



**The ability to plan.
The flexibility to adapt.**

AGILE FOR DISTRIBUTED DELIVERY

Ten principles to address key challenges

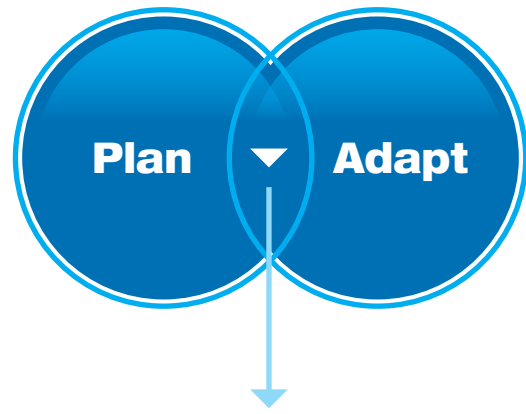
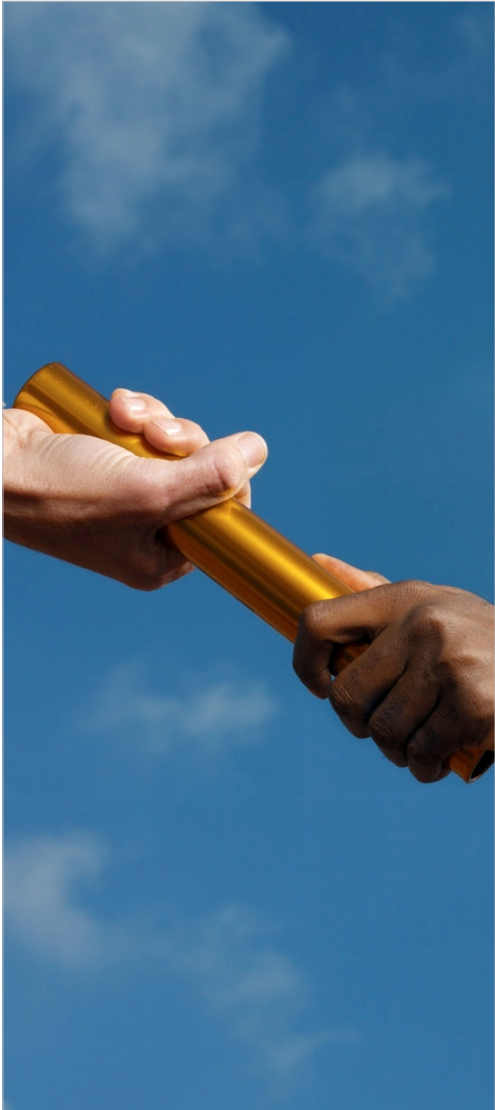


Rishi Raj Srivastav
Senior Architect,
Portal & CMS Technology Practice,
Mphasis UK Ltd.



Contents

1. Introduction	1
2. Illustration of an Agile Process & Deliverables.....	2
Process Metrics.....	3
Process Monitoring.....	4
3. Typical Agile Challenges.....	5
Visibility.....	5
Communication.....	5
Culture.....	5
Collaboration.....	6
Team.....	6
Rework.....	6
Configuration Management.....	6
4. Agile Principles.....	7
Principle 10: Kick off engagement with business workshops.....	7
Principle 9: Decompose requirements into useful stories.....	7
Principle 8: Adopt flexible and adaptive planning.....	8
Principle 7: Practice proactive risk management.....	9
Principle 6: Allow for frequent travel and selective co-location.....	9
Principle 5: Set up a continuous integration (CI) process.....	10
Principle 4: While documenting, create minimal waste.....	11
Principle 3: Have an open vendor partnership.....	11
Principle 2: Establish a regular rhythm of time boxed iterations.....	12
Principle 1: Believe in simplicity and common sense.....	12
5. Conclusion.....	12
6. Glossary of Terms.....	13
7. Further Reading.....	14



PRAGMATIC AGILITY

INTRODUCTION

Applying Agile methods for distributed delivery is about planning for a variety of scenarios —anticipating and developing a strategy, but not being so rigid as to lose the capacity to adapt and improvise when things don't go as planned. In the overlap exists a state of pragmatic agility.

This thought process and experience on various Agile projects has led to a core set of ten guiding principles that follow, which should provide a geographically distributed project team the required speed, yet allow enough flexibility to adapt to change in the creative ecosystem.

Earlier, the combination of agile and distributed delivery was always classified as high risk, but has now matured into low to mid risk with more and more distributed delivery organizations adopting and succeeding in Agile based delivery methodology. The paper covers ten distilled best principles that, when used well, can help address the key challenges faced in such a model. We begin with an example of an agile process, metrics and monitoring measures. Followed by key challenges faced while executing such a process.

Targeted Audience & Objective

This paper is intended for Project Managers, Delivery Managers, Architects, Tech Leads, Engagement Leads, Business Stakeholders and everyone else involved in collaborating, delivering or reaping benefits from a distributed, yet flexible, swift and iterative approach to software delivery. Basic understanding of Agile methodology is assumed. For people fairly new to the Agile methodology, please read through the glossary of terms at the end before reading the document, as it will give good introduction to the terms being used throughout the text.

ILLUSTRATION OF AN AGILE PROCESS & DELIVERABLES

The schematic below gives details of an example of Agile process adopted from Scrum, Lean, XP, and Unified models. This is an indicative process only and in this paper, the intention is not to explain the Agile methodology or the delivery process, but just to introduce some key principles that can help make it work in a distributed model. Reader can take a call on the Agile flavor and rigor he/she wants to apply based on his/her project context.

