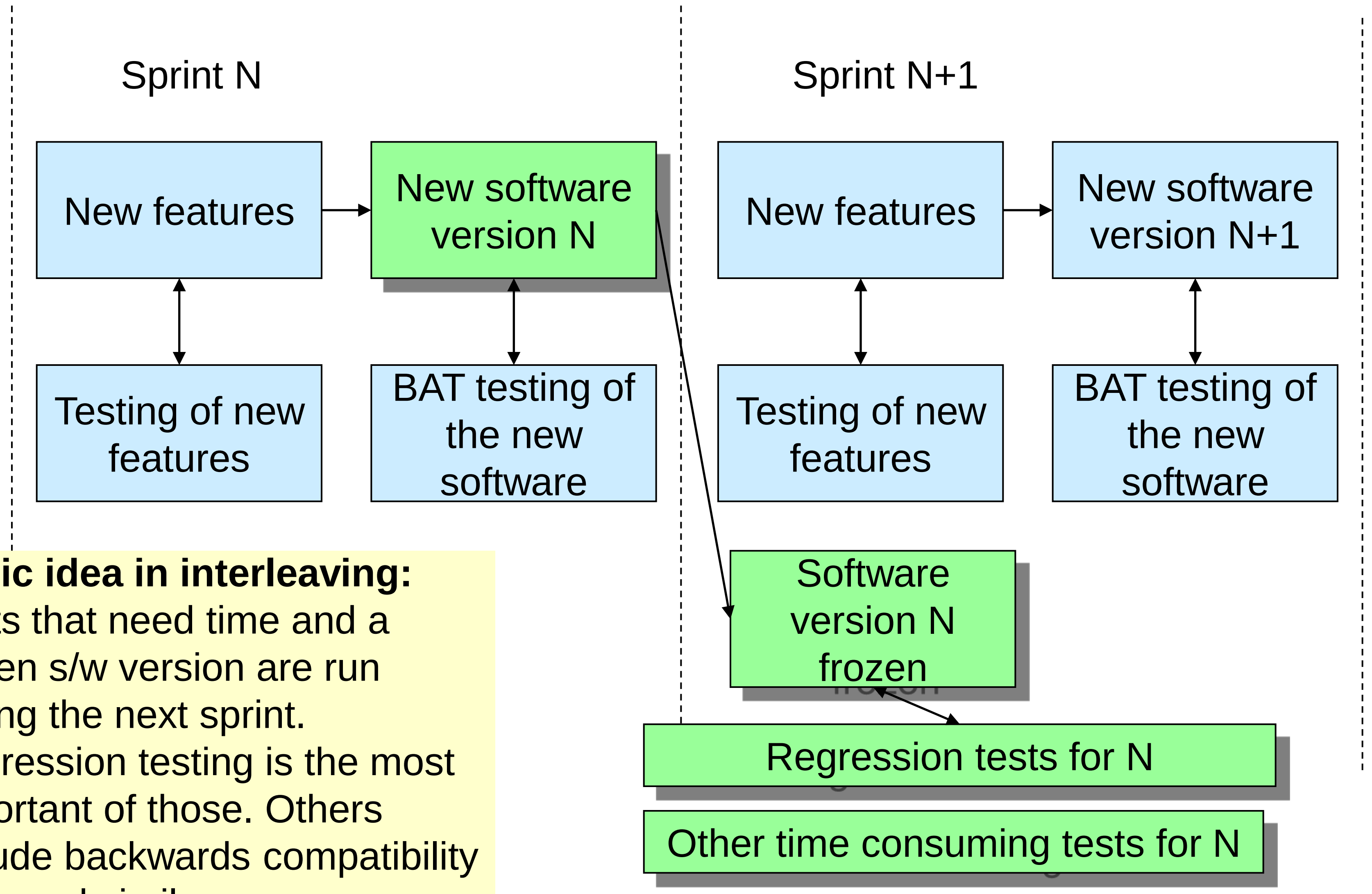
System testing in agile development 3/4

- Interleaving testing over sprints.
  - It is essential to maintain the rhythm of the developer team.
  - All testing that the developers have a role in, needs to be done during the implementation sprint.
  - But a more thorough testing, done by the testing team, and all special tests can be done during the next sprint.
  - At the same time, the test environment can be changed. This is essential!

