5th International Software Testing conference

February 21- February 22, 2005
Hyderabad, India

Agile Development/Testing – Is history repeating itself?

A Technical Paper

By
Shrini Kulkarni
Microsoft India R&D pvt Ltd
Hyderabad
India

Disclaimer:

The views presented in this paper purely, are of the author and in no way relate to or represent those of Microsoft Corporation. There are different teams in Microsoft currently that are trying out agile methodologies and there are different opinions emerging.

The ideas expressed in this paper are a combination of my own ideas and those expressed in public forums by some of the leading practitioners of the Agile processes. I have quoted directly or indirectly from Jim Hightower, Alistair Cockburn, Ken Schwaber, Kent Beck, Martin Fowler and many others, and relied on sources at www.c2.com, www.agilealliance.com, While I stand behind the concepts I've described, and I use them in my daily work, I could not succeed without these amazing thinkers and practitioners who freely give their time and expertise to improving the practice of software development. I wish to convey my gratitude to my colleague Nick Malick who gave lots of insights for this paper.

Introduction

In good old days we had project teams where everybody played every possible role. A developer performed system testing, a tester developed a core module, and a support analyst performed some development/testing. So it was the team of all-rounders. As projects started

failing due to - late delivery, cost overruns, and cancelled projects, it led to more clearly defined processes of project management, development, test, and support.

Then there were customer surprises, production bugs causing “Quality” to take center stage and software testing emerged as separate discipline. So roles like testers, Test leads, and Test managers evolved and we have organizations focusing on independent software testing services as core business. But the woes of customer though reduced but left him asking for more sophistication in delivery and agility.

The phrase “Agile“ was coined in February 2001. A group of proponents of what were then called "lightweight methods" gathered to find out what they had in common. The term lightweight" was not considered very flattering, so "agile" was chosen instead. The group also produced "The Manifesto for Agile Software Development" (www.agilemanifesto.org).

The values, principles and practice of Agile are best described as in agile manifesto –

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

It is important to note that agilists have no problem with the items on the right. It is just that given a choice, they will prefer to go with the items on the left.

Agile methodology tries to address following key customer pain points.

- Responsive to changes business requirements – Agility
- Quick adaptability
- Quality – with continuous customer involvement and feedback, customer gets close to what he/she paid for.

This required project teams focus on delivering customer demonstratable smaller chunks in small iterative cycles, responsiveness and agility to change in requirements. Hence agile team started surfacing –where in close knit project teams worked based on following three principles

- Communication –Avoid busy work – focus on optimum documentation and communication
- Simplicity in process and management – iterative development and daily team meeting
- Agility to customer requests and changes – small iterative cycle with each cycle producing something that delivers some immediate business value.

Served in various flavors like Extreme programming (XP), Scrum, Crystal, ASD (adaptive software development) DSDM (Dynamic system development Method) etc, it has really caught the eyes of purists, reformers and everyone alike. Common values and principles can be found at agile manifesto

The practice of agile as it is known today has emerged as light weight process[2] is fast generating interest and acceptance as sustainable SDLC model and practice.

In Quest of new SDLC models –

Most of the SDLC models that we use today suffer from following problems:

- Requirements are not clear – not clear to start with
- Customer can give feedback only by seeing some working functionality
- Requirements change too often – the problem is that Traditional models expect that requirement do not change.
- Decrease customer satisfaction due to budget overruns delayed deliveries etc.
- Problems with not getting everything upfront – big solid design to begin with.
- As per some estimates, project teams build about 30% of the functionality that is not needed. That means that a large portion of the time spent developing every application is wasted on developing (and testing, and debugging, and deploying) functionality that no one uses. This happens due to lack of customer involvement or lack of getting early and continuous feedback.

So there is quest to invent new models or methodologies that produce software in faster in cost effective manner, satisfying the customer in quality and time to deliver.

Features of Agile development

- People
  - Highly people oriented – expects high degree of discipline by people.

