International Journal of Academic Research and Development
ISSN: 2455-4197
Impact Factor: RJIF 5.22
www.academicsjournal.com
Volume 3; Issue 1; January 2018; Page No. 221-226

Agile projects: Foundation to successful outcomes

Subash Thota
Data Architect for Data Diverse, Washington DC, USA

Abstract
This paper provides a guide to all the basic processes and tools that need to be in place for an Agile project or delivery stream to be successful. It can serve as a template for new teams or projects or as a reminder and checklist for teams that have already adopted Agile. This paper gives some unique insights into the sort of things that you might observe if you try to do Agile whilst cutting some corners!

Keywords: agile, scrum, project management, backlog, burndown, velocity, epic, user story, task, sub-task, iteration, product owner, sprint, stake holder, stand up, story points

Introduction
For teams new to Agile, it is not easy. To make it work training and coaching will be required. The project will require a different set of environments and tools. The teams will often require new skills and very different interactions. The way requirements are captured, prioritized, elaborated and then delivered are all very different. With all of these challenges, one would wonder why so many teams, departments, and organizations have made the Agile transition. The benefits must be very high to be worth the time, cost and energy required to transition successfully.

Project Summary
With any project, if you join it midway through its delivery, you spend the first few days getting up to speed on all its aspects. As an agile transformation coach, my focus was on the agile delivery process, namely how requirements were captured, described, implemented and accepted. The following is a list of my findings.

Project Timescales
The project team had just completed Sprint 9 and was scheduled to deliver three more Sprints to complete the project. The original schedule planned to complete the project after eight 2-week Sprints with one further Sprint for contingency, so it was already late. Sprint 8 is being used for defect clearing rather than delivering new functionality. It soon became obvious that the real number of Sprints left to complete the project was not clear.

Requirements / Backlog
Any agile project should have a Backlog of Stories to deliver and indeed this project had a backlog. However, the backlog this project had was not ideal. There are certain criteria; each Story should be aligned with

- Independent (as far as possible)
- Negotiable
- Valuable
- Estimable
- Sized appropriately (small)
- Testable

Before a Story can be executed in a Sprint, it should have Acceptance Criteria (agreed between the developer, tester, and Product Owner). This means all stakeholders know what the Story will deliver; there are clear boundaries to what will and will not be included. The Stories in the Backlog should then be estimated (preferably in Story Points rather than hours).
In my project, the evidence and outcomes were as follows:

- Stories were formed by dividing a more traditional requirements document into pieces. This meant they were not delivering coherent 'slices' of an end to end functionality. The outcome of this was Stories needed to be carefully grouped to deliver meaningful functionality. Stories could not be completed independently of one another. End to end Stories would have included the screens, but these were delivered later in the project.
- Stories could not be completely delivered in one Sprint – the evidence of this was that Stories that were already completed would be re-opened, sometimes repeatedly. When a new Sprint was planned, the Stories would be a mix of new ones (with time effort estimates) and existing ones (without estimates). The project team called these 'baseline stories' but the effect these had was a large part of the project work was not estimated so a reliable prediction could not be made of the end date.
- Stories did not have Acceptance Criteria – the first step in developing Stories was for an experienced BA on the project to create a Sprint 'design document' which both developers and testers (to create their test scripts) would require before work could start.

The project was being delivered as a fixed price piece of work. Agile, incremental delivery processes handle uncertainty very well - early in the delivery cycle you can measure actual delivery performance and compare it with the

estimated performance. This allows teams to adjust plans or teams to meet the original timescales. Agile works very well with fixed-price contracts (although you would try to fix the price once 2 or 3 Sprints had been completed and the team velocity was known). Using Agile, the process is

1. Create the Backlog of Stories; Agile promotes collaboration of contract negotiation but, for a fixed price, you would probably break down your backlog further to get a better idea of the size of all the Stories in the Backlog and create Acceptance Criteria for any Story where the scope might expand.
2. Use the team to estimate all the Stories using Story Points
3. Deliver 2 or 3 (2 weeks) Sprints to determine the team Velocity
4. Go through your Backlog using the Story Point estimates to work out which Stories will be delivered in each Sprint
5. Create a fixed price contract based on the Backlog broken down into Sprints

