Implementing Agile in old technology projects

Sonik Chopra

Pitney Bowes India
C-31 Radhey Puri Near Krishna Nagar Delhi 110051 India
SonikChopra@Gmail.com

Abstract— **With this paper, I would like to share my experience of implementing Agile in obsolete technology projects. This is a reality that lot of our projects still run on old/obsolete technologies and customer base for these projects may be huge. These projects can be cash cows for the organizations and huge cost/effort is required to migrate them to latest technologies. Consider a banking or airlines application running on COBOL or FoxPro. Complex logics are built into these applications and they may be running successfully for the past many decades. Implementing Agile in such projects raises unique set of challenges. Can we really do rapid development with these projects? Can we commit that our development and testing efforts will be completed in a sprint in the next two weeks? Is our infrastructure supporting Agile development? And above all, are our teams ready to adopt Agile? I would like to share my experience of implementing Agile in old technology projects, the issues I came across and the path we followed to resolve the issues. Lot of issues are dependent on organization culture and other internal/external factors but the details mentioned below will definitely be helpful to understand the problems in detail along with ways to tackle it.**

Keywords— **Agile, Old technologies, Sustenance, Technical Debt, Agile Testing, Development practices**

I. Introduction

Lot of organizations these days are looking forward to implement Agile. Though it sounds very exciting in the beginning but when the real implementation is started then lot of issues pop up. And in most of the cases people conclude that Agile is not working for them. Whereas the reality may be that they have not looked back at their current ailing processes for the past many years. And on top of that if the teams are working on old technologies then this adoption can get more challenging. I have tried to capture different issues faced in development, testing, resources, management and operations side with some prospective solutions for same. There is no magic bullet to implement Agile but these guidelines will help to choose the right path.

II. Issues in implementing agile in old technology projects

There are different issues that teams can face while implementing agile in old technology project.

A. Agile not working for sustenance/support work

This is one of the common statements used that Agile is not suitable for sustenance work. Here sustenance work means that those projects where major active product development is not happening and teams are working on small features, bug fixes and support. Many of the old technology projects are in this phase as their core engine/Architecture is already robust and not changed in the past many years. And teams are working towards maintaining and supporting such projects. On further analysis of the issues, it was revealed that almost everyone was trying to implement SCRUM practices. No doubt that SCRUM Practices are very useful and have lot of benefits but we also need to consider the suitability for the project. Teams can face some unique set of challenges while working with scrum in these kinds of scenarios. Source code can be very old and unpredictable and it can get difficult to time box the customer bug fixes in a sprint. At times, teams may require flexibility to change the priority of bugs within a sprint and that is not recommended as per SCRUM practices. If Scrum is not suitable for a project then it doesn't mean that Agile is not working. Agile and Lean are big umbrellas where multiple methodologies are recommended.

For sustenance and support projects, Kanban may be more suitable. Kanban is a pull based approach that embraces continuous delivery of work. Kanban is a continuous flow process where issues are entered in a queue and pulled through a series of pre-defined steps. Kanban is often visualized on a Kanban board with each step represented by a different column. Work in progress limits are applied in Kanban to ensure that limited items are getting worked upon within a given stage of development.

B. Old technology code not ready for rapid development

Agile expects rapid development with potential shippable product commended by end of the sprint and this comes as a biggest challenge for old technology projects. A serious thought is required towards existing development practices. The source code can be quite complex and rapid development can further deteriorate the overall quality. Source code is the lifeline of any product and this turns out to be the biggest neglected area. Teams keep on adding code to the existing system without maintaining it and over a period of time it becomes unmanageable. Following practices are recommended to ensure that the code is not rotten and it is ready for rapid development.

1) *Automated Unit test cases*: Automated unit test cases are critical to maintain the stability of any product. Teams should target to add the automated unit test cases for all the new and critical features and over a period of time, there will be a good set of test cases to rely on. This will help in increasing the reliability

of product and the automation unit test cases should be run regularly to affirm the stability of the system.

