



# Critical Success Factors in Distributed Agile for Outsourced Product Development

Raja Bavani

MindTree Ltd., Rajiv Gandhi Infotech & Biotech Park, Phase 1, MIDC,  
Hinjewadi, Pune-411057, INDIA  
[rajabavani@mindtree.com](mailto:rajabavani@mindtree.com)

**Abstract:** Agile Software Development and the breed of Agile Methodologies (XP, SCRUM, DSDM, etc.) have gained popularity since 2001. Primarily founded as methodologies for software projects executed at a single location, Agile Methodologies have started showing promising results in multi-site projects too with many adopters and practitioners across the globe. For more than two decades, offshore delivery models have been successful in case of application maintenance and enhancement projects. In case of development projects iterative lifecycle approaches have gained familiarity and acceptability compared to the classical waterfall approach in delivering results and ensuring customer satisfaction. Agile Software Development focuses on early delivery of working software to measure the progress of projects and hence to mitigate risks. It creates an environment that responds to changes by means of being flexible and nimble. It discourages creation of extensive documents that do not add any value to the customer. Distributed Agile Software Development and Testing is nothing but applying Agile Principles and Practices to software projects executed by teams located across geographies. This paper is based on our experience in executing Outsourced Product Development and Testing engagements using Distributed Agile practices. This paper presents Critical Success Factors that need to be considered while implementing Agile Software Development and Testing across distributed teams.

## 1. INTRODUCTION

In Outsourced Product Development & Testing arena Agile [1] has been the mantra of success during this decade. This trend is going to continue as the adoption of agile practices continues to increase with seemingly better rate of adoption. In order to exploit the availability of talent mix and associated cost-effectiveness in various geographies, distributed agile has started gaining momentum [4]. When it comes to executing multi-site projects, agile practitioners do face several challenges due to factors such as distribution of teams that results in limitations in face-to-face communication, and cultural mix of teams that impedes

team bonding. However, whether a collocated agile team embraces distributed teams and attempts to implement distributed agile or several virtual teams come together to implement distributed agile, the short term results are not impressive because of these inherent challenges. This paper, though it does not attempt to address all the nuances of a specific methodology such as Scrum or the details of Distributed Agile Software Development in general, intends to present a list of ten Critical Success Factors that need to be considered along with agile practices and principles to ensure project success.

## 2. CRITICAL SUCCESS FACTORS

Agile Software Development [1] in a distributed environment does not mean step-by-step implementation of any specific agile methodology such as Scrum with high expectations on on-time high-quality delivery. It means collaboration among distributed teams to collate processes that follow agile principles and to put together a methodology that works for them. Projects that follow distributed agile suffer when a methodology accepted by a sub team drives the rest of the team. Successful distributed agile projects happen because of collaborative teams that drive to define a methodology for themselves [4]. The definition of such a methodology happens by means of open communication and minor adjustments to make things work as expected. More importantly, a methodology that works for one distributed ecosystem may not work for another distributed ecosystem. This is because, for any methodology, while the basic tenets remain intact, the implementation details vary across ecosystems [2]. With several years of experience in executing projects with distributed teams we have derived a set of Critical Success Factors, based on our Best Practices as well as Lessons Learned, to ensure project success. The following sections in this paper discuss these Critical Success Factors.

### A. *Set up the Base Camp!*

In case of agile projects, collocated teams and face-to-face communication enable high level of collaboration. Collocated teams can have face-to-face meetings and discussions to iron out the base architecture, design guidelines and understand critical milestones. Hence coding can start from the first week itself, if not from the first day. On the other hand, in case of distributed agile projects setting up the base camp is very essential in order to put the right foot forward. Setting up the base camp involves forming a seed team (with at least 1 Project Manager, 1 or 2 Technical Leads and a handful of Engineers). Typically, members of this seed team are selected from distributed locations. They come together and spend 4 to 8 weeks depending on the size of the project, at a central location where the project initiates from. The objectives of setting up this base camp are:

- Creation of initial documents or wiki pages that provide project scope, high level requirements or user stories for the initial 2 or 3 iterations or Sprints, proposed architecture, designs of initial user stories, and high level roadmap or critical milestones.
- Defining the structure of distributed teams so that engineers at a location are not treated as augmented team members reporting into a lead sitting at a different location.
- Identification of additional Project Manager(s), and Technical Leader(s) depending on the size of the project.
- Identification of tools to be used to track project tasks as well as to track engineering processes such as configuration management/source code control, code reviews, defect tracking, etc.
- Define basic guidelines on source code control, build & release and related processes.
- Setup of Development, Staging and Test environments and connectivity to remote locations.
- Preparation for project kick-off meeting or video conferencing.
- Establishing a clear understanding of what to expect during the first few iterations or Sprints.