The contract will be agreed for a certain number of Sprints which should be enough to build all the Stories on the Backlog (down to a certain point), perhaps with a little contingency. The key thing is, any new work identified is created as new Stories at the bottom of the Backlog. The customer then decides the priority but, crucially, if they get moved up the Backlog, something else moves down. If that then means it is below the fixed price contract line, either the customer does not get that Story, or they can pay extra to have an additional sprint to deliver them.

Sizing the Backlog

For any project, if you are going to agree a fixed price you need to understand how 'big' the scope is and therefore how much time and effort it will take to deliver it.

Of course, it is always possible that the work proceeds through the Sprints more slowly than planned (because the overall project is bigger or more complex than originally thought or there are other constraints on the progress that cannot be overcome). Therefore, if a project were to follow an agile process, we would not normally agree a fixed price for it until 2 or 3 Sprints had been completed and we had a concrete indication of the size of the project and its likely delivery schedule.

In the case of this project, the Stories on the Backlog had person-hours estimates created by the BA. But these did not consider Stories being re-opened. The estimates were higher because Stories did not have Acceptance Criteria, it was open to interpretation whether they had been finished. The final user screens were built separately, and the customer had the view that they would only be signed off once all of them were finished. This, of course, although contested, mean a massive backloading of risk to the supplier in the project.

Story Acceptance Criteria

The customer for this project understood enough about agile to know that things that had already been delivered could be changed. At the start of the project, although a fixed price had been agreed this was not against a fully fleshed out set of requirements. The requirements were detailed as the project progressed.

In an agile project, it is critical that Stories are not played until all the 'Ready to Play' criteria are met. 'Ready to Play' includes a set of Acceptance Criteria have been defined and agreed between the Product Owner, the development, and test teams. In this way, as Stories are delivered (with agreed Acceptance Criteria), additional work discovered can be added to the Backlog, beyond the delivery complete line. This additional work could then

- Be prioritized over later Stories – so replace other work
- Be added to the Backlog as additional Sprints – so be charged for

In this project, because a proper Backlog had not been agreed and Stories did not have Acceptance Criteria this additional work became a 'grey area' of a dispute between the customer and us.

Agile Process Implemented

In addition to managing the project scope effectively, it is also important to implement an effective, disciplined Agile process. For reference 'the Team' consists of all the developers and testers delivering the project, the Scrum Master, and the Product Owner. The process includes

At the Planning meeting, the Team can challenge the Stories coming into the Sprint – either because they are not 'Ready to Play' or because there are too many (too much work to complete)

- During the Sprint, the Team is supported by the Scrum Master (who removes impediments) and the Product Owner (who answers questions related to the Stories)
- At the end of each Sprint, the Team gather to reflect and devise improvements to their work – continuous improvement is a key feature of Agile that helps to bond the Team to a common cause, empower them and maximize effectiveness
- Frequent Story Grooming sessions (with the whole team) make sure the Stories are 'Ready to Play' from the Team's perspective (not the managers'!)
- Daily Scrum meetings help the Team to uncover (and resolve) issues quickly and ensure they are all focused on getting as many Stories as possible finished (to the required quality standard) at the end of each Sprint
- Acceptance of Stories (by the Product Owner) takes place throughout the Sprint (not just on the last day) so that there is plenty of time for any minor corrections that might be required

Putting this discipline in place requires time, some training, good communication tools and commitment. In the case of this project, only very superficial aspects of this process were put in place. Understanding the impact of this on the project is harder to quantify. But the following observations were made

- Planning was often seen to produce a commitment the team could not meet. Imposing weekend and longer hours working did not seem to improve this situation and seemed to degrade quality
- There was no evidence of Retrospectives taking place and none of any improvement in productivity or code quality
- Although the 'functional design' was delivered soon after the start of each Sprint, without agreed Acceptance

Criteria, the project was plagued with differences of opinion over what the detailed requirements of each Story were and when they were finished.
- Daily Scrum meetings served as management status reports and issues tended to be glossed over until the end of the Sprint when they became obvious
- The Product Owner only saw the Stories after the Sprint had completed leaving no time for corrections only defects

Test Automation

In a true Agile process, development and testing take place in parallel. The reason for this is that you don't have time to do development and then testing sequentially. The only way to do this is using test automation.