- Process
  - Simplified communication and project management – daily meetings.
  - Customer involvement at every stage of the project – seek continuous feedback.
  - Continuous integration – keep droping some working code to test as often as possible.
  - Start coding ASAP – Do not expect big and robust upfront Design
  - Test-Driven-Development, the programmer first devises a simple example of what some chunk of code should do and implements it as a test that the code either passes or fails. Since the code doesn't exist yet, it fails. He next writes a tiny bit of code that makes the test pass. There is now one good example of what the code does, and code that does nothing but match that one example. He then picks another example, probably slightly more complicated, and goes through the process again, slightly expanding the code. The process continues until there are no more examples.
  - Test as early as possible
  - Project proceeds in iterations each consisting of a week to a month at the end of which the team delivers “nearly shippable product” containing promised features.
  - Project team has agility – does not worry in N^{th} cycle about what is coming in $(N+5)^{th}$ Cycle – “let us cross the bridge when we approach”.

- Communication
  - Agile programmers rely a great deal on constant communication.
  - Teams typically work in bullpens rather than offices or cubicles so that there are no barriers to asking questions or sharing information. Most have daily meetings intended to tell each other what they did yesterday, what they plan to do today, and what help they need.
  - Programmers are not guided by set of documents but by communications within team and with the customer.

Flavors of Agile practice[1]

Today's agile projects are run in any one of the following flavors of agile with all following basic principles of agile while making subtle modifications to suit a given context.

Extreme programming XP

Of all the agile methodologies, this is the one that has got the most attention. XP begins with four values: Communication, Feedback, Simplicity, and Courage. It then builds up to a dozen practices which XP projects should follow. Many of these practices are old, tried and tested techniques, yet often forgotten by many, including most planned processes. As well as resurrecting these techniques, XP weaves them into a synergistic whole where each one is reinforced by the others. On this platform XP builds an evolutionary design process that relies on refactoring a simple base system with every iteration. All design is centered around the current iteration with no design done for anticipated future needs. The result is a design process that is disciplined, yet startling, combining discipline with adaptivity in a way that arguably makes it the most well developed of all the adaptive methodologies. Few practices[2] of XP are: Planning Game, small releases, simple design, pair programming, test driven development, Collective code ownership.

Scrum[3]

Scrum is an agile process for developing software. With Scrum, projects progress via a series of month-long iterations called sprints. Scrum is ideally suited for projects with rapidly changing or highly emergent requirements. The work to be done on a Scrum project is listed in the Product Backlog, which is a list of all desired changes to the product. At the start of each Sprint a Sprint(or an iteration) Planning Meeting is held during which the Product Owner prioritizes the Product Backlog and the Scrum Team selects the tasks they can complete during the coming Sprint. These tasks are then moved from the Product Backlog to the Sprint Backlog. During the Sprint the team stays on track by holding brief daily meetings. At the end of each Sprint the team demonstrates the completed functionality at a Sprint Review Meeting.

However management does not disengage during the sprint. Every day the team holds a short (fifteen minute) meeting, called a scrum, where the team runs through what it will do in the next day. In particular they surface to the management blocks: impediments to progress that are getting in the way that management needs to resolve. They also report on what's been done so management gets a daily update of where the project is. Scrum literature focuses mainly on the iterative planning and tracking process. It's very close to the other agile's in many respects and works well with the coding practices from XP.

DSDM (Dynamic System Development Method)

DSDM started in Britain in 1994 as a consortium of UK companies who wanted to build on the RAD (Rapid Application development) and iterative development. Starting with 17 founders it now boasts over a thousand members and has grown outside its British roots. Being developed by a consortium, it has a different flavor to many of the other agile methods. It has a full time organization supporting it with manuals, training courses, accreditation programs, and the like

Using the method begins with a feasibility and a business study. The feasibility study considers whether DSDM is appropriate to the project at hand. The business study is a short series of workshops to understand the business area where the development takes place. It also comes up with outline system architectures and project plan.

The rest of the process forms three interwoven cycles - the functional model cycle produces analysis documentation and prototypes, the design and build cycle engineers the system for operational use, and the implementation cycle handles the deployment to operational use. DSDM has underlying principles that include active user interaction, frequent deliveries, empowered teams, testing throughout the cycle. Like other agile methods they use short time-boxed cycles of between two and six weeks. There's an emphasis on high quality and adaptivity towards changing requirements.