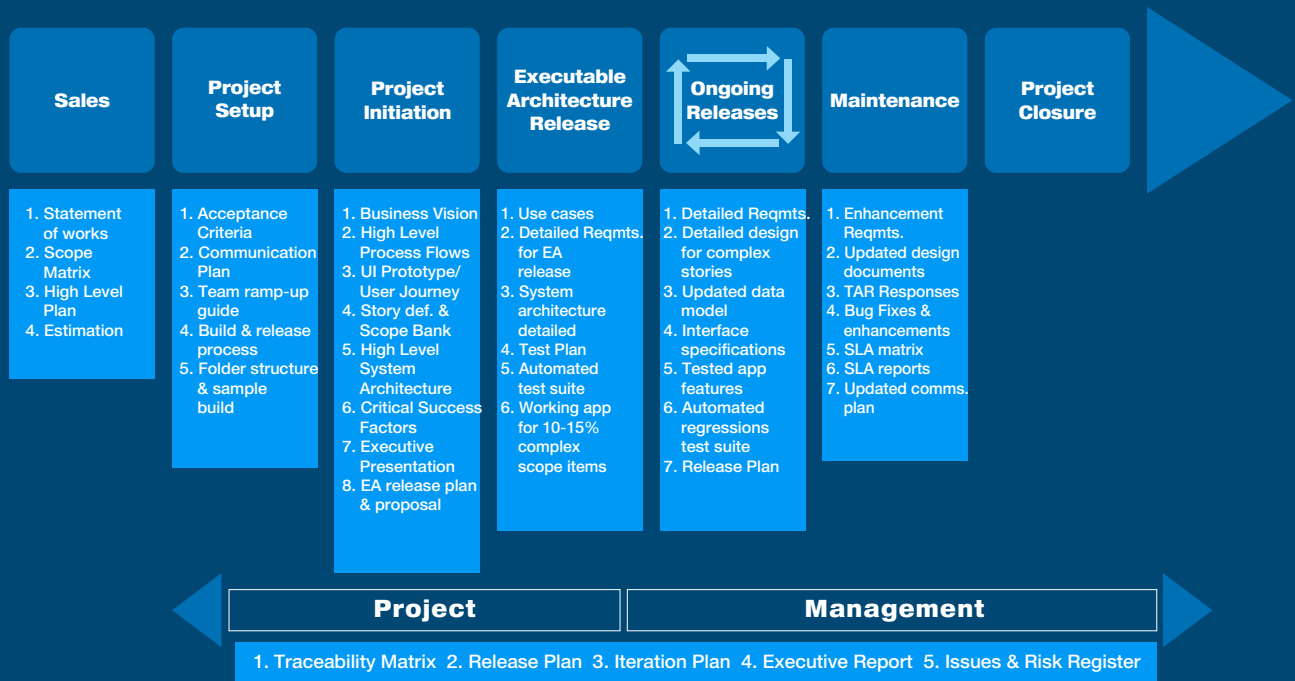


Figure 1: Example of Agile Process

- First, a small (typically 1-3 weeks) setup phase is required wherein we take care of some typical things which fall through the cracks for distributed projects and are always a cause of pain later, well-defined acceptance criteria, communication plan, team ramp-up guide and application infrastructure hygiene activities such as folder structures, build and release process with a sample are completed in this phase.
- In parallel to or soon after setup activities are done, Project Initiation is executed in the form of business workshops with key stakeholders. In this, we execute strongly facilitated workshops to communicate the business vision. This is the phase where the project direction and high level objective is understood and everyone is aligned to it – business stakeholders and the development teams together. Breaking of high level scope into well defined stories is followed by prioritizing the stories and putting them into a scope bank. High level process flows, UI prototypes and potentially user journeys, high level system architecture which would typically include logical architecture, physical architecture, data model and business services definition if needed, are also created in

this phase. The attempt is, however, just to document what we can at this point with confidence based upon what we know and not trying to evaluate the future needs and trying to create a design which would cater for this. Think about it as creating a quick whiteboard design which covers all components needed and their interconnections, implementing the clear business requirements which we are aware of. Business vision with critical success factors, scope bank and EA release plan would typically form the basis of executive presentation and subsequent EA release.

- Executable Architecture (EA) picks up the most complex pieces of the solution (technically complex, difficult user experience designs, or the ones with the most complex business logic) and validates the proposed system architecture. It should produce 10 - 15% of a fully working vertical slice of the application and is used as a basis for development in subsequent releases. EA release can be executed in a local or remote geography, the decision would typically depend on resource availability and bearing on cost involved.

- EA should be followed by an iterative sequence of ongoing releases, typically from the remote geography, broken down in to iterations or sprints (2 - 4 weeks long) till the entire application is delivered. Each iteration picks up the prioritized stories from the scope bank and there are weekly checkpoints with the stakeholders where the developed application is demonstrated and real-time feedback is received. Stories can be traded from the scope bank for new scope items or re-prioritized depending on the business need at the start of each release/iteration. This is typically done by the product owner(s). The release and iteration plans are adjusted based on the re-prioritized scope, actual effort spent by developers on similar stories in the last iteration, and other contextual factors based on previous release / iteration experience. If there is a corresponding change to the high level plan, that is made as well, and communicated to all stakeholders. Effort should be to trade or reduce scope, but keep timelines and thereby commercials the same unless it is absolutely required to change.
- Maintenance and project closure follow up. Project closure should definitely conclude with feedback, lessons learned being fed back into the process to close the loop.

PROCESS METRICS

Here are some of the key project metrics used in Agile which can improve visibility and bring issues to the fore, early in the cycle.

Project Management Metrics

- Velocity – Number of work units (stories) completed every iteration
- Daily count of stories completed, in progress, and not started
- Scope bank churn – Stories added or de-scoped
- Actual delivery time estimated for story versus estimated delivery time

Quality Metrics

- Code coverage: It describes the degree to which the source code of a program has been tested. It can be measured using tools such as EMMA, Clover, etc.
- Unit test pass rate: This indicates the quality of the code at a unit level. This can be a code method or a stored procedure or a cron job; unit

test everything you can. Unit tests can be written using frameworks like JUnit and pass/fail rate measured using tools like custom Ant scripts.

- Cyclomatic complexity: This is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. It can be measured using tools like PMD.
- Functional and non-functional defects per story: This can be reported using a combination of manual and automated testing.