System testing in agile development 4/4

- Testers’ tasks in the same sprint:
  - Basic testing of the new features using agile testing.
  - Immediately, when the developers give an impulse that something can be tested.
  - Systematic testing of new features starting with positive testing.
  - Fast, so that developers get support for their work.
  - Helping the developers to automate some of the tests.
- Testers’ tasks in the next sprint (if needed):
  - A more demanding negative testing, interoperability testing, robustness testing, system level regression testing, etc...

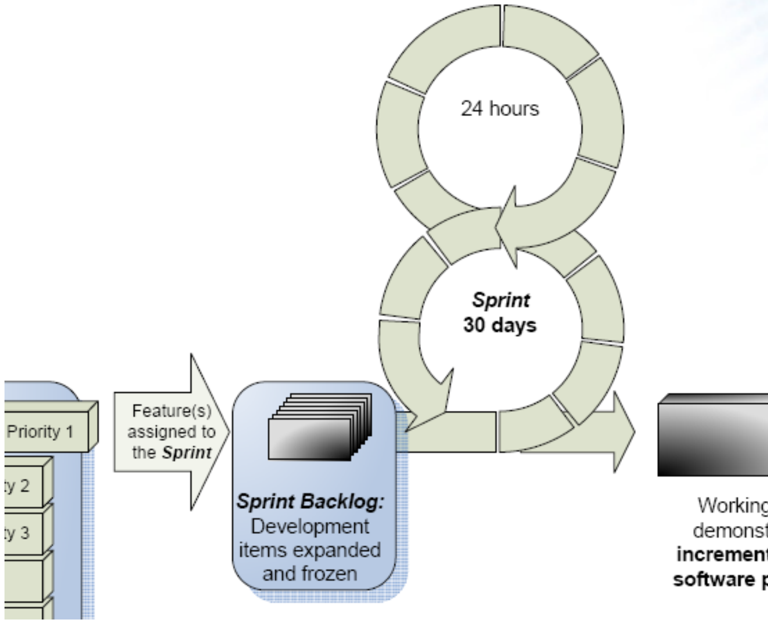
Summary of interleaved testing



Basic idea in interleaving:
Tests that need time and a frozen s/w version are run during the next sprint. Regression testing is the most important of those. Others include backwards compatibility tests and similar.

Side note: Scrum and cycles of control 1/2

- Scrum knows three cycles of control:
  - Day.
  - Sprint.
  - Whole Scrum.
- Interleaving of tasks to many sprints doesn't fit into its philosophy.
- What to do?



24 hours
Sprint
30 days
Priority 1
y 2
y 3
Feature(s) assigned to the Sprint
Sprint Backlog: Development items expanded and frozen
Working
demonst
increment
software p

Side note: Scrum and cycles of control 2/2

- Solutions:
  - Have fewer implementation tasks during sprints so that everything can be done during the same sprint. Sometimes this is possible.
  - Practice must be set before philosophy.
  - What is essential is the team dynamics. The basic team has one rhythm, but above it can be other rhythms and continuous processes.
  - Other agile processes may have different cycles.
- The basic idea is most important: All necessary things need to be done in projects and the basic processes need to be adjusted to the needs.

System integration testing in agile development

- System integration often causes most problems in projects.
  - The thing that a Scrum team produces works well, but the teams’ deliverables don’t work together or with other systems.
- System integration testing is a pair to system testing.
- Often it is a part of the customer’s acceptance testing.
- It takes time, trials and practicing and creation and management of environments.
- It needs to be started in the first sprint.
- It is a task for separate testers / team.

Usability testing in agile development

- Usability analysis and testing is traditionally not well done in agile processes.
- The developers trust the participating customer who is just an individual and does not represent all targeted user groups.
- Traditional usability analysis and testing methods can be applied in agile development.
  - Before the first increment of during it an UI concept is created, which requires analysis.
  - Fast analyses are done to new UIs in all increments.
  - A user test is done when a new UI is created or an old one is significantly changed.
- Test planning and reporting of usability testing are notoriously slow.
  - Such people need to be used who can work fast.

Performance testing in agile development

- It is essential that after choosing the basic architecture its sufficient performance is verified.
- This means that critical architectural issues should be implemented and tested during the first increments.
- Testing is repeated when the architecture changes and when implementing customer releases.
- Importance of performance testing in later phases is affected by the targeted use of the later releases – sometimes their required performance capacity is show just by the fact that they can be used.
- Automation of performance tests and running of them automatically suits well the culture of agile development.
  - Well planned basic tests can be run even daily.