Crystal[4]

Crystal is a family of human-powered and adaptive, ultra-light, "shrink-to-fit" software development methodologies. "Human-powered" means that the focus is on achieving project success through enhancing the work of the people involved (other methodologies might be process-centric, or architecture-centric, or tool-centric, but Crystal is people-centric). "Ultra-light" means that for whatever the project size and priorities, a Crystal-family methodology for the project will work to reduce the paperwork, overhead and bureaucracy to the least that is practical for the parameters of that project. "Shrink-to-fit" means that you start with something possibly small enough, and work to make it smaller and better fitting. Crystal is non-jealous, meaning that a Crystal methodology permits substitution of similar elements from other methodologies.

The Crystals share a human orientation with XP, but this people-centeredness is done in a different way. The Crystal also puts a lot of weight in end of iteration reviews, thus encouraging the process to be self-improving. This method believes that the iterative development is there to find problems early, and then to enable people to correct them. This places more emphasis on people monitoring their process and tuning it as they develop.

Challenges of Agile[5]

Adopting agile methods for any projects are typically associated with “skeptism”, resistance to change and most importantly getting a buy-in from sponsors and management. Here are few challenges that I have observed and faced. Views expressed against each of this are - a compilation of responses or voices of agile community at large.

How does Agile/XP handles design changes that come along? If an agile project is considered to be equivalent to building house, how would one add basement to a house that does not have one, after the house is built?
Kent Beck (Father/creator of XP) states with the assumption that design of software is inherently different than physical construction design, in that the cost of changing the design of a well-architected system does NOT increase over time. In other words, in software, we have no gravity. Therefore, adding a basement is no more difficult, for a well architected application, than adding a new roof. The design changes are incorporated as and when they come. It might cause us to have to drop a story or whatever we need to do to make time for the design change.

Is Agile "Anti process" oriented? It is often said that Agile does not follow any process.
When an agile project is not run on agile principles, it tends to become ad-hoc. When people from predominantly traditional models try agile, there is a strong urge to follow process and do the things as dictated by these models yet there is a tendency to call the project as agile. In such cases of ambiguity or people deciding to do a hybrid model, project tends to become Ad-hoc. Agile, fundamentally requires a high degree of individual discipline. Some times people tend to

become ad-hoc. They may panic in the face of difficulty and go back to old, bad habits even though they don't work.

What happens to metrics like bugs slipped to UAT by test etc?
For a defect to "slip to UAT," the requirements were either misunderstood or were not fully tested. While the second happens, the first is by far more common. Rather than saying "let's fix the requirements, and measure the drop in this number," Agile processes say "let's increase communication, and measure customer satisfaction and the total number of useful features delivered in a product."

What are the metrics that Agile project teams measuring? How about Effort and schedule variance?

Measuring anything is always tricky because it is open to interpretation. From the stand point of delivering the business value, metrics that we measure need to be built around “delivered features”. To count, a feature must be finished, meaning it must have its GUI, its database tables, its installation procedures, and its tests. And all its code must be refactored to fit in with the pre-existing code.

Here is what I heard about metrics measured by agile teams.

- The count of running, finished, tested features
- The time from a business decision to delivering the result
- Code Complexity
- Total lines of code
- Code coverage by unit and acceptance tests
- Velocity - The number of features finished per week.

Metrics like effort v/s schedule variance are not measured as this is pretty hard to track since re-estimations of stories will affect the schedule. At an iteration level, one can measure the completed stories with what was on the plate for the iteration. Sometimes it is possible to achieve more stories than in the initial iteration plan.

How are bugs managed in agile projects?

What is Agile's recommendation of Defect management? Is every bug logged and tracked as such - if yes this contradicts agile principle of simple communication -Emphasis on people than process.

By and large there are two approaches for handling bugs in agile methodologies. There is some degree of disagreement between using a full fledged defect tracking system and using simple backlog tracking system. Some Argue that there is need to have both as a defect tracking system is quality focused – meaning it helps to track the defects, causes, trends and remedial measures where as backlog sheet is “feature focused” as it keeps track of features and bugs in them.

TDD, pair programming, and continuous integration have been found to show absurdly low defect rates. Just as you rarely find yourself using a debugger, or hunting bugs for a long time, you also rarely find yourself delivering a bug. In case of legacy system that still has a lot of bug a defect tracking system would be of some use.