Continuous Integration Metrics

- Number of check-ins per team size: This determines if all developers are checking the code regularly and guards against people working off a stale code base.
- Frequency of builds: This determines how updated and integrated the code base is at all times. This cannot be more than a day by any means and ideally is set to couple of hours by various Agile teams.
- Duration of builds: A build taking a long time to complete is normally not a good sign. On top of it, if there are build breaks or excessive unit test failures it mandates stopping new code development and fixing the build. A build taking too long to be marked as complete very often, would require project management intervention and root cause analysis.
- Rate of build breaks: This needs to be kept at minimum and usually is a good lag indicator of design and code quality, leading to wasted team cycles.
- Duration of build breaks: Build remaining broken for longer durations normally indicates more serious underlying issues, such as broken contracts between system components, stale code base, lack of skills, etc.

PROCESS MONITORING

Burn-down chart

A burn-down chart tracks how much work remains on your project and whether you will hit your deadline. The vertical axis measures work remaining. The horizontal axis marks your iterations, and you should mark which iteration is your target end date for the project. After each iteration, mark your progress on the chart and you can project forward to see whether or not, you will hit your target end date.

DAILY BURN-DOWN

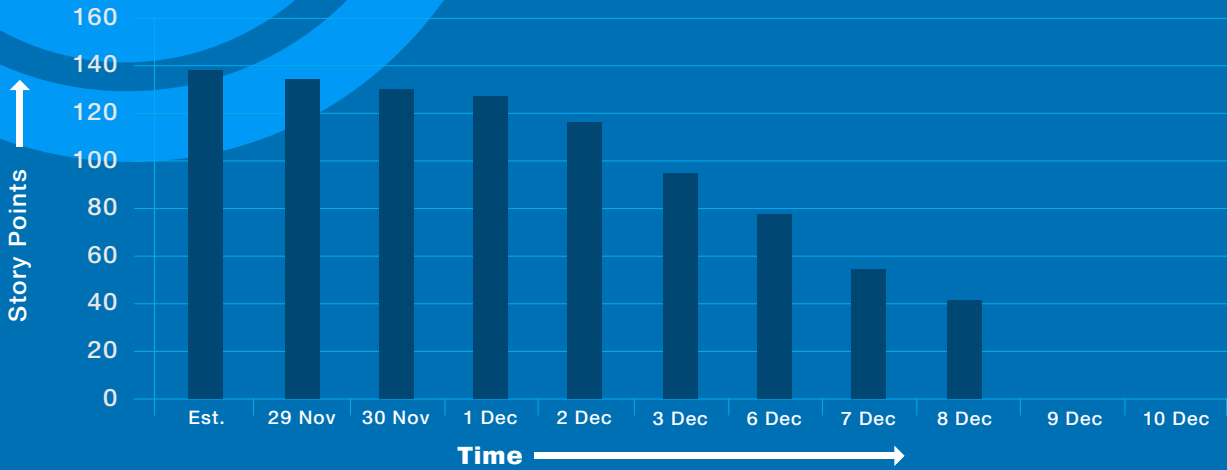


Figure 2: Burn-Down Chart

Burn-Up Chart

A burn-up chart tracks how much work is done. But it shows more information than a burn-down chart because it also has a line showing how much work is in the project as whole (the scope as workload), and this can change. On the burn-down chart, it is harder to show a changing goal.

