Testers Introduction to agile software development

Anko Tijman, Eurostar 2003

The prestige of 'Agile' Software Development Methods has risen sharply in the past few years. The best-known example of this is eXtreme Programming (XP). But agile development is more than just a countermove by nerds, as this article will show you, while we will also deal with the role testers play in this process.

Current situation

The figures with respect to the current status of our software development projects are clear: in 1995 the Standish Group [1] investigated the results of ICT projects and had to draw some shocking conclusions: 31% of the projects were discontinued prematurely and thus made nothing [thus did not produce the desired results]. Almost 53% of the projects came to an end with an overspending amounting to 89% of the original budget. And only a mere 16% of the projects were completed in time and within the budget. All projects were, to a greater or lesser extent, based on the phased waterfall method. In 2001 this research was briefly repeated; it was then shown that some 72% of the projects could not be called successful.
Of course, figures don't say a lot; for example, you can also explain it in a positive way when a project is prematurely stopped. However, considering the number of uncompleted projects in combination with the experiences of many ICT people, it appeared that something was really wrong. So the actual practice was at the point that there were bound to be reactions.

The Agile Manifesto

In February 2001 a group of people met who had the same goal in mind, i.e. creating a common basis for the ideas they had individually. The group included people like Kent Beck, Ward Cunningham, Ron Jeffries (eXtreme Programming), Ken Schwaber (Scrum), Alistair Cockburn (Crystal), and Arie van Bennekum (DSDM). It became a meeting with a special chemistry between the participants, and led to surprisingly much agreement of highly qualified people who yet had each (co) developed their own methods.

In the end the Agile Manifesto was framed: a description of the common standards and values set to software development:

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.

In addition, the principles arising from this manifest were framed:

Our highest priority is to satisfy the customer through early and
continuous delivery of valuable software.
Welcome changing requirements, even late in development.
Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months,
with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.

Build projects around motivated individuals.

Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development.
The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity--the art of maximizing the amount of work not done--is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Whereas the waterfall method is often characterized by excessive documentation, little contribution by clients, progress measurement through intermediate products (also often documents!), and many management reports by the project manager, this wind is blowing from another direction. We all know the results: projects overrunning their time, dissatisfied clients, too little time for testing, etc. The new development methods endorsing the Agile Manifesto [2] are aimed at customer satisfaction, at being able to respond to changing demands, at supplying working software as an intermediate product, and writing documentation if this contributes to the final product. Agile software development is characterized by iterative development, an incremental approach, feedback and much customer input. And, besides all this, there is communication, communication, and more communication.

We will now briefly describe three agile methods, the originators of which all subscribe to the Agile Manifesto. First, eXtreme Programming, a method that particularly emphasizes the practices performed by the team members. Next, Scrum will be described, an agile project management method. Finally, Crystal will be discussed, an agile method to be able to scale your development process per project. The role testers can play in each of these methods will be explained.

Extreme Programming

The initiator of XP is Kent Beck, with his book Extreme Programming Explained [3]. This book explains the basic ideas and values of XP, being based on four values:

- Communication
- Feedback
- Courage
- Simplicity

An important point within XP is the strong division between the decisions made on the basis of business interests (the customer role) and those made in the interest of the project (the developers role). Besides, the start of a project will always be characterized by simple architecture that evolves as time goes. All this based on the fact that it must be a system ‘fit for purpose’ – nothing more, nothing less.
Extreme Programming consists of twelve techniques, with each technique (practices) originating in one of the four values:

- Planning Game – a dialog between the customer and project team (!); in this dialog, decisions are made as to the size of the iteration, the tasks to be performed, the priorities, the time allotted to each individual task, and the time of release. The tester can play a role in this by asking many questions to both the customer (details about functionality, acceptance criteria, making no assumptions) and the programmers (has an inventory been made of all tasks, have test tasks been planned, etc.).
- Small releases – only those tasks that yield the most business value should be part of a release. In order to create extra value to the customer as quickly as possible, releases are made on a regular basis; in this way, risks of a big bang scenario are restricted. The tester can help the client determine the priorities of the tasks and, each time he will have to run the full script of acceptance criteria in order to give insight in the quality.