2) *Code reviews*: Code reviews should be in place and coding guidelines should be defined and followed. This becomes more critical if the codebase is old, complex and multiple people have worked on it.
3) *Pair programming*: Pair programming can do wonders for old technology projects. It will always help to pair programmers on a complex logic. Another case is to pair an experienced developer with a new one. This will help in ensuring that system remains stable and the new developer also learns faster. Pairing of developer and tester can also turn out to be very helpful. They can collaboratively think of different scenarios and while developers are developing, testers can write test cases and can help in interim testing.
4) *Refactoring*: There should be dedicated time allocated towards refactoring of code. With multiple people working on old codebase, it is very critical to make sure that code is healthy and obsolete/stale code is removed regularly from the system. If this is not done then there will be high maintenance cost for the code in future.
5) *Start migrating small modules to latest technologies*: Always look out for opportunities to migrate the small modules to latest technologies. Migrating the complete project to new technology at one go is not possible most of times and these small steps will be very helpful in long run. Stay focussed on this migration and over a period of time the project would be in good shape. Most of the old technologies have one or another way to hook with latest technologies like an adaptor, bridge or a component that can communicate flawlessly.

Remember that this cannot be done in a day. A consistent approach towards maintaining a healthy code yields long lasting results.

C. *Lack of test automation tools for old technologies*

It can get very challenging to implement Agile in its true sense without the support of automation tools. And this comes as a roadblock for old technologies as there may not be enough automation tools available or they are no more supported. If a suitable tool is not available then following approach can be adopted.

1) *Regression test automation*:
   - Gather all the use cases you want to automate and look within the application for available solutions. Most of the applications support one or another form of batch processing. With slight modification and refactoring, batch processing can be reused as a very good regression suite to test different scenarios. For example, consider adding all your test cases in a spreadsheet and batch processing can execute all of them in one go.
   - If no option available then look forward to develop an automation suite. This will be developed in the same programming language in which software is developed or any other compatible language. Take small steps rather than coming up with a giant suite. Just concentrate on the use cases and build on top of that.
2) *Unit Test Automation*: If a suitable unit test automation tool is not available then develop one. This is not a complex framework and with a minimum effort it can be developed quickly. Some of old languages are not object oriented and it can get tricky to implement the unit test automation framework on first go. Some thought process is required here to figure out how the current code base can be implemented with automation unit testing and it may require some refactoring.
3) *Other test automation*: Other than regression and unit test automation, there are many other areas where test automation can be very helpful. There are lot many tools available for performance testing, security testing, API testing etc. With a proper plan in place, these tools can be very helpful in Agile adoption.

It is very important to have a plan and roadmap in place for test automation. Test automation tasks should be prioritized equally along with customers' features otherwise it would be very difficult to change the ailing state of the project. It is advisable to budget separately for test automation work with a dedicated pool of resources working on that. More effort will be required initially to get this started and it can reduce over a period of time.

D. *Inefficient software builds*

This again is one of the major issues with old technologies. Builds are generally a neglected area and over a period of time, it becomes cumbersome, time consuming and resource dependent. The resources who are working on the builds for years are comfortable with it but Agile expects the builds to be fast, robust and readily available. As the build process is running fine for the past many years so teams are generally very reluctant to change it and this result in the notion that Agile is not working for us. Following points can be followed to improve the build process and make it more suitable to Agile.

1) *Identify manual steps in build and automate*: It is very critical to get away with any manual steps. Identify all the manual steps and figure out ways to automate them.

2) *Identify the slow areas*: Identify the time consuming areas in the build and figure out ways to reduce the time. As the process is running for past many years, there is a big possibility of presence of redundant and useless code.
3) *Identify all the elements of build process*: Identify what all constitutes part of build process. Identify the not required, stale and useless parts and remove them. Also identify the list of files getting deployed. There may be files not getting used anymore and can be removed easily from the build process.
4) *Identify communication with external systems*: This can also be one of the major areas for slow builds.

E. Big technical debt

There can be different kind of technical debts existing in old technology projects.