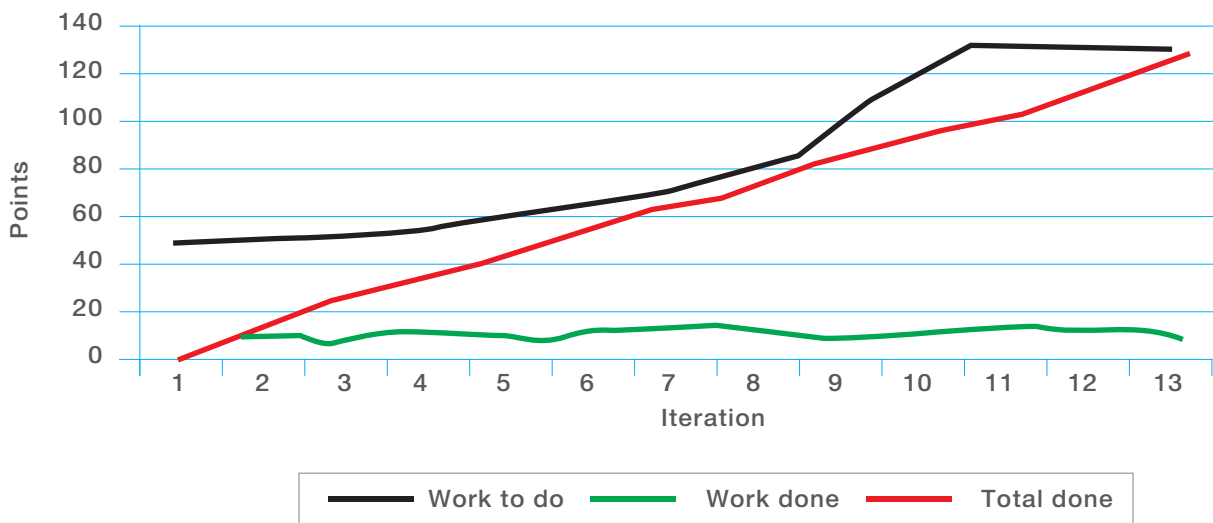


Figure 3: Burn-Up Chart

TYPICAL AGILE CHALLENGES

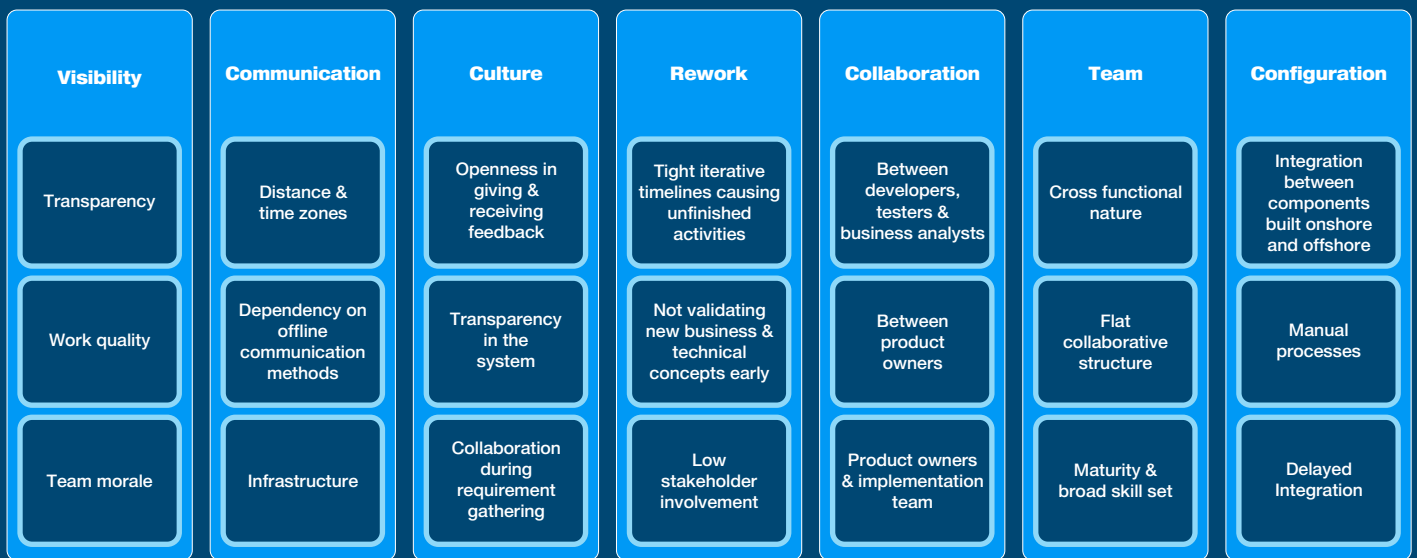


Figure 4: Typical Agile Challenges

Visibility

No one raises any issue but project still gets delayed. This is a big challenge for the PM to ensure that the issues are raised and tracked. Increasing transparency in the whole process between product owner and the project development teams, within the development team, and also between the development team and other supporting and auditing functions within the development organization, helps. If anyone is hiding or trying to hide any facts in an Agile delivery model, be alarmed. Work quality and team morale are two other aspects of Agile teams where the visibility can be masked very easily.

Communication

The communication gap among the team members is a typical challenge that a lot of Agile projects, which are so extensively dependent on active collaboration, struggle with. Being dependent only on project leadership, documentation, formal meetings, and e-mails is a bad old habit which need shedding. Also, due to time zone differences between western business centers and many of the major offshore development sites in India and

China, there are very few hours in the day when project participants are in both office locations at the same time. These factors, as well as the current cost and quality of telecommunications, serves to significantly decrease the volume and quality of communication between offshore and on-site teams. Team members need to be more reliant on direct or phone based, one-to-one verbal communication, which has the least possibility of gaps or errors. Be open, raise issues and seek help during daily stand-ups when you have not made progress in the day. This is a habit that needs to be instilled at all levels.

Culture

Culture offers both local and global challenges to software teams as they collaborate to understand requirements, build systems, and deliver product. Agile software practices through iteration, incremental delivery, and customer proximity can overcome cultural challenges to create synergies. Being open and transparent with issues and using the simplest and therefore most robust solution on the table using pure common sense are two values which transcend culture. Make use of them.

Collaboration

A closely knit collaborative team is likely to make full use of Agile principles and reap benefits sooner than other teams. Agile is about collaborating, raising issues early and also helping others resolve it. It is about picking up unfinished stories from others if you are done early, it is about helping others finding faults early in the game with their implementation rather than apportioning blame after the code is built and deployed. For development team, collaboration starts at daily stand-ups. Collaboration is needed between development team members during the development and testing cycles, business users and development teams during requirement gathering and also among business users to be able to correctly prioritize a requirement over other during project and iteration backlog meetings. This collaboration and honesty needs to exist, keeping all personal differences and politics aside.

Daily Stand-ups

Stand-up meeting is a daily team meeting held to provide a status update to the team members. The status allows participants to know about potential challenges as well as coordinate efforts to resolve difficult and/or time-consuming issues. The meetings are usually time boxed for 5–15 minutes and are held standing up to remind people to keep the meeting short and to the point. The meeting is usually held at the same time and place every working day. All team members are expected to attend, but the meetings are not postponed if some of the team members are not present. One of the crucial features is that the meeting is intended to be a status update to other team members and not a status update to the management or other stakeholders. Team members take turns speaking. Each member talks about his progress since the last stand-up, the anticipated work until the next stand-up and any impediments they foresee.

Team

For Agile to be successful, it needs people with emotional and business maturity. Team members should understand the cross functional nature of the methodology and also be able to learn and pick up a new skill required to implement a business functionality. It is important for the team to realize that technology is really a business enabler; it serves best when it results in useful productive

software, rather than a perfectly crafted masterpiece. This may not always be possible especially with geographically dispersed teams, economic pressures and reduced bench strengths. This needs to be addressed by optimal use of closely knit Agile team structure, frequent travel, and good coaching by an experienced mentor.

Rework

Whilst not following 'best practices' and moving at the pace required in Agile, it is very easy to accrue a lot of rework to be done later. This affects project schedules, quality of deliverables and of course has a cost implication. Good Agile practices avoid both rework and rebuilding of finished products. High level design and concepts validated in the executable architecture release phase, business stories elaborated as late in the schedule as possible to make the best and most informed decision using unit tests & customer defined tests for each story make sure "done is done" in Agile.

Configuration Management

"Bringing it all together" for implementation in the integration, testing and production environment has always been a challenge. Many teams that have built components offshore have seen huge problems when the time came to integrate the offshore and on-site pieces, or even multiple offshore pieces into a working system in a remote integration environment. Defining team structure to address vertical slices of business functionality (across all technical components), continuous integration, automated build and deployments and unit testing are just a few arrows in your Agile quiver to handle this age old enemy.

Continuous Integration

Continuous Integration is a software development practice where members of a team integrate their work frequently. Usually each person integrates at least daily leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

AGILE PRINCIPLES

Below are a set of ten key principles which are industry proven in delivering large Agile projects in a distributed delivery model. While there can be numerous others added to this list, this is a distillation of the elements which are instrumental in the delivery process. If followed, there are very good chances the Agile project you are leading or intending to benefit from would stay on track and get you the benefits of reliability, speed, and flexibility that we all so desperately seek in a mature software development methodology.