Distributed projects do not provide good results when some of these activities happen during project execution while a set of stakeholders continue to expect working

code and the project team strives to complete these preparatory tasks. This is the zone of frustration that could impact team morale. In order to ensure success, setting up the base camp is essential. When the base camp is set, induct the rest of the team and introduce them to the project. The journey will be progressive and successful.

During the first project, it is worth extending the base camp to produce working code for few initial stories – this helps in refining the build and release processes. Though it is not mandatory to setup a base camp for projects of the same nature, base camps of shorter duration can be considered to set expectation and kick start such projects.

Setting up the base camp provides several benefits. On project execution front this provides an opportunity to have adequate clarity on technical environment, tools and key engineering processes [3]. On people front this provides an opportunity for rapport building that can assure efficiency in resolution of issues and conflicts during the project. Overall, this provides an opportunity to start distributed agile projects on the right foot.

### B. *Ensure Explicit Delegation and Validate Assumptions*

Collocated teams can deal with crisp forms of delegation as there is abundance of avenues such as follow-up discussions or informal conversations to catch up with details. Distributed teams miss such opportunities. In distributed environments, delegation happens over chat, email, phone calls or video conferences. It is the responsibility of the delegator to be explicit in delegating tasks and setting intermediate milestones if applicable. On the other hand, team members need to master their skills in asking the right questions to understand necessary details.

Be explicit when you delegate. Mention the criticality, the background, the audience and the priority of task on hand. “I need these features implemented by Monday” is no better than “We have a management review and demo to senior executives on Tuesday, 11th August. So, I need these features for this demo by Monday, 10 August. Let us setup a video conference to go through these features before the management review.” Validating the understanding of team members is the only way to ensure explicit delegation. During conference calls over phone or video conferencing it is good to ask the team members to articulate their understanding of the situation or read out the action plan. In essence, ensuring explicit delegation



sets right expectations.

Assumptions made during initial stages of projects go unnoticed until end-users raise issues after the final delivery. There has to be a balance between ‘Uncertain Assumptions’ and ‘Convenience’ when we go agile. We need to make and appreciate certain uncertain assumptions to make progress in any project. However, at regular intervals, we need to clarify or validate such assumptions and make timely corrections. The impact of unresolved ‘Uncertain Assumptions’ on Product Quality could be fatal. In this context, customers’ perception on product quality would remain negative due to their initial experience during User Acceptance Testing. Besides, depending on the magnitude of such assumptions the overall product testing activities may have to be repeated in part or full to ensure a successful rollout. Finally, product release may not happen as planned. In distributed environments the chances of executing projects with ‘Uncertain Assumptions’ is more and hence an additional level of status check is required to have assumptions validated or clarified at regular intervals.

### *C. Cut Communication Loops*

Timely resolution of design issues or other technical problems provide clarity to developers as well as testers. When it comes to Distributed Agile, timely resolution of issues or timely selection of technical alternatives becomes very crucial due to the geographical spread of the team and the absence of onsite customer for face-to-face interaction and query resolution. In this environment, there are times when team members start managing technical discussions through emails, chats and telephonic communication instead of having a set target date to complete the discussion. First of all, it is detrimental to any budget driven project. Also, such communication loops will drain the energies of productive team members.

Observe long-running unproductive communication loops and resolve them. This is doable only when there is high level of communication and transparency across teams. Watch out for 1-1 interactions that show insignificant results. Facilitate the resolution and closure of stretched queries and streamline project progress.

For example, a smaller development team across the shore may start yielding to minor delays or multiple hops of communication from larger teams at other locations. In this process, the team members would keep certain technical discussions pending until a status check

or a major milestone. Distributed teams need to empower local leaders to identify and resolve impediments on a daily basis. Communication loops that are in progress may not appear to be true impediments. A way to identify and resolve impediments is to encourage leaders at each location to have informal discussions with their teams so that they understand the pulse of such situations.