Information security testing in agile development

- In assuring information security it is essential to deal with the level of the users' activity system.
- It is tackled with an information risk analysis, which can largely be done already in the concept phase, before incremental development.
- One needs to update the concept level analysis during the last sprints.
- Experts must participate in the creation of testable information security requirements.
- Verification of those is done dynamically at the time of their implementation.
- Software developers shall have the main responsibility.
- Additional analyses and tests are done by experts of information security testing at suitable stages of development.
- Main principle: Security of all customer deliveries must be ensured.

Validation of safety in agile development 1/2

- This applies to safety critical automation systems and standard IEC 61508.
- Validation of safety requires most of all good total process, that is based on the thoughts of the ”old world”: planning, documetation, validation and verification, configuration management etc…
- This is the world of ISO 9001, CMMI, IEEE standards.
- Can this happen in agile development?
  - Sure, but it will take some effort…

Validation of safety in agile development 2/3

Requirements for validation	Effects on processes
Validation of safety features against safety requirements	The requirements need to be identified and described. Simple XP style user scenarios are not sufficient as requirements. Safety analyses need to be made to the user scenarios.
The object of validation is the whole system	The software will be implemented well, but how it works in the system is what really counts.
The configuration that is validated needs to be known – change management essential	All changes need to be recorded, their effects and risks need to be analysed and testing planned. For the last validation tests the software needs to be frozen.
The whole quality assurance chain needs to be in shape – analyses, tests…	For example code reviews, FMEA, proper system tests need to be included in the process

Validation of safety in agile development 3/3

Requirements for validation	Effects on processes
Reliability of safety functions needs to be sufficient (mean time to failure MTTF etc.)	Reliability needs to be analysed. After changes the analyses need to be updated. Effect of refactoring needs to be analysed
Validation plan with criteria. Most essential criteria is the integrity level	A quality assurance plan must be made and within it the validation criteria defined. The document can live during the project, so it must be updated in sprints.
Planning and documentation of testing is important	System testing can not be just exploratory testing. System tests need to be designed (test conditions, techniques) and documented. Agile testing can be applied on top of that

Regression testing in agile development 1/3

- Characteristics of an agile project:
  - Constant change.
  - Refactoring of things that work, implementing them as needed to be better, more maintainable etc…
- This causes a huge regression risk.
- On the other hand, the whole process has often been build to stand regression well:
  - The automated test sets in unit and integration tests capture most broken things.
  - Increasing UI level test automation in the “acceptance testing” of agile processes enables catching regression also in system testing level.

Regression testing in agile development 2/3

- But what about the situation where testing has not been automated enough – as the situation often is.
- Then more emphasis to regression testing is required.
  - During the next sprint after implementation.
  - As a continuous process around the incremental development loop.
- Regression testing of each customer release need to have sufficient coverage and quality.

Regression testing in agile development 3/3

Level of test automation		
Unit testing and integration testing are automated. Lots of system level tests automated (smoke tests, basic UI tests, regression test set)	No freezing of s/w Automated at the end of sprint Manually during the next sprint – system and system integration	Freezing of s/w Automated at the end of sprint Manually during the next sprint – system and system integration
Weakly automated	Freezing of s/w Manually during the next sprint – system and system integration	Freezing of s/w Manually during the next sprint – system and system integration
	No special reliability, quality or safety requirements	Critical level requirements
		Requirement level of s/w