Principle 10: Kick off engagement with business workshops

It is imperative to understand the objective of the engagement and the end state of the application intended for build/enhancement. For this, a team can often make more progress in a well-executed 2 - 3 days of business workshop than in a few months of research, individual interviews and one to one meetings. This is an adoption of the accelerated business requirements technique and works very well when the intention is to align all the business stakeholders quickly and get a clear objective to kick off the work. A business workshop should be a strongly driven and facilitated working

session typically over three days with a delivery team and a group of 6 to 10 carefully selected stakeholders. This is typically executed during project initiation phase just before the EA release. The goal is to extract decisions using the knowledge and experience of the stakeholders in the room. Each workshop requires diligent preparation, specialized or prepared facilities, and an experienced team of facilitators and recorders. It should deliver business vision and prioritized business requirements in the form of a scope bank.

Principle 9: Decompose requirements into useful stories

User requirements need to be broken down into “useful” user stories typically during the project initiation phase and then reprioritized/discussed during the start of every release. A useful story typically implements business functionality so that it is testable, small enough to complete in an iteration and independent of other stories in its own iteration. Story cards are typically written by the product owner to articulate business needs during the business workshops introduced above.

Best practices whilst creating story cards:

- a. Story cards must be small and complete.
- b. Avoid dependencies between story cards within an iteration.
- c. Be brief – Write stories in 2 to 3 sentences.
- d. Keep the story card simple; define business terminology upfront.
- e. Each story needs to be capable of being estimated to have a business value, be small, be testable, and independent.
- f. Keep the stories consistent – Do not include contradictory, incoherent, and conflicting statement which would lead to wrong interpretations.
- g. Keep a link between story card and stakeholders and take that information to traceability matrix. This helps get questions answered and issues clarified when they arise.
- h. Define customer tests as part of the story creation exercise, so we know how to validate it has been built correctly.
- i. Keep a guideline handy for story prioritization.
- j. Include non functional requirements on the story cards when needed.

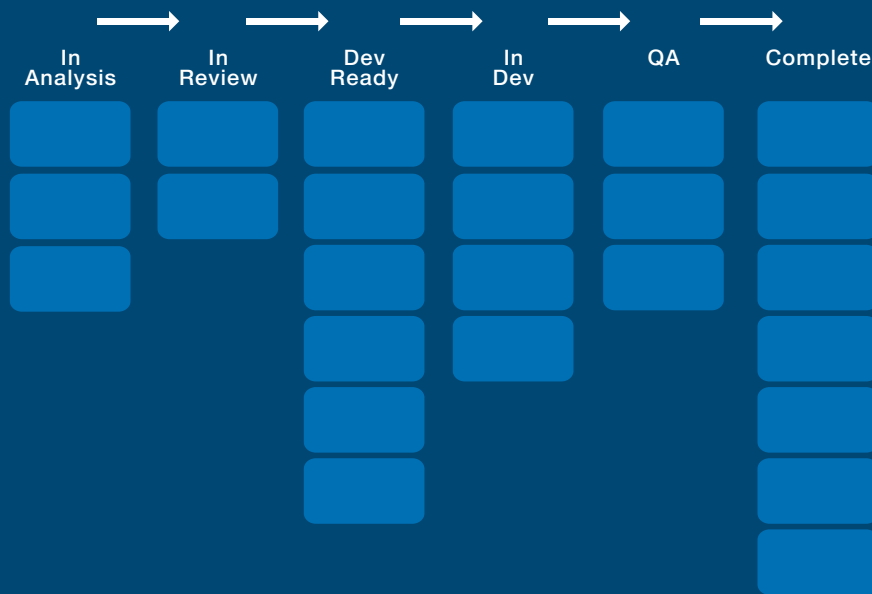


Figure 5: Scope is broken into stories and then tracked to completion

Principle 8: Adopt flexible and adaptive planning

If you plan for a project in a conventional manner, you would define your plan fully up front and try to track to it. Plans are usually base-lined and deviations are quite often not allowed. This discourages you from changing the plan, even as you learn some of your original estimates need adjustment based on actual effort spent, or as the project changes. As a result, you don't have a credible way of understanding where you truly stand in relation to completing, what you set out to deliver. Your Agile plan should typically have three levels, each with successively more detail and a shorter time period as depicted in the interlocked gear diagram shown:

- a. High Level Plan - Shows key milestones and bounds the project on a 6 - 24 month timeline
- b. Release Plan - Maps all the project stories across iterations for a single release lasting 1-3 months, identifying key internal and external dependencies, assumptions and risks
- c. Iteration Plan - Maps stories across a single 2 - 4 week long iteration, with each story broken down into very detailed tasks, with clear exit criteria outlined for each story contained in the iteration

In each iteration, the Project Manager works with the teams to change all three levels, revising the High Level Plan and Release plan, and creating the Iteration Plan for the next cycle. This maximizes the ability to learn from each time boxed iteration, maximizes flexibility to change the plan, and minimizes wasted effort in re-planning.



Figure 6: Flexible and adaptive planning

Principle 7: Practice proactive risk management

Risks are part of the real world in any project execution; expect them. In Agile projects, encourage risks to materialize early. Plan for risks. Plan for the project to go wrong. As early as possible, attempt the most challenging work in the most business-critical areas of the project (where the impact of an issue would be greatest). By front-loading a controversial user experience design, integration of a new release of a major software package, or even the testing of complex business processing, when you discover issues, you'll have the maximum time possible to adjust course, remain in control, and have the best chance of steering the project to success.

The Executable Architecture release phase is all about risk mitigation. You create an early working version of part scope of what you are building, including the most challenging and business-critical work – 10-15% of the overall functionality – in a real working environment. If you defer a major integration issue to later in the project, chances are you will not have sufficient time to address it as thorough as necessary. As a result, you will be forced to cut either scope or quality.