### *D. Facilitate Tool Driven Query Resolution*

Email is not appropriate for query resolution for two key reasons. First, emails do not facilitate cumulative status tracking to reveal metrics such as average query age or the number of pending queries. Second, a valuable query and its answer may get lost over email between two individuals.

Query resolution over a chat session is not effective either. It is time consuming as it is a complex process that involves thinking, referring, investigating and typing. However chats are useful for a quick handshake or discussion.

The most appropriate way for query resolution is to use a web based query tracking tool. Timely query resolution provides clarity to developers as well as testers. When it comes to Distributed Agile, timely query resolutions become very crucial due to the geographical spread of the team and the absence of onsite customer for face-to-face interaction and query resolution. In this environment, there are times when team members start managing query resolution through emails, chats and telephonic communication instead of using a centralized query-tracking tool. First of all, it is valuable to use a centralized query-tracking tool. Next, it is important to watch out for pending queries and resolve them on time. Else, the team is constrained to work with ambiguity. This will impact product quality.

‘Stretched Query Resolution’ is one of the root-causes of schedule slippage in software projects. Team members tend to keep certain queries pending until there is a status check and a defined target date to resolve pending queries. In addition to query tracking using a centralized tool, this symptom can be recognized by interacting with individual team members to know about delays in query resolution and understanding the action orientation of email as well as other forms of communication. An added advantage of using a query tracking tool is the knowledge base that gets created in the project.

### *E. Initiate Test Drives*



Frequent delivery of working code to enable predictability and visibility is not sufficient in distributed agile projects. Leads and Managers at every location need to test drive the product at the end of each major milestone. It is very effective to plan a day of Test Drive at every location for every critical milestone to evaluate the external quality of the product. These Test Drives will provide valuable inputs to the management team and help them get a better view of predictability and visibility.

An added benefit of these Test Drives is that feedback on Test Drives will originate from all locations. This will keep all team members committed and challenged to improve product quality. Product demonstration happening at one location is not sufficient. Typically this is a demo where one engineer demonstrates and the rest of the team at that location observes. If such demos happen at a central location, team members across other locations may stay at the receiving end in dealing with feedback. Initiate and try Test Drives! This increases participation in understanding product health and collaboration in resolving issues.

#### *F. Assess Internal Quality*

Internal Quality primarily means Design Quality and Code Quality. Assessment of Internal Quality is to identify technical issues and to avoid large refactoring [5].

There are two primary dimensions of Software Quality namely, Internal Quality and External Quality. External Quality is an attribute that relates the end-user experience. External Quality can be assessed and improved through defect prevention as well as black box testing. Issues related Internal Quality could pose serious consequences in the form of unexpected naive defects, technical issues and maintenance nightmares. Poor Internal Quality encompasses the root cause for issues related to External Quality. Thus, in order to improve Software Quality, Internal Quality must be improved.

Trivial code quality issues occur due to various reasons such as a) introduction of new developers who do not understand the coding standards (implicit or explicit) followed by the team, or b) aggressive timelines that force team members to do quick fixes and dirty enhancements. A well-performing agile team produces consistent results. Whenever there is a change, for example, introduction of new team members, there is a good chance of encountering code quality issues.

Aligning new team members towards code quality is very critical.

#### *G. Manage Effort Variance Constructively*

In Agile projects effort variance can happen due to several reasons. Project Managers need to deal with effort variance constructively. It is good to budget for effort variance while executing distributed projects – especially during the initial stages. While dealing with effort variance it is required to make sure that the team is not stressed out or de-motivated to keep variance to 10 to 15%. Meanwhile, any trend that shows higher variance needs intervention in terms of collaborative discussions, to understand the driving factors or issues that contribute to such variances. Sometimes collaborative discussions on huge effort variances provide opportunity for optimization in the form of automation of test data creation or provisioning of additional infrastructure to improve productivity. Constructive discussions to understand the reasons for effort variance can be used to increase team morale [6].

#### *H. Take Stock of User Stories for Status Checks*

Setup a core team to take stock of user stories in addition to performing status reviews at the end of iterations or Sprints. This team needs to include Project Managers and Project Leaders across locations.