When building an application iteratively, even one where it is mostly configuration rather than customization (coding), you still run the risk of breaking things you have previously built each time you add something new. This is where test automation comes into its own – together with an automated build process it means almost every change made can be checked to make sure everything still works, almost as soon as the change has been made.

Be careful how much you use some 'pseudo' automatic test tools. A tool which tests a Web page or application using record and playback is a compromise because the test cannot be instrumented and run for the first time until after the code has been delivered. If it interfaces directly with the page, it is likely that any change to the page – even small changes to positioning look and feel will stop the test from working meaning that it needs to be rerecorded.

On this project, almost all the testing was manual. So, towards the ends of the Sprint, a period needed to be set aside to execute this manual testing. This can be made to work for small, low-pressure projects. But as the application grows there is more testing to re-run and more time needs to be allowed at the end of each Sprint for testing. On this project, that time did not exist; in fact, the developers seemed to be squeezed just to complete coding in time. The outcome was when the release for Customer testing was cut; smoke regression testing revealed many defects. The pressure was then on the development team to not just deliver scope but also fix defects in the next Sprint putting additional pressure on them which led to even more corners being cut and even more defects.

Definition of Done

To compound the testing issues, only half the testing was performed within the Sprint - Business Acceptance Testing was scripted and executed by a separate team. The Stories were not reviewed by the Product Owner as they were completed - instead, the first look he got at them was in the demo that took place following the end of the Sprint. Business Acceptance Testing formed about 15 man-days of effort (again, all performed manually) making it very hard to bring into the Sprint. A 'Definition of Done' existed but was not adhered to (and made little sense when so much Story testing took place after the Sprint had finished). The visible outcome of all these compromises was again, lots of defects reported which needed to be fed back into the following Sprint for rectification.

Story Estimation

In a conventional Agile project, the team developing and testing the Stories provide the estimates of how long work will take using relative estimation in Story Points. On this project, estimates (in hours) were provided by a senior business analyst (the same person who created the functional design at the start of each Sprint). It is hard to say that this process meant the delivery team was not committed to delivering the Stories in the Sprint. But I did not see a Sprint where all the Stories planned for a Sprint were successfully delivered (i.e., all Stories completed with a low level of defects).

Team Engagement and Continuous Improvement

A key feature of Agile projects is a continuous improvement – measured many ways but including increased productivity, better quality, better flexibility or less rework. But this only works if the team doing the development work are actively engaged in a continuous improvement process. The development team for this project was managed by a manager so had little involvement with the team. There is little evidence to suggest that Sprint Retrospectives took place or actions were created coming out of them. The team carried out a process akin to a retrospective. But without those being involved, this had little impact on day to day delivery.

Tools

There are many tools available which help delivery teams to be effective as well as keep track of the work they are required to deliver. These become more important for Agile teams as things move more quickly and for distributed teams to ensure information is readily available and shared with all. There are tools like Jira available which allow Stories to be created, managed and tracked through to completion which is indispensable for agile projects.

On this project, there was no code configuration management tool in place. The combination of multiple developers working on the same codebase, rushing to complete development at the end of each Sprint and the lack of this tool caused many problems. These were evidenced in the multiple defects in releases, variations in functionality between test environments, functions 'disappearing' after they had been built and defects reappearing after they had been fixed.