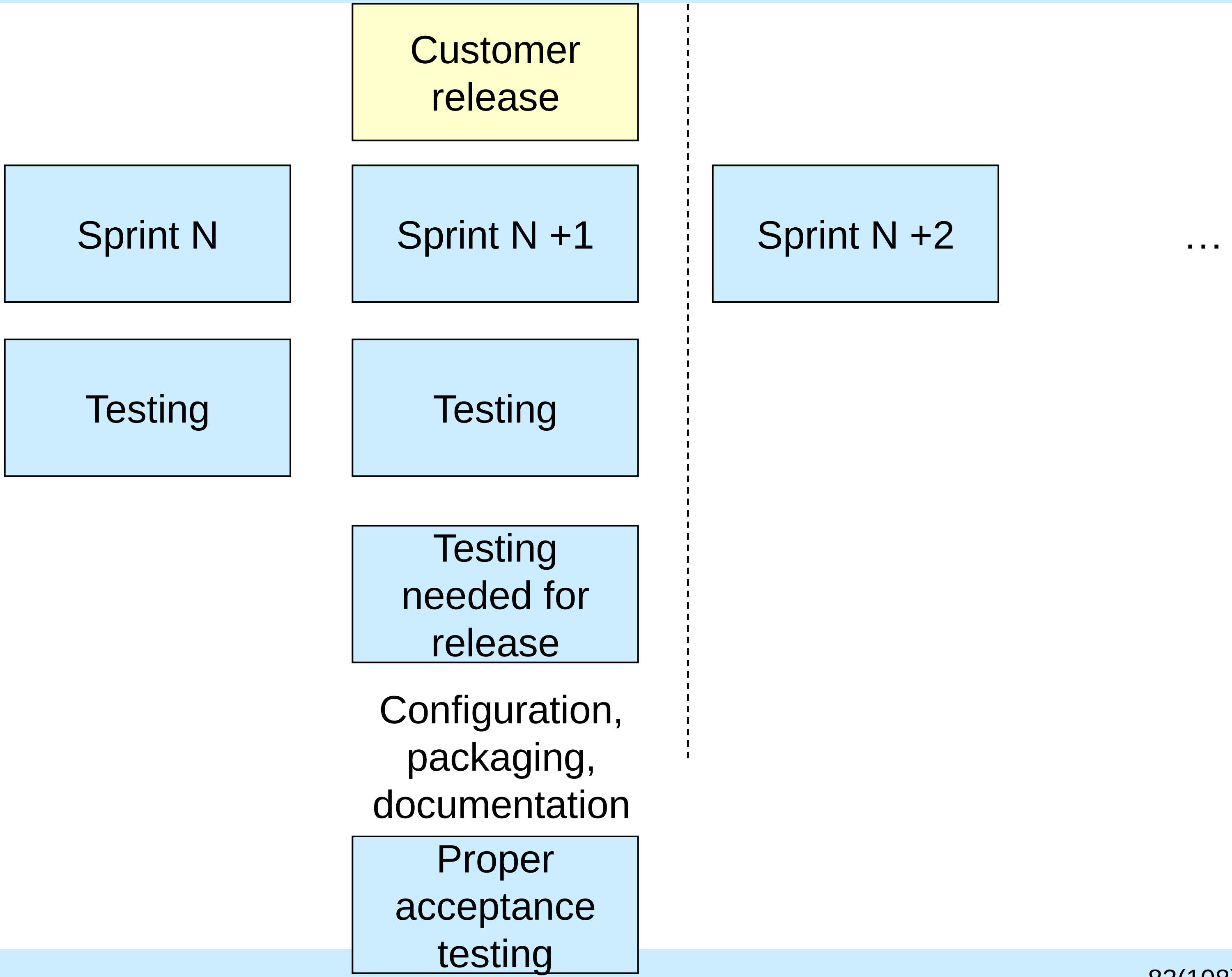
Agile regression testing?

- Regression testing is usually very systematic.
  - Test that are targeted to testing of just the things that have been changed and their possible effects.
  - A pre-designed standard regression test set
- These are either automated or implemented as manual test cases.
- This can be complemented by agile testing of for example
  - Things that are hard to express as test cases.
  - Full use scenarios.
- Observations made by a skilled tester is always a benefit in any kind of testing, because systematic tests are always just mechanistic samples.

Various kinds of sprints 1/2

- In an incremental project, in some increments the testing can be very agile, but for customer releases (at least the final one) targeted to production, testing needs to be very systematic.
  - The focus is a comprehensive testing of a solid product.
- Because the rhythm is essential, there should be a rhythm for customer releases too – for example every third sprint.
  - This way the special actions required can be spaced in a suitable way.
- So, a release plan is needed. The most essential feature of them is the rhythm – the contents are not know well beforehand.

Various kinds of sprints 2/2


Customer release
Sprint N
Sprint N +1
Sprint N +2
…
Testing
Testing
Testing needed for release
Configuration, packaging, documentation
Proper acceptance testing

Acceptance testing in agile development 1/4

- Customers give some kind of acceptance for each sprint, at least by a review.
- Customers releases should have customer’s acceptance testing.
- A traditional agile practice is that the customer’s representative defines the acceptance tests and they are executed by the customer, testers or developers. These are often automated.
- **This is completely insufficient.**

Acceptance testing in agile development 2/4

- Those tests mainly validate the implementation of a requirement.
  - Some report is produced correctly.
  - A form can be used to enter data in a database.
- This is acceptance only in this context.
  - Not the customer’s organisation’s acceptance!
  - Really only just a smoke test – after that can the real testing begin.