Bugs are found in the testing of the story. A bug may become a story if it's too big to fix during the same iteration, or if it's not really a bug but an unstated requirement/feature.

The handling of bug reports is often different in Agile projects (write them on a card; plan them like any other potential change; etc.) It might be useful to tell a story of a bug as it goes from being found to being fixed. It'd also be useful if the bug were inherently interesting, a representative of a class of bugs useful to know about.

Although some of the "QA" testing is done in the dev environment and reported informally. Most of the functional testing is done on deployed builds and reported using a bug tracking tool of one type or another.

What about test cases?

An agile project team member gets involved with following categories of tests.

- Unit tests - Developer facing tests : are written in executable format
- Acceptance tests – Customer facing or business facing tests – written by testers and form end to end business scenarios.

Test cases form the detailed requirements for a user story and use cases. The goal is always to write test cases in EXECUTABLE format, meaning in a format usable by a test tool. If possible avoid writing test cases twice.

For a typical medium-sized business application, the right unit test suite would be several thousand test cases taking a few minutes to execute. Probably written in Java or C#. The right system (acceptance) test suite for the same business application would be maybe a few dozen end-to-end scenarios. Probably written in a higher-level language.

The right integration test suite would be somewhere in between, if at all necessary.

When and how UAT (User acceptance testing) happens

I know most agile projects don't do a single sitting of UAT because the customer is involved right through the during the development cycle. In other words, UAT happens throughout the project. However, if we have a very large system with many components we might need have a “two” week iteration at the end devoted to system testing. We sometimes have customers come in right after that and sit with the testers to review results, do some of their own testing, etc.

From the measurement of quality standpoint, customer facing testing done by either customer or any team that is separated from project team, the concept of traditional UAT be used. The bugs found during such testing are considered as test escapes and need to be investigated upon to see how to prevent them.

What happens to Test planning?

A Test plan is a design document of the test team. The test planning is an important task from the high level view: what areas of features will be tested and how will resources be spent on creating these tests. Unit tests should not be included in a test plan. However, the test plan should still make it clear which functions are expected to be of high quality because unit tests were delivered, and which functions will be tested by providing data, models and automation.

Working on traditional waterfall projects, typically it has been observed that nobody reads the test plans anyway, and certainly nobody maintained them as changes occurred. High level test cases/scenarios are written before or at the start of each iteration, then detailed test cases are written as coding begins. Product owners and testers write the test cases, programmers automate them, it's a collaborative effort.

When it is a client requirement to produce a test plan as a deliverable, spend minimum time on this and mainly included the release plan, and refer the reader to the location of the actual test cases.

What about Test Automation? Are current "state of the art" automation tools in-line with agile principles?

Test automation/UI automation or automated regression tests as performed by current industry standard – record and play back tools do not fall in line with Agile.

The trouble is these tools neither guide development (can't be done first) nor do they increase agility (the tests are more brittle than the code).You can't record a script for a product that doesn't exist yet!

So test automation in Agile terminologies – boils down to writing executable tests (both unit and acceptance tests) Here you can write a test code for a piece of functionality that does not exist yet and hence support Agility.

Traditional Test automation (read UI in most of the cases) assumes that requirements are nailed down in advance, and that developers get the UI finished and locked down early. If the UI changes, some capture/replay literature seem to say that this is a failing of the development process, not that the tests are fragile.

Agile testing – role of tester

- Be a customer's representative with dev team and dev team's representative with customer.
- Help customer to think through the stated requirements, use cases and user stories.
- Unfold user stories, ask intelligent questions, make sure that user stories are detailed and exhaustive enough and Write acceptance tests.
- Tester is no longer a sole quality advocate or final filter on quality – but shares the responsibility with the whole team.
- Help development team in executing unit tests
- Work with developers to clarify the requirements where gaps exist caused by boundary conditions and error conditions that were poorly understood or described in vague terms.
- Offer feedback on the use cases to insure that the minimum success criteria is meaningful and that all alternative paths are understood and meaningful
- Help development team with develop features with minimum documentation that is available and keep clarifying as iteration progresses.

- Pair with the programmer to produce better software.

