Are All Pigs Equal?
- or are some more equal than others?

by Stuart Reid

"ALL ANIMALS ARE EQUAL BUT SOME ANIMALS ARE MORE EQUAL THAN OTHERS."
George Orwell, Animal Farm, 1945.

Imagine you are offered the opportunity to take up one of two jobs that are similar in all respects, except one is working on a traditional project and the other is working on an agile project. Which one would you take?

One thing is practically certain - most developers would jump at the opportunity to work in an agile environment. But would (or should) a tester?

"How agile is this project?"

As a tester one of the first questions you would need to ask is "how agile is this project?" This begs a further question: "*Are* there levels of project agility?" An agile purist would point to the agile manifesto (see right) and the associated principles and argue that a true agile project would be aligned with all of these. The reality is that there are very few truly agile projects out there; in practice the label of 'agile' seems to be accepted as long as the project develops software incrementally with the delivery of each increment typically taking no longer than 4 weeks. Alignment with the other principles varies dramatically. So the question of 'how agile?' is legitimate, but how does level of project agility affect testers? To answer that question you need to understand that a fundamental difference between working on an agile and a traditional project is that an agile team is empowered to make decisions and everyone is jointly responsible for the output of the team.

Agile team bonding

Everyone on an agile team works together towards a single goal and the best agile teams are those where the management have been able to provide an environment that nurtures the feelings of empowerment, togetherness, joint responsibility and trust within the team. In my experience of talking to team members on successful agile project teams, it is these characteristics that they generally find most attractive. Ensuring this healthy team environment is rarely cited as the main objective of organizations adopting an agile approach, but it is often seen as a key attribute associated with successful agile projects.

Manifesto for Agile 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.

Kent Beck	*James Grenning*	*Robert C. Martin*
Mike Beedle	*Jim Highsmith*	*Steve Mellor*
Arie van Bennekum	*Andrew Hunt*	*Ken Schwaber*
Alistair Cockburn	*Ron Jeffries*	*Jeff Sutherland*
Ward Cunningham	*Jon Kern*	*Dave Thomas*
Martin Fowler	*Brian Marick*	

Testing outside the agile team

An integral part of building a cohesive agile team is the shared responsibility for the team's output. The delivery of useful, operational software on a regular and frequent basis is a goal of a pure agile project. If the team's output is going to be a fully-tested, usable piece of functioning software (as declared in the 'Principles behind the Agile Manifesto' – see http://agilemanifesto.org/principles.html) then testing must be an integral part of the team that produces it – but many supposedly agile projects do not deliver usable software at the end of each cycle. Instead, they deliver software that still needs to be tested in a realistic environment, has not been tested against non-functional requirements such as performance, and also still needs to be user acceptance tested.

Thus on many 'agile' projects we find that the necessary specialist testing is not performed within the agile development team, but instead done as a separate activity some time after the agile development team delivers their output. So the main reason a tester needs to ask the question "how agile is this project?" is to determine whether they are going to be embraced into the togetherness of the agile team. The alternative is they are given an outsider's role of checking the agile development team's outputs and feeding incident reports back to them while the development team are only really interested in concentrating on creating their next increment – much the same as with traditional development approaches. In this situation the tester would not even be considered a pig, let alone an equal pig, but

would more likely be considered a chicken (see right).

Multi-functional agile teams

Suppose it's good news and the project is more agile than most and delivering usable software at the end of each sprint (the majority of projects that label themselves as 'agile' do not deliver usable software at the end of each sprint). In this situation the testers will necessarily be an integral part of the agile team and thus pigs. So far, so good – you'll presumably get all the benefits of being fully bonded into a successful team. You now need to establish what your expected role will be. If we return to the purist view we may well find that there is an expectation that all agile team members are able to perform any activity needed in the team – the so-called 'multi-functional' team. So, one day you may be programming, another day testing, and a third helping subject matter experts write user stories. This is one of the least likely 'pure agile' approaches you may encounter, but worth checking on, in case this happens to be one of these rare projects. For many testers this is going to be a deal-breaker – because "if they could design and program (and get paid more for doing so) they probably wouldn't be testers, would they?"