Acceptance testing in agile development 3/4

- A real acceptance testing means acceptance by the customers and users.
- It is essential to do using an open viewpoint, in sufficiently diverse ways, in a non-mechanistic style.
- It must not be made by the client’s representative, but the customer’s organisation needs to participate in doing:
  - Professional test plans.
  - Professional testing.
  - Professional decisions about acceptance.
- In information system development an acceptance testing environment is also needed from the customer. It may consist of many environments of varying complexity and connectivity.

Acceptance testing in agile development 4/4

- Acceptance testing is quite a continuous activity.
- As the sprints produce new versions to the client in a steady rhythm, their testing can be of many types:
  - Acceptance of new features.
  - (…)
  - Testing the new version in practice, in production use.
- For example information systems can have many test levels and a new software version passes many customer’s test levels and test environments before it is found out to be production-ready.

Organisation of testing

- The bulk of testing is integrated to the development team.
- Testers are part of the team.
- Testing teams are needed / can be used in for example:
  - Comprehensive functional testing in system testing level.
  - System integration testing.
  - Interoperability testing.
  - Release related testing, like configuration testing.
  - Any special tests.

Tester’s various tasks

1) Testing consultant to the project.
2) S/w developers support.
   - How to test the increment’s results, what tests to automate.
3) Agile testing of new features.
   - Rapid revealing of errors.
4) Systematic testing of new features.
   - Basics during the sprint.
   - Time consuming tests during the next sprint.
5) Regression testing.
6) Testing of non-functional characteristics.

Tester’s role 1/2

- Often integrated part of the team.
  - Essential, because otherwise is not part of the oral communication.
  - What is lost in objectivity and independence is gained by interaction.
- Role in co-operation:
  - The consultative role is emphasised: the tester should help developers do good designs and implementations.
  - Tight schedules and light documentation require good co-operation.
  - A supervising role: Someone needs to watch out that requirements are understood correctly and quantified for testing when needed (for example performance requirements).

Tester’s role 2/2

- Viewpoint to test levels:
  - Testers’ most important view is still in system testing.
  - In Scrum-process the tester can with the client plan the “acceptance tests”.
- Viewpoint to test planning:
  - Key person in drafting the master test plan.
  - In Scrum-process the tester plans with the client the “acceptance tests”.

Requirements of agile testing to testers 1/2

- Good skills.
- Strong professional self-respect,
  - Equal status with developers (in agile development) in the team.
  - Courage to make fast decisions.
- Skills with many kinds of testing techniques.
  - Personal ”toolbox”, from which to choose suitable methods as needed.
- Dynamic between plan-based and exploratory approaches.
- Understanding of users thinking and activity and ability to simulate different kinds of users.
- Build / release orientation – one must assure the quality of the frequent customer releases.

Requirements of agile testing to testers 2/2

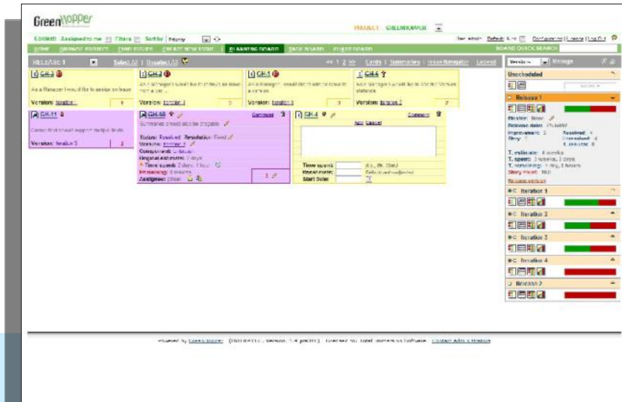
- Ability to stand insecurity and constant change.
- Co-operation skills – with developers and customers.
- Ability to change working style according to the project’s needs.
- Understanding of the product technology and product concept.
- Ability to learn new things fast.
- Ability to rapidly figure out essential things to test in new features.
- Intelligence.
- Speed – the work on increments must be started immediately – and the errors must be gotten to get fixed and re-tested.

Testing teams?

- Testing teams are still needed.
- A counter-force to developer-driven projects.
- Experts’ co-operation in planning of special test and managing the whole of the testing.
- Test levels from system testing and above are the testing teams tasks.

How does the test team get information?