Executable Architecture

An executable architecture is an implementation that realizes the Software Architecture. It is used to validate that the 'architecturally significant requirements' are correctly implemented. It validates the architecture as an integrated whole through integration tests. The team gains feedback about the architecture from the customer or stakeholder by providing the executable architecture for verification. This way the executable architecture helps to assure that the core functionality is stable enough to build the remainder of the system

It is a recommended activity in the Unified Process* as part of the elaboration phase wherein the partial implementation of the system serves to validate the architecture and act as a foundation for remaining development.

(*The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework.)

Principle 6: Allow for frequent travel and selective co-location

Co-location is one of the key tenets of Agile. For large and geographically distributed teams, it is not always possible to achieve co-location. This can still be addressed by following some basic principles which are easily achieved in a distributed model as well.

During the EA release, have the product owner(s) travel to the remote geography (if EA is happening out of remote geography). This will help clarify high risk requirements, change architectural options, and seek real-time feedback on POCs and customer tests.

Have a rotation plan for Business Analysts / Product Owners, Testers wherein they work on the next iteration requirements and customer tests and then travel back to the development geography to work with the team while executing that iteration. Also, allow for travel of a stakeholder to the remote geography to answer any requirement questions and/or to close on a contentious user experience design, seek clarifications from the business.

Stories defined should have real business value and there should be well defined business milestones at the end of each week/iteration. Have weekly / iteration checkpoints with the key stakeholders and showcase everything developed, seek real time feedback.

Plan for some hours of overlap between local and remote geography teams for a few hours in a day for things such as requirements clarification, bug prioritization, demos, etc. A daily 20 minutes sync-up after daily stand-ups, between the development team and the product owner for clarifying requirements and reconfirming the priorities works very well.

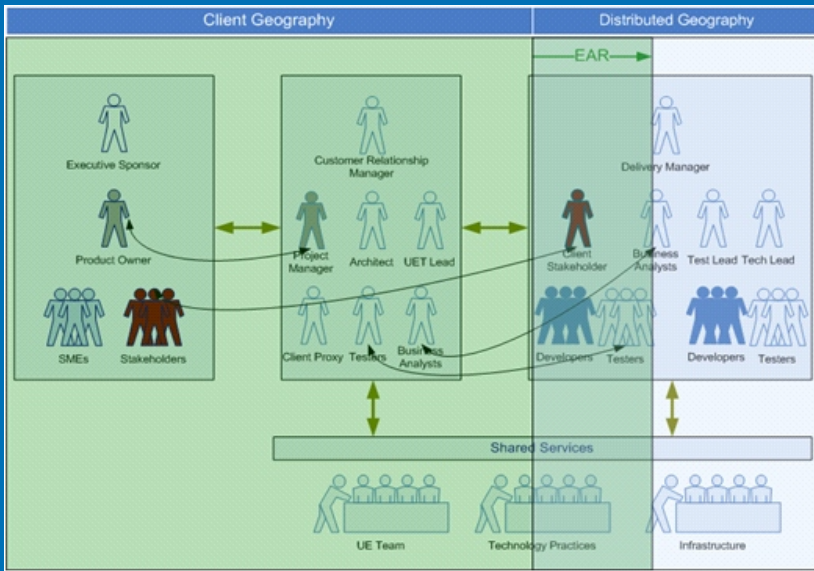


Figure 7: Travel and co-location

Principle 5: Set up a Continuous Integration (CI) process

The team can deliver completed work with short iterations only if the right technical infrastructure and lightweight technical processes to validate the quality and completeness of all stories are in place. If not, tasks accrue for later and also potentially introduce a lot of rework.

The most critical piece your team will set up in Agile process, is Continuous Integration (CI), a process and environment that allows developers anywhere in the world to submit code to a common code base and to automatically kick off the build process. This way, your team immediately sees problems that surface (via automated e-mails) and fixes them, leaving a stable code base in place at all times. This should be extended to allow for automatic deployment and execution of automated regression tests at least once in a day / few days (less than a week though). This will save long stretches of time after integration releases to fix integration issues / rework code which doesn't integrate well. Your team will also setup the

following processes to complement CI:

- Test-Driven Development (TDD):** Developers first write a unit test for each module of software, and then write the module, resulting in complete unit test coverage for all code, as well as reduced waste in unnecessary code. Produce code coverage and percentage pass/fail report after each build, have a minimum value for acceptance.
- Automated testing:** Once your testers successfully execute a functional test, they will develop an automated test to replace the manual one, adding it to the automated test suite that's part of the CI build.
- Full lifecycle QA:** Your testers are part of the team as soon as you plan any stories that involve software (within the first 1-2 iterations at the latest). This is an example of risk mitigation; all testing is front-loaded in the overall plan so that issues surface early and your team can promptly address them.

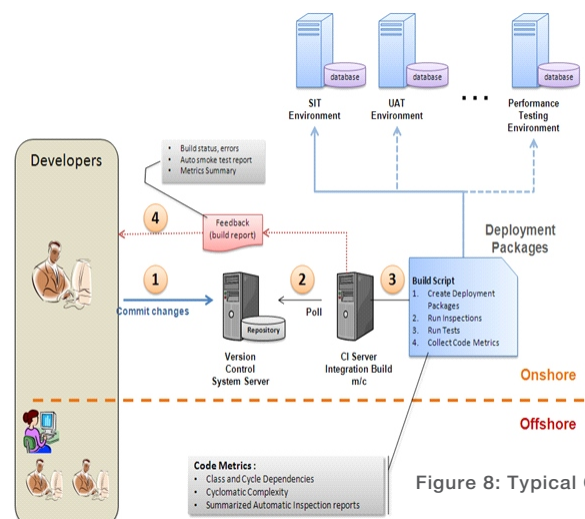


Figure 8: Typical CI Setup

Principle 4: While documenting, create minimal waste

Agile's manifesto values “working software over comprehensive documentation”- something that can be achieved if project development, management, future enhancement is all contained within one team or limited number of very well connected vendors. In most cases this would not be the case, in those cases and as a general rule create minimal waste, but do create the basic documentation required to keep the communication going between different geographies and vendors.

Define use case levels and templates. While fully dressed use cases will be very rarely needed if you are following Agile, do one pager high level use cases for all stories during project EAR and ongoing releases phase. Keep them simple and strictly to one page to allow frequent updates with minimal effort. In some cases, you might need mid-level two-page descriptions in cases where there are a lot of business rules and applications to integrate.