I have already covered the lack of test automation, continuous integration or even code configuration management tools. In addition, Jira was used to track defects. Stories (which could have been held in Jira too) were in a spreadsheet. So, the extensions offered by Jira to provide an electronic Kanban or Sprint board were not taken advantage of. Quality Centre was used to store test scripts, so the testing (although manual) was well managed. This project demonstrated the importance of the correct tools to a project. It also showed that common tools across all parts of the team should be used.

Lessons Learned

Up to this point, I have talked about some of the things that can go wrong. So now I will examine (based on my experience) what things should be put in place to make projects successful. The details will vary from project to project, but these are good starting points. An experienced Agile Coach will be able to determine the details.

Even if, on the face of it, the project looks straightforward, simple even, always follow the basics – this project was the supposedly simple configuration of an industry standard platform. Some of the basics were not in place which meant, when it became more complex, the controls were not in place to manage the project. 'Experience Certainty' – we all know this strap-line – we must deliver on this every day. There were a number of aspects of this project where we seem to have forgotten this:-

We are a level 3 CMMI organization. That means any project, should meet those standards. On this project, I discovered that the project was not using a code version management tool. As an experienced CMMI assessor, I am fully aware this would break level 2 criteria. Feeble excuses (in this case, the customer was not willing to pay for it?) are exactly that. No matter how small, simple or temporary our projects are, no matter how hard we must compromise on the budget, all our projects should conform to our highest standards of project management and delivery.

The project was run as an agile project and some of the team members on the customer side had previous agile experience. Where they saw us doing things that did not follow agile practices, they just assumed 'We know best.' And this argument is perfectly reasonable. The responsibility for setting up a project for success sits fully with us, not our customers. Agile delivery is a very effective way of controlling and delivering projects. However, it is also highly disciplined and easy to get wrong.

One of the unique insights I had working on this project was to understand what it is agile coaches bring to projects. One of the unexpected messages was it is not necessarily experienced working on agile projects. The project had an experienced Scrum Master on the team and other team members who had worked on agile projects. But they were all less senior team members. They saw things being done in the early stages of the project that they did not feel were correct for an agile project but were unable to prevent this. In short, they did not have the authority to influence the project leadership to change things.

Agile coaches are experienced, senior members of the project team who don't just have knowledge of best practice, critically they also have the authority and gravitas to ensure this is put in place. For this project, a good example was the project Backlog; Rather than being a prioritized list of Stories that met the INVEST criteria it was formed by slicing up a conventional requirements list. One of the outcomes of this was that Stories that had been played in previous Sprints were re-opened to 'finish-off' pieces of work, sometimes Sprint after Sprint. So, estimates for Stories (in this case in hours, not Story Points – another failing) were meaningless because you did not know if the Story would need to be re-opened again in future Sprints.

Project Setup

This is the most critical part of the project – setup correctly, even if the project has early teething troubles, it will have the tools it needs to address them and improve. But these steps will be hard to setup and take longer if the team has not previously worked on agile projects. As my example project showed, it is very easy to slip into a situation where not all the criteria for success are put in place. If you are following an agile process, there are certain foundations you need to have right (such as test automation and continuous integration) which are critical to success. If you compromise on these at the beginning, the compromises will get worse as you go on. If you really can't put in place these foundations, think carefully about whether you should do waterfall or some other hybrid process.

Agile projects work best when the delivery team has the right processes in place, and they are empowered to use them effectively. You need a management structure in place that supports this. It is very easy to have a 'command and control' structure, especially with teams. There are a number of challenges agile projects bring

- All the team is involved (with estimates, planning, retrospectives, etc.). This means anyone in the team, could potentially be put up in front of the customer (either directly or via WebEx).
- Agile works best with smaller teams of experienced engineers rather than a large pool of juniors supervised and directed by seniors. Although some junior team members can be accommodated if they need to work from detailed designs or pseudo code created by more senior team members they will delay the team too much for effective delivery.
- Effective test automation requires people with both good testing skills and development skills. The skills required will very much depend on the technology being used to build the application and the related tools available to the team.

Everyone needs to understand the process, and with agile being new to many, that means training. A lot of the success of agile comes from changing people's behaviors. And that will require coaching. But the benefits in terms of productivity and quality will be more than repaid.