- Representative in development team’s meetings.
  - Video feed to others?
- Electronic process control.
  - Information systems instead of wall charts.
  - JIRA to control task.
  - Applications to replace wall charts (for example GreenHopper plugin for JIRA http://www.greenpeppersoftware.com/confluence/display/GH/FEATURES
- Co-operation with developers.
  - Specified work pair, even when working remotely?



Test plans

- Even agile testing needs a stable, planned context:
  - Master test plan.
  - Quality assurance plan.
  - Project plan.
  - Behind those, a process model that defines the general principles of testing.
- Each increment may require its own short test plan that is based on what is implemented in the increment.
- Customer releases require a plan and checklists.

Co-operation in test planning

- In many models developers, testers and customer plan tests together.
- So called “acceptance tests”.
- Those tests describe how the new features should work.
- Tests also document details.
- In the ideal agile world, this results in automated acceptance tests.
- There is a danger that test planning stops at this – designing challenging negative tests requires test planners’ special skills.
- There is a danger of degenerating to the traditional “sales director specifies the test” principle.

Risk analyses 1/2

- In the traditional software development the following are made:
  - Project risk analysis, considering also requirement specification.
  - Risk analysis for the testing project (system testing team).
- Project risk analysis is just as essential in agile development.
- Besides risk analysis of test project, the general risks of validation and verification are emphasised:
  - Is the project correctly planned in this regard?
  - Is the process suitable for the product and releases?

Risk analyses 2/2

- To each increment it is good to do a rapid risk analysis:
  - Risks of the increment.
  - Risks of new features.
  - Testability of new features.
  - As tool for analysis for example Software increment risk map.

Prioritisation of tests

- For example Scrum is based on prioritisation.
  - Most valuable things are implemented in first sprints.
  - Other things that have the most risk should also be implemented soon.
- Prioritisation of testing follows this main line.
- Essential prioritisation principles:
  - Value to customer drives implementation – is the value going to be realised?
  - Risk based thinking.

Test management

- In ”implementation close” testing it is integrated in the agile process / task management.
  - Tasks are not done until the new feature is also tested.
  - For example the burndown chart used in Scrum.
- This will cover the testing done within the team.
- System testing and above needs its own management.
- Release testing needs its own management.
- So, traditional test management is still working and needed.

Key principles of test management

- Test critical things in first increments.
- Ensure that unit and integration testing works.
- Monitor the automated acceptance tests.
- Associate system level testing tasks to development tasks.
- Ensure regression testing.
- Ensure that all features are tested properly, more than just the automated tests.
- Manage the testing tasks that are interleaved over sprints.
- Emphasise change control.

Error management

- There is a temptation to makes errors just tasks in the project’s backlog.
- One needs to differentiate errors and their systematic management and the tasks that are opened to developers.
- Thus: a normal error database is required.
- The error lifecycle needs to be adapted to the project’s needs.

Test logs

- There are new needs for test logs in agile development.
- As testing is planned in a more light and dynamic manner, the logs do now have the situation of the “old world”, where logs just sign tasks to be done.
  - Now they document the substance what has been done.
- As part of the test logs it is good to have a diary type log, which document test tasks which are done.

Synchronizing testing and development

- There are ways of synchronizing which have different rhythms
  - The process meetings especially at the start and end of sprints
  - Information exchange between individuals
- Synchronization must be emphasised to ensure that testing always tackles the right things
- Some organisations have weekly synchronisation meetings between testing and development
- And of course, inside the team, synchronisation is continuous

Requirements of agile testing to organisation 1/2

- A well thought out, stable software process and its associated testing process.
- Strong skills – if things are not managed, agile turns into chaos.
- Strong positive attitude towards quality.
- Understanding of the what good quality comes from – robust code, customer approach, architecture.
- Understanding and supporting of the success factor of agile development.
- Development that has a positive attitude towards testing.
- Understanding of all kinds of testing that is required.

Requirements of agile testing to organisation 2/2

- Keeping the software in working order all the time so that it can be redirected rapidly. Bug count should ideally be zero.
  - Fix old errors before developing anything new.
- Good communication.
- Continuous documenting.
- Ready made models for testing different things.
- Light documentation models for testing.
  - Plan templates that are easy to use in planning of how a small increment is tested.

Summary

- Agile test methods are important but they are just a part of the toolset of professional testing.
- If agile is implemented badly, it becomes unmanaged chaos.
- Agile requires strong project management and test planning.
- Importance of personal skills is emphasised in agile.
- The ideals of agile processes can not be relied in practice, because all necessary elements are seldom present.