- Support in customer facing testing – get feedback, engage development team and facilitate constant communication.

A Typical Agile project – my experiences

This was project that we needed to build where in customer had a very vague idea of what they wanted to build and were having problems in visualizing the whole thing. We quickly cobbled a team of experienced developers, tester and project manager and started off what is known as exploration into world of agile. Except one in the team for every body it is a new experience of building something like this. We all were very committed and excited to make this project a success.

We started off with some high-level design document that outlined core components and suggested architecture. Both dev and test started banging off this stuff trying figure out what this all about. We also had a set of use cases and user scenarios extracted from the users that depict how the system could be used. Most of the project management and communication happened using a scrum sheet that served as a single repository of requirements, development tasks and dates of the code deliveries. The whole V1.0 project was broken down into 5-6 iteration of 2 weeks each with each iteration providing some set of features that could be demonstrated to the user. We had daily meetings that allowed the whole team to sync up what that will done that day and at the end of the to see whether we could achieve what we thought we could.

Scrum sheet contained a big laundry list of features/bug fixes and issues in the form of "Back log sheet called "user Stories". Each iteration started with a planning game or ceremony where whole team gathers in a room with a prioritized list of stories that are lined up for that iteration. These were pulled from backlog sheet based on priorities assigned to individual user stories. Each story was broken down, unfolded with more details and there come the dev and test task that make up iteration. Each team member that comes out with an estimate for completion of the task allocated to them.

From project management's perspective – it is SCRUM sheet all the way. Everybody at the end of the day updated the tasks allocated to them by entering the time in hrs that task has left to be done and completed task would have "0" as on that day's time column. Depending upon the progress made for a task, PM would then predict whether or not that piece of functionality would make it iteration demo to customer.

There were customer demos at end of every iteration and customer was really excited about the way we were delivering the stuff. There were changes at every iteration. Whole application was built like a jig saw puzzle where we went on discovering different pieces of the app as we started seeing the ready made piece. There was continuous communication in the team; every team member knew everything about what is happening in the middle. Everybody contributed their piece at every stage. There was little "complaining" from dev or test about changing requirements as they were ready for it any stage of the project. So were the program managers – did not insist on many mandatory sign off's of the documents like, test plans, test cases. There was great sense of faith in the capability of the team to deliver.

At every stage – the team demonstrated agile principles –

- Test driven development

- Simplified project management and communication.
- Agility in response to changes.

Why not to go for Agile?

- Requires a paradigm shift. As is the case with any change there is resistance to change and purists say agile is too ad-hoc. It tends to be more that way that when an agile team gets stuck between agile v/s traditional.

- In close knit teams where the functional barrier between Dev and test is fast thinning – Testers runs into risk of loosing objectivity

- Management blues/Budget issues – difficult to fund something that will be built over 5 iterations spread over 1-2 months without knowing which shape the end product takes.

- There is still skeptism over the way agile team handles complex design and architecture problems. It is difficult to convince management that Agile is the way to go and deliver such complex applications in iterations. Traditional model handles such cases by building robust and stable upfront design taking months of design and changes after design freeze are risky and difficult to implement. There needs to be a change in the mind set to accept that even these so called "Robust and stable designs" fall flat when it come to implementing something that developer did not think of in the first place. This is something agile can handle.

- Issues with continuous customer involvement – it is a bandwidth issue. As most of customer feedback comes from group of selected end users and these people get involved with feedback only as their part time work – they have their day's work to do.

Road Ahead

Having seen what agile is about, what are challenges, how agile teams work, let us look what is in stored for agilists?

Use Agile when -

- Requirements are uncertain and volatile.

- Start with teams that are smaller in size – Experiment and then try with bigger teams

- As far possible, try with the teams that are geographically located at the same location.

- Customer is easily accessible and likes the idea of agile dev.

Conclusion

Stable is the antithesis of agile. Agile makes it easier to change, because change IS a given. This is precisely the problem that agile model is trying to solve. Agilists say "Requirement changes – Welcome".

In agile I see the distinction between tester and developer disintegrates as testing is another story to be told. This future cross-role of developer-tester is difficult to articulate and develop and I wonder where this path will lead. In an agile organization, does one have to choose between the role of developer and the role of tester? It could be that one could play both roles as needed to deliver what customer needs.