A user journey for the entire application and UI prototypes for key pieces are useful to elicit feedback from the business users and is helpful in passing the context “visually” to the remote teams.

Create a high level system architecture diagram (typically containing logical, physical architectures, and data model. For SOA based projects, have the business service definition done here too) as part of the project initiation phase. This highlights architectural flaws early and also can be updated based on findings from the EA release phase. Keep the system architecture document updated as part



of the subsequent releases, which initially sets the basic integration mechanism and architectural direction.

The scope bank along with high level use cases, user journeys/UI prototypes and a system architecture document would fulfill the minimal documentation that the remote implementation team would need to get the basic context. This should be backed up by Product Owner and architect / business analysts / testers travelling to the development geography and also audio/video conference calls with local and offsite teams as needed.

Whilst in most cases white board designs work, define a mid level design template for complex and high risk stories. For all designs, do a walkthrough with one to two technical stakeholders early in the cycle to avoid surprises/changes later. Set expectations for this involvement upfront in the project setup phase.

Principle 3: Have an open vendor partnership

You run and know your business. Hence, an open partnership with your vendor will make your project more successful. Value vendors' input and include them in your processes, plans, and decision-making to a degree that makes sense with their schedules. Be open and honest with project issues.

Ultimately, the Agile methodology hinges upon an open relationship at every level. Your vendor should always have full access to the project management tool so that the actual status of each iteration is always visible. If you have a bad iteration, be prepared to be honest and open; you will be surprised how forthcoming all stakeholders will be to work to resolve the problems.

Formalize any vendor partnership with the product owner or a similar role as a single point of contact empowered to make scoping and planning decisions for the project. In a consensus environment, or for a larger program, you can fill this role with a team or a committee. The Product Owner works with the Project Manager to validate all the work to be completed as a list of stories. The product owner needs to travel to remote geography for EA release (if EA release is being executed offshore, first attempt should be to execute it on-site) at minimum and possibly at the start for all subsequent releases to minimize information gap and also help team adjust priorities if they hit a roadblock. At minimum, keep the daily 20 minute sync-up between product owner(s) and the development team just after the stand-ups going.

Principle 2: Establish a regular rhythm of time boxed iterations

Break down all scope into stories and stories into time boxes. Commit to a target, fix the end date and execute it. Within the time box, you must complete the planned work. There is no concept of partial credit. **“Done is done.”**

There are clear tests that the team must pass to get credit. Time boxes also help you improve estimation as you move forward. At the end of each time box, you can see if you are still on the right track to meet your end goal. Regularly stepping back and evaluating progress is healthy; your goal may change, and what you learn with each time box

can be applied to the next one. Avoid long stretches of time where nothing is delivered. Keep your iterations the same length throughout the project. This allows you to establish a regular rhythm and to understand your velocity so that you can plan for future iterations.

Principle 1: Believe in simplicity and common sense

Agile is a better way of developing productive software quickly and is based mostly on common sense. So when you are stuck with an important decision to make and there is no well defined process which addresses it, pick the simplest option which makes most sense in the situation.

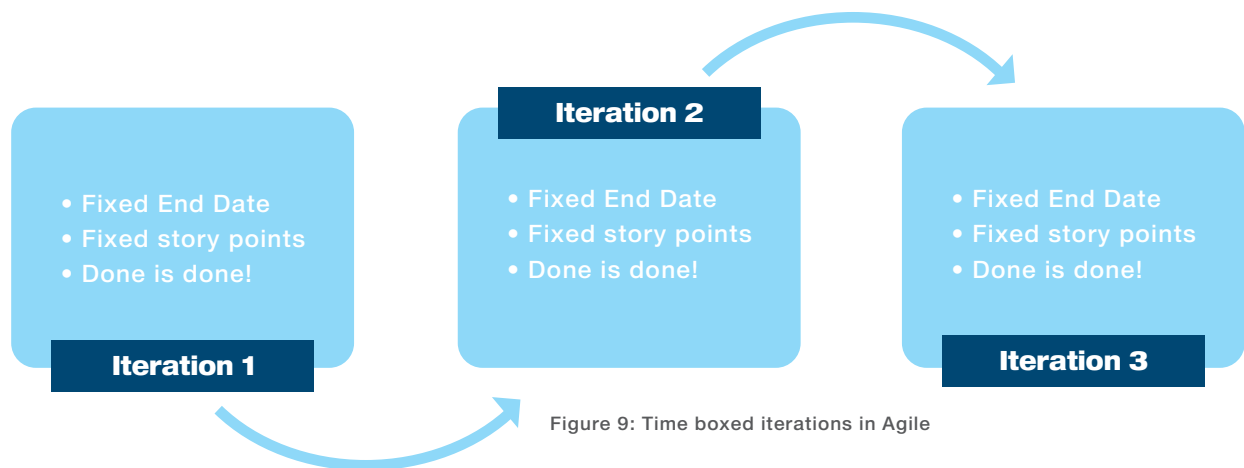


Figure 9: Time boxed iterations in Agile

CONCLUSION

People who have done it right will tell you Agile is really a better and faster methodology for delivering successful software projects. However, it does involve a different mindset to adopt and execute in such a model. Execution may seem challenging when teams are large and dispersed across geographies, cultures and time zones. Initially it seems intimidating to work using the agile methodology and some teams which start using it get frustrated and quickly resort to blaming it on the methodology. They fall back to the conventional waterfall methodology they are used to. While this may seem comfortable, it really does not bring the required return of investment to the business organizations at the required pace.

Accept the ambiguity in and today's distributed nature of, large software delivery. Learn and assimilate the new methodology, challenge and

seek answers to the doubts that rise, from experienced people/organizations who have executed in this model successfully. Use the principles introduced in this paper and initiate the change. Once you have adopted the new model, build the most complex functionality first and validate the architecture by doing it than just hearing about it. Monitor the progress made in the iterations by using various metrics and tools available and when you seem to be deviating from the plan, adapt to the change. Adjust plans, take lessons learnt back into the process and tweak it. Be driven by market needs, get the most important features out there quickly using the simple and most robust design options. Listen to your stakeholders during execution, and once you are done, analyze the feedback and adopt the learning.