Despite the ideal of a multi-functional team and the corresponding benefits it creates with planning, scheduling and reviewing, practically no agile teams achieve this 'nirvana'. It is worth noting, however, that those teams that aspire to this are often perceived to be good places to work in as they expect and support team members in their continued professional development to become more widely skilled. Imagine the pleasure of working in a team where the developers are trying to understand how they can improve their testing practices, and where testers are treated as equals as they have demonstrated their ability to add value in design and code reviews using the development skills they have acquired.

An agile development and test process?

So, you decide to take on the nominal role of test analyst within an agile team. What can you expect to be your responsibilities? Let's consider what testing typically gets done in an agile sprint. Testing needs to be aligned with development (even in agile) so one place to start would be considering what development activities take place. In fact, it would be useful if we could define a generic tailorable process for agile projects and then we could see both the development and test activities and how they inter-relate (see figure below).

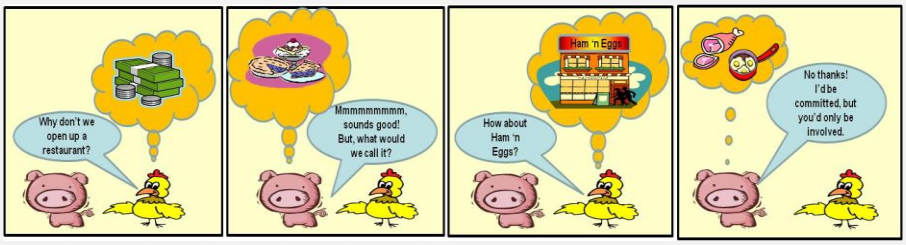
Those of you who read the agile manifesto earlier (or already know it) may well be wondering how I can even consider defining a process in an article on agile. *"Individuals and interactions over processes and tools"* is right at the start of the manifesto. It states at the end of the agile manifesto that the stuff on the right is considered of lower value, but I find that everyone knowing what process they are following is not just useful but absolutely crucial. For me the major difference with the process in agile projects is not that it is of less value, but rather that it is flexible and not a fixed process that rarely changes as in traditional projects. In an agile project the team should be empowered to evolve and (hopefully) improve their process, perhaps as often as after each increment.

Testing roles in agile teams

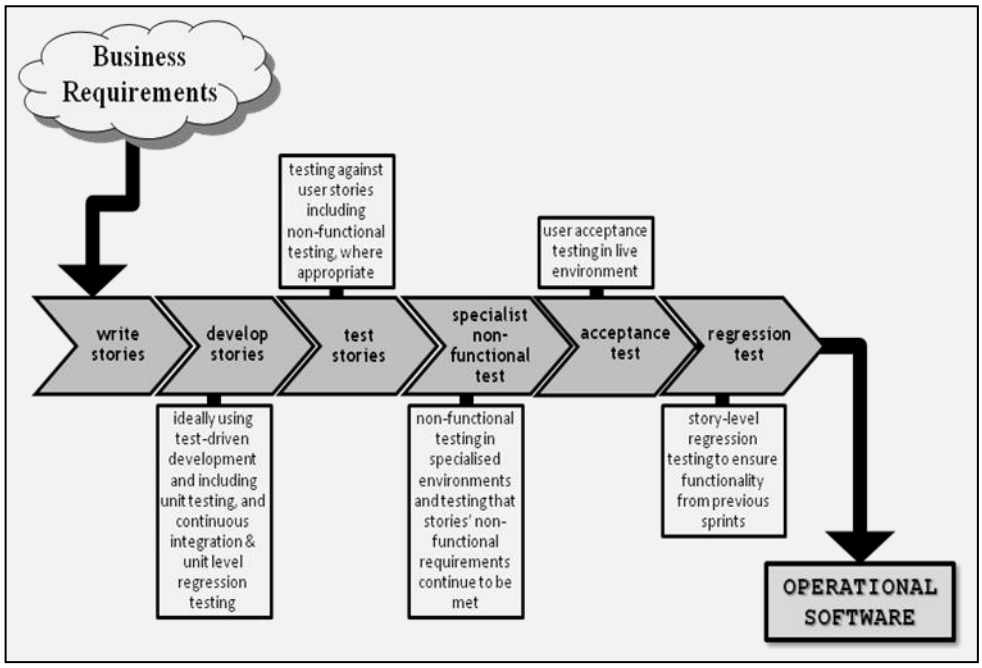
In the provided model of a generic agile development and test process, testing is explicitly shown in a number of stages. Of

Pigs & Chickens

The various stakeholders working on and with agile projects are often referred to as pigs and chickens; you are either one or the other, and note that neither term is supposed to be offensive. This classification is based on the joke in the cartoon below. Those stakeholders who spend all their time on the project are known as pigs (e.g. the sprint team members), while the remaining stakeholders (e.g. domain experts, who are occasionally approached for advice) are known as chickens. If you hold a daily scrum meeting then the pigs are expected to have their say, while the chickens are expected to listen.