- Testing – all software will be developed according to the TestFirst principle: first a unit test is written, and only afterwards the actual code. The unit test serves as a mechanism to get the proper operation of the functionality going, and it will prevent regression test mistakes once applied in an automated framework. The unit tests therefore should always be 100% successful in order to be able to guarantee proper software performance. Together with the developer, the tester can compose test situations for the unit test and he automates the acceptance tests. The tests will give a decisive answer (feedback!) about the project's progress. The customer can decide to accept the score of successful acceptance tests being less than 100%, for this strongly depends on the sort of mistake and the importance of the new functionality. If need be, the system can be released with known errors.
- On-Site Customer - XP prescribes that the customer is fully available for the project, in order to answer questions, to solve differences of opinion, and set priorities. The customer writes so-called UserStories: concise descriptions of the purpose of the functionality. No extensive detailed descriptions, but brief, pithy sentences, preferably on postcard format. The (possible) lack of specification is set off by the customer's full availability for explanation and decision-making. The tester is the customer's representative with the programmer, and the programmer's representative with the customer. In this, his key task is to translate each other's demands and wishes.
- Metaphor – a description in a few sentences of the higher purpose of the project, which may at times be abstract.
- Simple Design – only doing what you know you have to do. Don't anticipate what's coming, for you don't know.
- Refactoring – in order to be able to continue to produce business value, you must sometimes have the courage to change a code around. Do this if it is necessary; this keeps the design simple. The unit tests should continue running.
- Pair Programming – reviewing code is a good practice; why then not 100% reviewed code? Programmers are working together on the software to be developed and take turns. Testers can test together with testers or developers (of course it's called Pair Testing then!).
- Collective Ownership – when a programmer sees an improvement can be made in existing software, he must do so – with the aim of being able to supply extra value to the customer.
- Continuous Integration – quick integration of software means small problems. Small mistakes can be solved quickly. It is necessary to frequently integrate to be able later to produce software that is working in a quick and efficient manner.
- 40-Hour Week – to keep people fit, clear-headed and sharp, restrict working overtime to a maximum of one work week; more overtime is useless.
- Coding Standards – when every code is well readable and understandable for each programmer, this will improve the quality of his work.

Observing all these practices will automatically create an atmosphere in which the complete team is directed at quality and testing, and not just the testers! So far the tester's role has been described in one book only [4]. In actual practice, however, it will rarely happen that –as described by XP- a customer is full time involved in a project, is capable of framing detailed acceptance criteria by himself and of executing those tests as well. Adding testers to the project team will lead to smoother relationships between programmer and customer as they can act as intermediaries between the two parties.

Scrum

The Scrum method (in rugby the method of beginning play in which the forwards of each crouch side by side with locked arms) is based on the so-called empiric model of process control. Its originator Ken Schwaber [5] was looking for answers to the question why the traditional approach to many projects failed. The waterfall model is based on the 'defined' model. The defined model of process control prescribes that it is necessary to have a complete understanding of each part to be made. With a well-defined set of inputs the same outputs are generated. A defined process can run from the beginning to the start and generate the same results in this process.

As said, Schwaber starts from the empirical model of process control. System development has got so much complexity and unpredictability that it must be checked on the basis of a model that expects the unexpected, i.e. the empirical model. Since the processes cannot be defined in detail, generate unpredictable and nonreproducible results and each project is different, checking can only be done through frequent inspections and adjustments.

All this led to the Scrum project management method Scrum. Scrum roughly consists of three aspects:

- Backlog – the accumulation of tasks that should be performed within the project – not only the programming tasks, but also the testing and documentation tasks can be accumulated. The tester may add his own tasks here and ask critical questions if it concerns the intended quality and functionality.
- Sprint – an iteration of one month in which part of the tasks is performed; in advance it was determined which tasks. During the sprint no disturbances are tolerated when it involves the contents of the sprint. At the end of each sprint a brief reflection is made: the 'retrospective'. At this time matters can be raised so that the next sprint is even more successful.
- Meeting: the complete project team will meet preferably each day to discuss the project's progress. The fact that everybody is there and the meetings do not last more than 15 minutes, problems are quickly observed. The problems are not solved during the meeting, but is taken up by the people concerned themselves. During the meeting three questions are at the forefront:
    1. What did you do since the last meeting? This involves measuring of the daily progress.
    2. Do you have any obstacles? At this point possible obstacles come to the fore. For example, in- and outchecked software where programmers stand in each other's way or cannot test it yet because one object is still lacking.
    3. What will you do before the next meeting? The purpose of this is that the participant commissions himself to his tasks instead of the project manager giving him the order and planning.

The project manager using the Scrum approach will be more inclined to concentrate on the inside of his project than on the outside: after all, the starting-point is the unpredictable of software development. Yet the approach is much more aimed at getting people involved instead of presenting the team members with their tasks and planning. With respect to the Backlog, the tester aims at inventorying test tasks so that these can be planned and performed in a better way. Also, it is the tester's nature, when drafting the tasks, to pose critical questions in time and observe potential problems in time.