One of the reasons Agile projects are more effective is because things move faster and teams work more closely together. But this makes the tools supporting these teams even more critical than for traditional projects. There are plenty of inexpensive tools around (e.g., the Atlassian toolset) available or for teams using Microsoft technology, TFS is available as an extension to Developer Studio.

Test automation is almost mandatory for successful agile teams, and where tools are available to support this, they should always be adopted. The more mainstream the tool, the more likely you are to find engineers with experience of using them, the better supported they will be and the higher the likelihood that they will be enhanced in future. Try to avoid building your own bespoke tools. Not only does this mean you will have to support them, but you will also need to train any new users.

Processes

Scrum is the most popular agile process (and is at the core of bigger frameworks like SAFe). SCRUM is widely used, and there are many books, papers, tutorials and blogs available covering all aspects of SCRUM as well as related techniques like XP, TDD, BDD, etc. If you are adopting SCRUM for the first time, adopt the whole process – don't pick and choose the

bits you like or the bits you think will be easiest to implement. Although there are experienced, successful teams who have tailored their SCRUM process, this will have put in place a disciplined process in the beginning and then tailored it carefully as they learned what worked and did not work for them.

Customer Engagement

A cornerstone of any agile project has an empowered and effective Product Owner. This is a person from the customer who understands the requirements from a customer perspective and is empowered to make decisions about the detail of them. This is particularly challenging for teams where many people from the customer team need to come together to reach a decision on requirements. Although there are short-term workarounds for this situation (doing more detailed up-front work on Stories before they come into a Sprint), longer term, this needs to be solved. Otherwise, the delivery team will be unacceptably delayed.

Create your Requirements - First Backlog

One of the biggest cultural changes for teams (both customer and supplier) in agile projects is how the requirements are captured, recorded and elucidated. In the early stages of a new project, especially in a team using agile for the first time, creating an effective Backlog of Stories will be the piece of work that takes the most time and energy to create, understand, prioritize and estimate. But the effort will be worth it when the team beds down and starts to build a regular stream of working functionality that is fully tested and ready for deployment.

When creating your Backlog, there are golden rules to apply which will help you to be successful

- Create 'Ready to Play' criteria so that you can easily identify whether a Story is ready to come into a Sprint or not
- Create 'Definition of Done' criteria that means everyone understands what needs to be done before a Story can be delivered from a Sprint as completed. Make sure it includes User Acceptance Testing and tests to make sure non-functional tests are passed. This should also include updating any documentation that will form an output with the release
- Make sure every Story on the Backlog is a true 'end-to-end' Story
- Make sure every Story on the Backlog has a set of Acceptance Criteria that have been agreed between the Product Owner, a representative of the development team and a representative of the test team
- Make sure the Stories are prioritized as far as possible according to Business Value but that high risk or foundation Stories are also prioritized

Design Considerations

There is a great deal of available reading matter covering architecture and design for Agile projects. Some of this will advise that no design whatsoever is required up front, it can all evolve during the build cycle. Whilst others propose a much more rigorous up-front design process to ensure success. My own view is that the decision you make will vary widely from project to project and depending on technology. I make the following recommendations

- You do enough requirements understanding (building the initial Backlog) to understand and scope the scale of the project
- Do the same for the design – do enough up-front design to understand the risks inherent in your project
- If the project is high risk (e.g., from a non-functional perspective – a website requiring high throughput would be a good example) then do more upfront design
- Where high-risk requirements need to be delivered, deliver them early (or Stories to prove they are successfully delivered) as early as possible in the project
- Make sure they have automated tests that verify their delivery and run these regularly throughout the build process
- Where multiple applications form your landscape, again you will require more detailed design, especially where those interfaces are concerned
- Decouple teams as far as possible from versioning interfaces. The provider of each interface should then maintain at least 2 valid versions of the interface always.