Every one in an agile team is playing roles that help the team to achieve the common goal of delivering what customer wanted and at the same time not continuously complaining about “un clear requirements”, “Changing requirements”, longer design and analysis cycles etc”.

Is this change for Good? If yes? for whom? Are we going back to historical days where there was not dev test separation?

I would assert that history is not repeating. In the past, we used these practices in isolation, without methodology or reinforcement. When projects started to fail, we were told that it was our fault for not keeping an eye on quality, and then the solution was to increase project management. Now, we are simply realizing that the solution was not better than the problem, but that there is another way. Yes, **we returned to our roots**, but we then embarked on an entirely **different** path. That is different. That is Agile. **Welcome Agile.**

Glossary

User story

It is a smallest unit of work that can be estimated for development and testing and one that delivers a specific business value to the customer. The "story" is not a description of a feature in a program, but the underlying real world problem that the software is designed to solve. An important concept is that your project stakeholders write the user stories, not the developers. User stories are simple enough that people can learn to write them in a few minutes, so it makes sense that the domain experts (the stakeholders) write them. A user story carries a priority assigned by the customer and has estimates. User stories should only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement. When the time comes to implement the story developers will go to the customer and receive a detailed description of the requirements face to face.

Iteration

An agile project is delivered in iterations which are nothing but small project cycles ranging from two weeks to a month. Through the concept of iterations, project teams keep in pace with customer requirements by planning the project iteration by iteration. Each iteration is pretty independent and project team working on any given iteration is not aware of and does not bother to know what is lined up for next iteration. In true agile style, a team member in agile team treats every iteration as last one in the project so that team is ready to deliver few working code at the end of each iteration.

SCRUM sheet

It is a single repository used by an agile team. It contains the details of iterations, backlog sheet – list of user stories, project tasks. It acts as main source of information for project team – used both as repository of features and project management tool giving details iteration dates etc.

Test driven development

It is one of the core practices of XP, where a developer developing a piece of functionality, writes tests first before the code.

The steps:

- Write a test that specifies a tiny bit of functionality
- Ensure the test fails (you haven't built the functionality yet!)
- Write only the code necessary to make the test pass
- Refactor the code, ensuring that it has the simplest design possible for the functionality built to date

The rules:

- Test everything that can possibly break
- Tests come first
- All tests run at 100% all the time

Source: http://www.objectmentor.com/writeUps/TestDrivenDevelopment

Pair programming

Two programmers working side-by-side, collaborating on the same design, algorithm, code or test. One programmer, the driver, has control of the keyboard/mouse and actively implements the program. The other programmer, the observer, continuously observes the work of the driver to identify tactical (syntactic, spelling, etc.) defects and also thinks strategically about the direction of the work. On demand, the two programmers can brainstorm any challenging problem. Because the two programmers periodically switch roles, they work together as equals to develop software.

Source: http://www.pairprogramming.com/

SRUM Master

In the Scrum process, this is the person responsible for keeping the team following the process. This is somewhat similar to a project manager role. However, because Scrum asks the team to organize itself, this person may or may not be the perceived "leader" in a given Sprint.

The Scrum Master is the team member responsible to interface with the world outside the Scrum iteration, and to remove/resolve blockages that impede the team's work within the Scrum iteration.

Refactor

This is a coding activity where in a developer modifies or changes a piece of existing design or code that works by removing redundancy, eliminating unused functionality, rejuvenate obsolete designs. This helps to keep the code simple and tidy. The aim is to keep your code clean and concise so it is easier to understand, modify, and extend. It ensures that everything is expressed once and only once. In the end it takes less time to produce a system that is well groomed.

References

1. www.martinfowler.com/articles/**new**Methodology.html - "New methodology" by Martin Flower
2. Extreme programming by Ron **Jeffries** http://www.xprogramming.com/xpmag/whatisxp.htm
3. SCRUM: **http://www.mountaingoatsoftware.com/scrum/index.php**
4. Crystal: http://alistair.cockburn.us/crystal/philosophy.html
5. [Agile testing] – Yahoo group discussions http://groups.yahoo.com/group/agile-testing/