The role of the product owner or customer representative can be that of either a pig or chicken dependent on whether they devote all or just some of their time to the project. Similarly, testers can be either pigs or chickens dependent on whether they are an integral part of the sprint team or used as a separate testing service after the sprint has finished.



course, not all stages will be required in all sprints and not all the testing shown in this process would be the responsibility of the tester. For instance, the testing during the 'develop stories' stage would normally be the responsibility of the developers, although the tester would be expected to provide advice in this area, as necessary. Similarly the acceptance test stage would ideally be executed by the end users, but the testers would normally play an advisory role to these users (e.g. how to write good acceptance tests) and take responsibility for setting up this stage of testing.

Testers would normally take prime responsibility for the 'story testing' (testing against user stories) and the story-level regression testing, which will involve running a regression test suite that is built up from the story tests of previous sprints. While much of the non-functional testing would be performed as part of the story testing, some may also be performed during unit test (e.g. memory management). However, occasionally specialist test skills or environmental constraints may mean that a separate stage is required.

It should also be noted that the order of processes shown in the development and test process model need not be followed exactly, as some projects, for instance, will perform the story-level regression testing more than once in a sprint, and may end the sprint with acceptance testing.

Different testing skills?

In many respects the testing processes followed in an agile sprint are much the same as those performed during a traditional project, as the testing still has to follow a fundamental test process. Differences typically arise in two areas: automation and the test basis (i.e. how the software under test is specified).

User stories

Unsurprisingly, agile projects prefer a lean approach to specification, which typically means that specifications (user stories) are far shorter than in (well-run) traditional projects (but remember bigger is not always better!). This economy of documentation is partially offset by the fact that the author of the user stories is normally close at hand and can be questioned directly when issues arise. In many agile projects testers are closely involved in the story writing, teaming up with business analysts to ensure that the stories are complete and testable – and ideally helping to define acceptance criteria up-front.

Isn't agile testing all exploratory?

A natural and common response by testers to poor specifications is to go for an exploratory testing approach, but this scenario should not occur on agile projects. Not because exploratory testing is inappropriate, but because poor specifications should be rare. This situation does not, however, just happen because we have labelled the project as 'agile' – the agile team need to ensure it happens by adopting a process that supports the creation of user stories that include the minimum amount of information needed to both develop and test the requirement described by the story. One way of doing this is for a tester to be directly involved when the stories are created, so guaranteeing that the stories take into account and support the tester's perspective. Another way is for the story template to require authors to explicitly include acceptance criteria with the story and for the team to never accept stories into a sprint that are not complete in this respect.

Unless the agile project is dysfunctional and adequate user stories are unavailable then the testing in an agile project is not just exploratory (see 'agile testing' below). You need to have the full range of test techniques available to you, so that you can select the right one for the situation. Agile projects produce software for a wide variety of applications, some of which need to meet regulatory requirements and some of which may be safety-critical, meaning we can't just rely on a single approach with no repeatability. Exploratory testing may often be an appropriate choice, but rarely will you use exploratory alone; I would normally expect it to be used to complement more systematic techniques. And, of course, where we create automated tests (discussed on next page) these tests are necessarily all scripted (and so not exploratory) encouraging us to use an exploratory approach to provide balance.

Agile Testing

The misunderstanding that all testing on an agile project is exploratory is perpetuated by misuse of the term 'agile testing' as a synonym for 'exploratory testing'.

AGILE TESTING ≠ EXPLORATORY TESTING

The term 'exploratory testing' was coined in 1983 by Cem Caner, and the following description is provided by James Bach, a leading advocate:

Exploratory testing is simultaneous learning, test design, and test execution.

In other words, exploratory testing is any testing where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.