This way, application releases do not need to be coordinated, making releases more flexible and back-out plans safer. The key thing to remember is to empower your delivery team (including Architecture and Design) to drive how much design they need rather than how much you have time to deliver. It is better to do too much up-front design than too little. Then, once your sprints are established and working well, try to do your design work on a 'just-in-time' basis but use Retrospectives and feedback loops to guide you.

Testing Considerations

For every Sprint, indeed for every Story, you will be making changes to your application stack that may well be changing code that has previously been updated for other Stories. This means you have the potential for injecting defects into Stories that have previously been delivered, possibly weeks or months or even years ago. So, at the end of each Sprint (and ideally at the end of each Story), we need to rerun as much of our regression testing as possible. For this reason, Agile development relies on test automation. Although it is possible to perform some limited Agile development with manual regression testing, it will severely limit your ability to release Increments and will allow defects to potentially build up in your codebase requiring an additional test / fix / retest cycle to be executed before each release. So, the golden rule with testing is budget for (in terms of money, time and resources) is implement test automation from the start. If you are modifying an existing application where little automation exists, things are much more complicated. By there are principles that you can use to guide you

- Try to automate the tests for any new development you do
- It costs about 3 times as much to instrument an automated test as to create and run once a manual test. So, any test you plan to run more than 3 times is worth automating
- Try to isolate your development (either at system boundaries or within the application) so that the unchanged parts of the application require little or no regression testing before release

The aim when developing each Story is to create 'live-quality' software. To do this, ideally, all the phases of testing that would take place before a conventional release (User Acceptance Testing, Operational Testing, Security Testing, etc.) needs to be performed within the Sprint, Story by Story. In addition, Stories are built to a set of Acceptance Criteria agreed with the Product Owner before the Story comes into the Sprint. We write automated tests to verify the Acceptance Criteria have been met (ideally using a framework like Concordia or Gherkin and Cucumber). This means the Product Owner (and probably other business representatives) will be involved on a day to day basis on the project, reviewing Stories with the team to create Acceptance Criteria then validating that completed Stories have met their Acceptance Criteria.

There are two popular ways to deal with non-functional requirements in Agile projects

- Deliver non-functional requirement Stories early in the project - prioritize them on your backlog and ensure when you deliver them you create automated tests that validate them. This way, every time you do a build and re-run your tests you know that the non-functional requirements are still being met
- Create a 'Definition of Done' that includes non-functional requirements validation. This will compel the teams to ensure that these are met for each Story they complete

Deployment / Go Live Issues Every organization has steps that need to be completed before a release can go live. These can range from additional specialized testing to sign-offs, additional documentation to automated Back-out facilities. These will need to be adapted to allow for regular releases - initially aim for a release every 2 to 3 Sprints, but once that is in place, a target to release every Sprint should be set. DevOps techniques should be embraced to automate the deployment process all the way through to the live environment.

Conclusion

Transforming to Agile is not easy. In the early years, it struggled to get established in mainstream organizations, partly because it was leftfield but mainly because there were so many stories circulating of failed attempts, sometimes spectacular failures. In almost all cases, these failures were caused because the process was poorly implemented.

The project I have described was my first experience of the impact of setting up a project poorly, cutting some corners to save time or cost, if making configuration changes to an existing application stack should be straightforward. It taught me some powerful lessons. Not least that projects need to be set up for success and if that means being disciplined about setting up the process, this is what must be done.

About the author

Subash Thota works as Data Architect and specializes in Big Data, Cloud, SAP, HANA, Data Integration and Data Analytics with significant experience in Project Management, Agile, and Data Governance. Subash has written several papers in the field of Big Data, the Cloud, and Analytics.

References

1. Extreme Programming-http://www.xprogramming.com /xpmag/whatisxp.htm.
2. http://www.gartner.com/imagesrv/summits/docs/na/business-intelligence/gartners_business_ analytics __21942 0.pdf
3. Scrum Methodology- www.scrumalliance.org
4. BI Rapid Delivery Framework
5. Beck, Kent; (2001). "Manifesto for Agile Software Development". Agile Alliance.