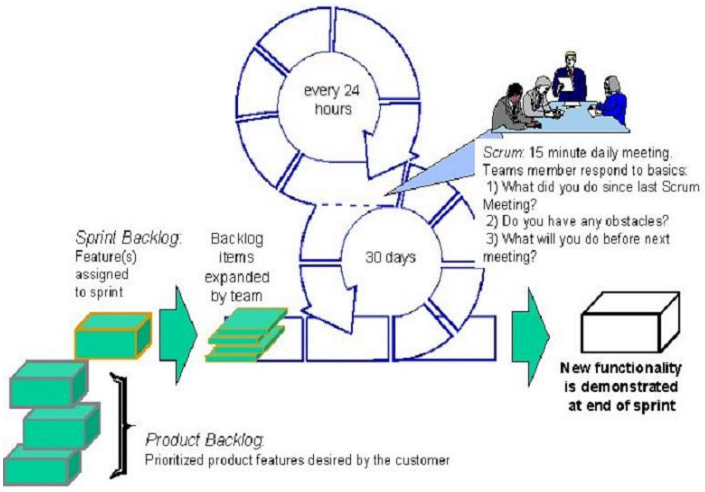


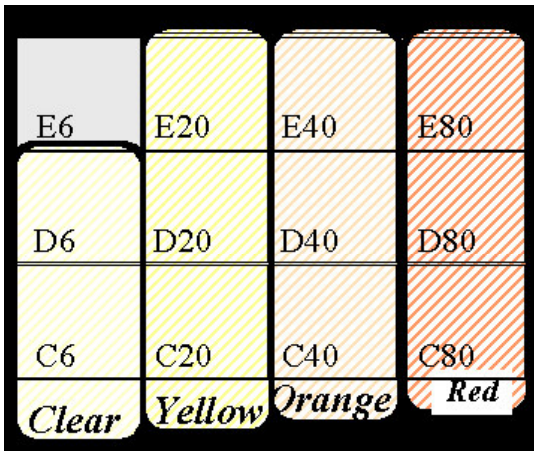
Fig. 1: The Scrum process

Since the tasks to be tested are often of a smaller size, the risk of mistakes is smaller. However, after a number of sprints many tests will have to be performed, a matter that should be taken into consideration. Automated test scripts can be helpful tools in this. The most important matter is that by way of his activities the tester provides feedback to the project's progress.

Crystal

In many projects, communication is an uphill battle, processes outstrip people and one method is used – irrespective of the fact whether it is the right method for the project. In Crystal, people are at the centre;

it is an adaptive and scalable “shrink-to-fit” method. In a project, people must become people again, communication with each other must occupy centre stage, and the method must be tailored to the project.
In his book Agile Software Development [6] Alistair Cockburn (pronounced Co~burn) explains the Crystal methods. Just like a crystal changes colour under the influence of light, likewise the process under the influence of external circumstances. For example the importance: if the project is not critical and few people are involved in it, it is sufficient to have a very limited description of what you are going to do (procedures) and you need to build in little method. Contrarily, if the project is intended for a life-critical application and dozens, if not hundreds of people are involved in it; this requires much more checking of the process. When the project’s size changes and becomes larger, more method must be applied. In the light of Crystal the colour of the project will then change as well. Up to six people it will be Crystal Clear, up to about 20 people Crystal Yellow, up to about 40 people Crystal Orange, etc.



As the risks of the project increase, the solidness of the process will have to be geared to that. Crystal distinguishes four levels: Comfort, Discretionary, Essential, and Life. It may be clear that a project for the website of the –proverbial – local basketball team is of a completely different order than a project for the new version of pacemaker software. Cockburn prescribes that actually a new method description should be made for each project; at any rate, it will need critical consideration. Even during the project, when changes occur, it must be possible to change one’s method, or anyway think about it.

Fig. 2: The Crystal methods

Underlying the Crystal methods are the principles of Agile Development: incremental delivery, measuring progress in working software, automated regression tests, and direct involvement of the customer. Testers, and particularly those that are involved in Quality Assurance, are reputed for always blaming fellow team members for their lack of discipline, when something goes wrong. According to them, they should have worked more in conformity with the process: it’s their own fault! Crystal holds a mirror up to their face: the process must be tailor-made to the project, and not the other way around.

Anyone involved in a project that develops software the agile way, will have to realize that the values of this approach differ quite a bit from the traditional approach. Being trained in a traditional waterfall approach, testers will have to change their mindset: they should be more focused towards customer satisfaction, working software instead of comprehensive documentation, giving and receiving lots of feedback from and to the project, and - most of all - lots of direct communication with the customer and the developers. This new approach can also deliver value to our traditional projects, for what would be the consequence of applying the Agile Manifesto to them?
Anyway, instead of presenting themselves as 'quality police' testers should take a much more constructive attitude, characterized by starting a dialogue with both developers and client, and two magic words: co-operation and flexibility.

[1] The Standish Group International, The CHAOS Report (1995)
[2] Agile methods include:
- Extreme Programming www.xprogramming.com
- Crystal www.crystalmethodologies.org
- SCRUM www.controlchaos.com

- Lean Development www.poppendieck.com
- DSDM www.dsdm.org
- Adaptive Software Development www.adaptivesd.com
- Feature Driven Development www.featuredrivendevelopment.com

[3] Extreme Programming Explained: Embrace Change, Kent Beck, Addison-Wesley (1999)
[4] Testing Extreme Programming, Lisa Crispin, Tip House, Addison-Wesley (2002)
[5] Agile software development with Scrum, Ken Schwaber, Mike Beedle, Robert C. Martin, Prentice Hall (2001)
[6] Agile Software Development, Alistair Cockburn, Addison-Wesley (2001)