Each story has a life cycle. Define this life cycle based on the context of your project. For example, every story goes through various stages such as defined, coded, code reviewed, unit tested, integration tested, bugs reported and bugs fixed. Create a matrix or an excel sheet to take stock of all stories. At first, this is typically a day worth of work that delivers immense value. Subsequently this will require about 15 minutes of time every alternate week. Check if your tool can create this report. Status reporting based on burn down charts will reveal work progress in terms of efforts expended but will not provide any indication on product quality or indication in terms of outstanding defects per story. Taking stock of user stories helps project managers understand the work progress at a broader level. Especially in case of large projects this approach will provide a clear view of stories that undergo frequent changes or stories that are not completed because of technical issues or stories that are not tested because of show stopper defects.

#### *I. Invest in Root Cause Analysis*



Implementing a standard technique for Root Cause Analysis (RCA) [7] is very important to the success of distributed agile projects. A simple technique such as '5 Why' or use 'Fishbone Diagram' can be used to do Root Cause Analysis [8]. Skipping RCA will lead to poor decisions that may jeopardize the project. For example, if there are several P1 defects during an initial Sprint, the root cause can be something other than the technical competency of developers. Also, if a P1 defect is found during a demo or test drive, the root cause can be something other than the competency of test engineers. Incident based deductions need to consider root causes as well as trends for decision making. In RCAs of distributed agile projects it is required to involve all stakeholders or locations that are involved in critical incidents so that there is participation from all ends. Investing in RCA enables preventing underlying problems and hence facilitates cost savings.

#### *J. Compliment People to Improve Processes*

Appreciations motivate people to perform better. Appreciate on time. Ask for suggestions to address impediments. This will impress engineers to become creative and suggest process improvements. If there is a trend of consistent delays from an engineer then schedule a counseling session to discuss the trend and direct him to tasks that require his strengths.

Announce monthly or quarterly awards across locations to rope in all team members and create a charged environment.

One way to appreciate is to write an appreciation email or express words of appreciation over a conference call where all team members are present. Another way to appreciate is to empower people with new roles such as 'Code Quality Champion' or 'Build Champion'. It is the responsibility of the management team to come up with such innovative ideas to grow strong virtual teams. Engineers at disparate locations miss the opportunity of getting face-to-face appreciations from some of the key stakeholders of the project. This fact is often forgotten by practitioners who are very keen to optimize execution and improve productivity by focusing on negative factors in the project. Successful managers create a rewarding environment by appreciating teams for a job

well done. This motivates teams to optimize execution and to improve productivity.

Motivated virtual teams create the right environment for process improvement in the form of identifying areas where processes can be improved to save efforts. Such teams exhibit high level of ownership and ask the right questions to optimize their code, to identify reusable components, to automate the build process or to share expert tips in debugging or defect reproduction.

### 3. CONCLUSION

A methodology that embraces Agile Practices for Distributed Software Development and Testing is not the panacea to ensure on-time and quality deliverables. A great deal of conscious monitoring is required to exploit the benefits of Agile Practices especially in distributed or virtual teams. Critical Success Factors presented in this paper can be adapted to suit the context of any distributed agile project to deliver what the customer wants.

### REFERENCES

- [1] Alistair Cockburn, Jim Highsmith, *Agile Software Development: The People Factor*, Computer, November 2001. p. 131-133.
- [2] Amr Elsamadisy and Dr. Gregory Schalliol, "Recognizing and Responding to "Bad Smells" in Extreme Programming," ThoughtWorks, Inc., Chicago, IL 60661.
- [3] Barry Boehm, "Get Ready for Agile Methods with Care," Computer, January 2002. p. 64-69.
- [4] Brain Hoang, Bobby Khatkar, Joseph Momoh, Willie Tu, "Distributed Development and Scaling of Agile Methods," unpublished.
- [5] Martin Fowler, *Refactoring (Reading, MA, 1999)*, Addison-Wesley.
- [6] Paul Hodgetts, "Refactoring the Development Process: Experiences with the Incremental Adoption of Agile Practices," Agile Logic, Inc., Proceedings of the 2004 Agile Development Conference.
- [7] David N Card, "Understanding Causal Systems," CrossTalk - The Journal of Defense Software Engineer, Oct 2004. p. 15-18.
- [8] David N Card, "Learning from our Mistakes with Defect Causal Analysis," IEEE Software, Jan-Feb 2008. p. 58-63.