If, at one extreme, fully scripted testing is performed where all tests are designed up front and test execution is simply running these pre-designed tests then exploratory testing provides the opposing view of this. In practice exploratory testing does not have to be completely spontaneous and 'partially planned' approaches such as session-based (exploratory) testing have been found to be very successful.

Exploratory testing as an approach can be considered to align closely with the *spirit* of the agile manifesto, but care should be taken to not equate exploratory testing with the testing performed on agile projects. It is widely accepted that the most effective test strategies (and this applies to both agile and non-agile projects) include both systematic scripted techniques and exploratory testing. In fact, given the reliance of agile projects on test automation (which inevitably uses scripted tests), it becomes obvious that agile projects cannot use a test strategy based solely on exploratory testing.

Thus, given that most people associate the term 'agile testing' with the testing performed on agile projects, it becomes clear that agile testing cannot be considered another term for exploratory testing. Conversely, exploratory testing can be considered to be an agile approach to testing.

Test Automation Skills

Although automation is common in traditional projects, it is an absolute necessity in agile projects (despite it being on the 'of less value' right-hand side of the manifesto). Test-driven development, automated builds and continuous regression testing all rely upon automation and won't work without it, while story-level regression testing and acceptance testing are also automated on many agile projects.

Do not worry, however, if you are not a tools expert. On many agile projects the test analyst does not take responsibility for supporting test automation as this is often considered to be the responsibility of either a specialist tools developer or a 'part-time' job for one of the developers. So not being able to program and not being a test automation specialist does not disqualify testers from working on most agile projects.

Lean development and testing

It appears as though developers get a definite benefit from the use of the lean approach advocated on agile projects as their requirement to document is substantially reduced – and who likes creating documentation? You might be asking yourself if there's a similar benefit to be gained by the testers – and there is.

A typical attribute of a successful agile project is a co-located team where each team member can speak to another without having to move from the team area (and ideally their desk). This introduces the possibility that when testers identify a potential issue with the software they simply go to talk to the relevant developer about the issue. This works best when testing is performed as soon after the developer releases the software to test as possible. If the issue can be immediately resolved by the developer then it can be argued that there is then no need for the issue to be recorded in the project's incident management system. Of course, where the issue cannot be resolved within the current sprint then it needs to be documented. The counter argument for recording every issue found by testing often cites the situation where a developer may just nod to the tester but then subsequently ignore or forget the issue. This argument, however, is flawed as the software will not get signed off as tested until it passes the tests, which it currently has not and will not do until the issue is resolved and the software retested.

No incident reports?

A traditionalist may well find this a difficult concept as previously they will have been forced to diligently document all issues and may well argue that without metrics no improvement can take place. However, in an agile project a retrospective is held at the end of each sprint to agree how improvements can be made and most suggestions will be supported by argument based on what happened in the last few weeks rather than an analysis of incident reports collected over a prolonged period. Perhaps the most persuasive argument I have heard for recording all incidents was made by two testers on a successful agile project whose boss was a traditionalist. His major input to their annual appraisal was the number of issues they had raised during the last year! In contrast, I find the most persuasive argument for not collecting fault metrics is to ask those who advocate their collection when they last found time to analyse them and act on the results.

The tester benefits on two fronts from not having to document all issues they discover. First, they don't lose the time that is spent on writing up each bug report. Second, there is less chance that their bug reports will cause an 'over the wall' division in the team between developers, who see themselves as being positive and moving towards the goal of delivering usable software, and testers, who are seen as simply trying to slow this process down. On an agile team all team members are supposed to be working together towards a single goal and, if done with sensitivity, verbally communicated issues are easier to accept than those received from an anonymous incident management system.

So are all pigs equal?

George Orwell's anti-Stalinist novel, Animal Farm, begins with the animals declaring their equality, but later the pigs form an elite and move to "all animals are equal but some animals are more equal than others". In an agile project we have already seen that the sprint team (all who are fully involved) are referred to as pigs, but are all members of the sprint team considered equal? It certainly appears as though the developers get major benefits from working on agile projects and most of the drive towards agile comes from developers. But, do the testers in an agile project team also gain from going agile?

The adoption of a lean approach in agile means that the amount of documentation is kept to a minimum, which is perceived as a benefit by all those that have to generate it. A concern of testers is that this is taken too far and inadequate documentation is available to support testing (and future changes), but in a well-run agile project practices should evolve to ensure a happy medium is achieved. Of course, the extension of this lean philosophy to incident management means that testers gain a similar benefit.