1) *Defects deferred in each release*: With every release few defects are deferred to future releases but they are never fixed. These defects are accumulated over a period of time.
2) *Quick and dirty way to fix a problem rather than following the right path*: This will work for a short term but it gets difficult to manage system in future.
3) *Testing deferred to future*: or lack of good automation test suites.
4) *Not refactoring the code on regular basis*:
5) *Documentation deferred to future*:
6) *Lack of development skills*: resulting in poor quality of code.

It gets very difficult to make process changes and implement new technologies in the projects with big technical debt. These projects become very inflexible, rigid and unstable over a period of time. There can be different ways by which technical debt can be managed.

1) Dedicate a percentage of time with each release to clear off technical debt.
2) Dedicate a release or sprint towards technical debt
3) Implement the code in right way rather than applying a quick patch.
4) Ensure that teams are well versed with good coding practices and code is maintained properly.

F. Teams not ready to change

This again is one of the common issues faced while implementing Agile. It can get even more difficult while dealing with team members who are working on the old technology projects for the past many years. Teams generally know the project very well and are not convinced to change their existing processes. All the recommendations are of no use if the people who have to implement that are not ready to adapt. There are different ways by which this problem can be handled.

1) Talk to the team members in group and in person and try to understand their issues. Most of the times it is more of reluctance rather than any concerns. And these discussions can help to identify any major gaps and there may be some valid concerns too.
2) Make sure that the teams are well rewarded and morale is high. This is critical for the success of the program.
3) Arrange for trainings/coaching sessions and this will help the teams to realize real benefit of Agile. Work with the teams to understand their knowledge gaps and you can customize the training contents accordingly.
4) Introduce new members in the team and rotate people if possible. This will be very helpful in getting fresh ideas and can be instrumental in agile adoption.

G. Different organizations within the company working in silos

This is not an issue with old technology projects only and can play a major role in Agile adoption. Consider the scenarios where operations team has not adopted Agile and still working on their old SLAs. If the turnaround time by these organizations is say two weeks then this can be a high risk for Agile adoption because we'll not be able to deliver on time. Here are few points worth considering in this regard.

1) Getting changes in a different organization can be challenging at times. You've no direct control on them and getting your point of view implemented will be difficult. One way to approach this issue is to work through higher management and they further work with different organizations to enable Agile.
2) Involve different organizations in your scrum ceremonies like release planning, sprint planning and daily scrums. Consider them as part of single team and don't isolate them. This will help them to understand the dependencies better and they can adjust accordingly.
3) Generally procurement process takes long time and there can be rigid processes involved which cannot be changed overnight. And it might be difficult to change these processes at all because of legal constraints and organization culture. It is always recommended to come up with any procurement requirements (like an additional hardware/software) as early as possible to give sufficient time to the organization. And keep all the organizations apprised of the new processes you are implementing and this will help them to align easily.

H. Unable to cope up with Agile Metrics

This may sound irrelevant on the first look, but many teams do come up with issues related to Agile Metrics. Most of the teams relate Agile metrics to velocity and burn down charts and struggle to maintain the same. An important thing to understand here is that Agile metrics should always be light, practical and should be derived easily without big extra effort. Look beyond Velocity and see what all is required to track the health of the project. If you are following Kanban then tracking cycle time can be very helpful to understand the overall time taken to complete a task. Other important metrics include(but not limited to) trends for unit test automation and regression test automation suites, health of builds, technical debt reduced in a sprint/release, Static code analysis and code coverage metrics.

III. CONCLUSION

Agile methodologies expect rapid though sustainable development and this can be challenging for teams working on old technologies. With right kind of mindset, plan and zeal to change, this can be achieved easily. It is very important to keep the teams along while implementing these changes otherwise you'll not be able to cover that extra mile. Take small steps at a time, retrospect frequently and adapt.

REFERENCES

[1] Agile Testing: A Practical Guide for Testers and Agile Teams by Lisa Crispin and Janet Gregory

[2] Coaching Agile teams by Lyssa Adkins

[3] M. Fowler, Technicaldebtquadrant, 2009

[4] M. Poppendieck and T. Poppendieck, Lean software development: Published in Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th International Conferenc

[5] E. Lim, A Balancing Act: What Software Practitioners Have to Say About Technical Debt, N. Taksande and C. Seaman, Editors. 2012. p. 22-27.