6. GLOSSARY OF TERMS

Business Vision: This is usually a brief paragraph or a one pager in simple clear text describing the end state of the software application being built and the benefit business intends to derive out of it.

Continuous integration (CI): Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily leading to multiple integrations per day. For more details, look at the side bar.

Customer tests: In Agile development, typically for each story, we ask the customer to tell us how he/she will know when a story is complete in the form of tests and examples. These tests are ideally in a form that can be used with an automated tool, but they may also be higher-level tests, or guidelines for later exploratory testing.

Distributed Delivery: It refers to a model often used in software services industry where delivery is spread across multiple geographies with part of the team (typically business analysis and design) is co-located where business locally and part of the team (typically development) is based out of a remote geography.

Executable Architecture: An executable architecture is an implementation that realizes the Software Architecture.

Extreme Programming: Extreme Programming (XP) is a software engineering methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles (time-boxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted. Other elements of Extreme Programming include programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, simplicity and clarity in code, expecting changes in the customer's requirements as time passes, and the problem is better understood, and frequent communication with the customer and among programmers.

Lean: Lean programming is a concept that emphasizes optimizing efficiency and minimizing waste in the development of a computer program. The concept is that efficiencies can be applied and waste managed at all levels - each individual, every department, interdepartmental operations, the organization as a whole, and the relationships of the organization with customers and suppliers.

Offshore: The remote (offshore) team is usually a part of the technical team. This team is effective at taking development packages (stories, system architecture and UI artifacts) and developing, customizing or configuring applications to meet the desired results.

On-site: In a distributed delivery model, on-site team typically includes anyone that, by necessity, needs to be based locally. It always includes an on-site project manager that interacts with the teams for any scheduling, scope, change, cost, quality, issues, risks and customer satisfaction issues. Other resources may include Business Analysts Or Systems Architect (SA), etc.

POC: POC or Proof of concept is typically executed to validate a design concept, usually a use and throw construct.

Product Owner: The Product Owner (typically someone from a Marketing role or a key user in internal development) owns and prioritizes the stories.

Scope Bank: This is typically a spreadsheet which contains list of stories, their prioritization, estimates and release they are scheduled in. Stories in this scope bank can be traded for new scope.

Scrum: Scrum is an agile process for software development. With Scrum, projects progress via a series of iterations called sprints. Each sprint is typically 2-4 weeks long. A key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need (often called requirements churn), and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. As such, Scrum adopts an empirical approach, accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements. Scrum enables the creation of self-organizing teams by encouraging co-location of all team members, and verbal communication across all team members and disciplines that are involved in the project.

SLA: A Service Level Agreement (frequently abbreviated as SLA) is a part of a service contract where the level of service is formally defined. In practice, the term SLA is sometimes used to refer to the contracted delivery time of the service or performance.

SOW: A Statement of Work (SOW) is a document that captures and agrees the work activities, deliverables and timeline that a vendor will execute against in performance of work for a customer. Detailed requirements and pricing are usually specified in a Statement of Work, along with many other terms and conditions.

Stand-ups: Stand-up meeting is a daily team meeting held to provide a status update to the team members. For more details look at the side bar.

System Architecture: This is a high level design document typically comprising of logical and physical architecture and data model. In some cases, it may also comprise of business services and high level interface specifications.

TAR: Technical Assistance Request is typically raised in a maintenance project by the stakeholders in response to a user complaint or a new enhancement to be introduced in the system.

User Stories: This is a software system requirement formulated as one or two sentences in the everyday or business language of the user. User stories are used with Agile software development methodologies for the specification of requirements (together with acceptance tests). Each user story is limited, so it fits on a small paper note card, usually a 3x5 inches card, to ensure that it does not grow too large. The user stories should be written by the customers for a software project and are their main instrument to influence the development of the software.

7. FURTHER READING

- Agile Manifesto - <http://agilemanifesto.org/>
- The New Methodology - <http://martinfowler.com/articles/newMethodology.html>
- Agile software development ecosystems – Wiley (Cockburn, Highsmith)
- Rational Unified Process – Best practices for software development teams:
http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
- Agile Architecture: Strategies for scaling Agile development - <http://www.agilemodeling.com/essays/agileArchitecture.htm>
- Continuous Integration - <http://martinfowler.com/articles/continuousIntegration.html>
- Why should business people care about continuous integration -
http://www.agileadvice.com/archives/2005/07/why_should_busi.html
- Appropriate Agile measurement - <http://www.berteigconsulting.com/AppropriateAgileMeasurement.pdf>
- Agile Metrics - <http://www.slideshare.net/Siddhi/agile-workshop-agile-metrics>
- Role of Agile coach -
<http://www.agilejournal.com/articles/columns/column-articles/1917-the-role-of-the-agile-coach>



ABOUT MPHASIS.

Mphasis is a \$1 billion global service provider, delivering technology based solutions to clients across the world. With currently over 41,000 people, Mphasis services clients in Banking and Capital Markets, Insurance, Manufacturing, Communications, Media & Entertainment, Healthcare & Life Sciences, Transportation & Logistics, Retail & Consumer Packaged Goods, Energy & Utilities, and Governments around the world. Our competency lies in our ability to offer integrated service offerings in Applications, Infrastructure Services, and Business Process Outsourcing capabilities. To know more about Mphasis, log on to www.mphasis.com

For more information, contact: sales@mphasis.com.

USA: 460 Park Avenue South, Suite #1101, New York, NY 10016, USA
Tel.: +1 212 686 6655, Fax: +1 212 686 2422

UK: 88 Wood Street, London EC2V 7RS, UK
Tel.: +44 20 85281000, Fax: +44 20 85281001

AUSTRALIA: 9 Norberry Terrace, 177-199 Pacific Hwy, North Sydney, 2060, Australia
Tel.: +61 2 99542224, Fax: +61 2 99558112

INDIA: Bagmane Technology Park, Byrasandra Village, C.V. Raman Nagar, Bangalore 560 093, India
Tel.: +91 80 4004 0404, Fax: +91 80 4004 9999



0511