Test-driven development (TDD) is an advantage to both the developers and the project as a whole. Once they have grasped how it works nearly all developers embrace it as the only way they want to write software. Lead developers also like it as even their less capable programmers seem to produce reasonable quality code when using it. TDD generally results in higher quality code and creates automated tests as a by-product; these can then be used for automated unit level regression testing at practically no extra effort. This practice also raises developers' awareness of testing, which can only be of benefit to the testers.

As mentioned earlier, the team spirit and trust in a well-run agile team is considered a major benefit by those working in it; this applies equally to the testers as long as they are embedded as part of the sprint team. In this situation, the tester shares with the rest of the team in the responsibility for the team's outputs. In a successful team this means they also share in the reward of knowing they are producing something useful to the users, and, importantly, they get this (hopefully) positive feedback on a frequent and regular basis.

The best deal

In many respects I believe the testers get the best deal of all those working in an agile team, although this is partly based on comparing their agile situation to that on traditional projects. In agile they typically take part in a wider variety of tasks, even

over the short duration of a typical sprint. They get to interact closely with the developers and BAs and can even get direct access to the users on a regular basis. While working on story writing they can expand their analysis skills, while the emphasis on automation in agile allows them to improve their skills in test tools and scripting if that is an area they are interested in. Many agile teams use a pair-programming approach, which provides the perfect opportunity for testers to pick up and demonstrate programming skills, potentially allowing them to become true multi-functional agile team members.

If you consider there to be a career progression within an agile team then the most obvious move is from being a team member to scrum master. There is no reason that a tester could not make this move as easily as any other agile team member given the necessary experience, and some of the most successful scrum masters I have seen have graduated to that position by way of testing.

Conclusion

So, if you are offered the choice of testing on a traditional or an agile project, which way should you go? That depends. If you've got this far through the article it is safe to assume you are not a software tester who just does the minimum they need to do each day and who has no interest in making their jobs and lives more interesting (if you read articles on software testing that probably puts you in the minority of people in the software testing profession). So, my guess is you'd probably be better off going for the agile option.

But, it is not just a question of whether you are ready to work on agile projects. There is also the question of whether the project is agile *enough*. At the end of George Orwell's Animal Farm the pigs have reduced the commandments to the single "ALL ANIMALS ARE EQUAL BUT SOME ANIMALS ARE MORE EQUAL THAN OTHERS." There are now many projects that label themselves as 'agile', but some agile projects are more agile than others. If I were joining an agile project as a tester I would need to know that the testers are treated as pigs and that all pigs are treated as equals.

If you want further help on making your choice then fill in the completely unscientific Tester's Agile Checklist and see which way it takes you.

Tester's Agile Checklist

Answer each question 'yes' or 'no' then check your score using the grid below.		Y/N
1	Would you be offended by being classified as a pig?	
2	Do you consider exploratory testing to be the equivalent of hacking?	
3	Are the testers on the project considered to be part of the sprint team?	
4	Do you find contentment in writing fully documented incident reports?	
5	Is the output of each sprint immediately usable by the users?	
6	Are you happiest when working alone?	
7	Are you too shy to question decisions made by developers and business analysts?	
8	Do you find that regression testing is a waste of time?	
9	Are all projects in this part of the organization agile?	
10	Does the project have both a scrum master and a project manager?	
11	Are the developers using test-driven development?	
12	Do you find yourself panicking when faced with short deadlines?	
13	Are continuous integration and automated regression testing implemented on the project?	
14	Is exploratory testing all you want to do?	
15	Do you consider test automation to be someone else's problem?	
16	Can you see yourself sitting at a terminal and working on code with a developer?	
17	Does the thought of a lack of detailed specifications make you uneasy?	
18	Do you stick with your plans no matter what?	
19	Are you willing to take joint responsibility with the rest of the sprint team for the deliverables?	
20	Are you happiest following a fixed set of procedures rather than looking for a better way?	

Award yourself one mark for each of your answers matching those shown below.

1	N	6	N	11	Y	16	Y
2	N	7	N	12	N	17	N
3	Y	8	N	13	Y	18	N
4	N	9	Y	14	N	19	Y
5	Y	10	N	15	N	20	N

Scores of 15 or more suggest you should accept the challenge of working on an agile project!