Agile & Accountable Methodology

A white paper

Author: Amit Unde, Lead Architect

Abstract

Agile software development methodologies are being advocated to provide faster response to the business community and to effectively manage the changing requirements of the industry. At the same time, these methodologies are at times criticized for being ‘non-accountable’ or "people dependent". The implementation of these methodologies, therefore, is often considered a challenging task, especially for software service providers who are developing software solutions offshore. This paper discusses the methodology that enables ‘agile and accountable’ software development with the onsite-offshore development model. It also explores some market-leading tools that support successful execution of the methodology, and provides guidelines for their effective implementation.

Table of Contents

Agile Development Methodologies

The principles of agile development methodology can be effectively summarized by the following manifesto, published by a group of software consultants and practitioners:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiations
- Responding to change over following a plan

Thus, agile development methodologies focus more on people and collaboration coupled with effectiveness and manoeuvrability, rather than formal, people-independent processes that advocate adhering to plans, formal sign-offs, checklists and change control. The nature of agile methodology is 'adaptive' and not 'predictive'. It accepts the fact that requirements will keep changing and so does not try to predict or subsequently document all the requirements. Instead, it also proposes developing 'working software' for requirements currently known, and then modifying it for changes in requirements as and when they occur.

Benefits

- Faster turn-around time, enabling quick response to the business community
- Flexibility in changing requirements, when deemed necessary
- More visibility and control to end users, due to frequent communication

Limitations

- Agile response is possible only when people experienced in technology and application functionality are involved
- Requires resources to work closely in one location or time-zone
- Tacit knowledge makes the software heavily people-dependent. It makes redeployment of resources difficult

Need for Onsite-Offshore Software Development

Onsite-offshore model

In an outsourcing engagement, the onsite-offshore hybrid model is the most popular software development model offered by service vendors to their clients. In this model, the outsourcing work is distributed between the onsite center (usually the client location) and service provider's offshore development center. Usually, 20-30% of the work is done at the onsite center that includes requirement gathering, planning and initial designing, co-ordination, issue resolution, deployment and support. The actual development and testing of the software is done at the offshore center, based on the requirements provided to them.

This model offers an array of benefits to customers like great cost savings, increased productivity (due to extended work cycle), increased resource bandwidth and flexible resource management. At the same time, it poses tough challenges mostly due to the distance between offshore and onshore centers. These include possibility of communication gaps, restriction on changes, lesser visibility and control on progress. Similarly, there are other issues like greater attrition of resources at offshore locations. Also, team members are usually reluctant to continue on the project in the same role for a longer time, which makes frequent rotation of resources absolutely necessary.

Process-oriented methodology

Service providers have attempted to deal with these problems by putting strong formal processes around the software development life cycle (SDLC). Most software service providers follow the processes as suggested by CMMi model.

These processes require creation of several artifacts such as comprehensive plans, requirement specifications, design models, test cases, bug reports etc. The processes demand review and formal sign-off of all documents before moving into the next phase of SDLC. A huge metrics collection is also required to determine productivity, schedule slippage, efforts/budget overrun, and defects data. These metrics are useful in bringing about improvements in the process.

Adapting new methodology

Process-oriented methodology provides many benefits in terms of knowledge management, quality assurance, quantitative measurements and continuous improvements. It reduces the dependence on certain people for the success of projects. However, these processes also have many limitations:

- Process overheads: These heavyweight processes impose overheads on the software development, which slows down the turn-around time for implementing change.
- Unpredictability of software requirements: This methodology relies on predicting the requirements, preparing a plan for implementation and then following a plan for development. However, more than often, it is very difficult to predict the requirements. Even if one does predict the requirements, the

business fundamentals change so rapidly that the requirements are bound to change just as quickly. The business demands software developers to accept the reality of the changing requirements and be prepared with a quicker response.

- Intangible nature of software: The software is intangible in nature. At times, it is very difficult to understand the value of software features, unless one uses actually uses it. Business users require working software so that they can evaluate features, provide feedback and evolve the desired requirements for the end product.

Due to these limitations, even though the project is successful as per the plan, end users may not be truly satisfied with the final product. They expect more visibility, better control and faster responses to their needs. This requires service providers to look at agile development methodologies and apply them to their situation. At the same time, one would also like to retain the benefits that traditional process-oriented methodology offers. This is quite a challenge as the philosophies behind both these methodologies are at loggerheads with each other — one demands reliance on people and informality, while the other stresses people-independence and formality. In the following sections, a methodology is suggested that attempts to combine the best of both these worlds.

Proposed Methodology

Objective

The objective of this methodology is to enable agile and accountable software development in the onsite-offshore model, by effective implementation of processes, tools and techniques in collaboration with business users.

Philosophy

The methodology is a combination of agile methodologies like scrum, as well as some other CMM good practices. It provides a process framework that one needs to suit one's situation. In some cases, one may opt for very formal processes like CMM, while a more informal approach may suit others. The core thought behind this methodology is that a 'one size fits all' approach does not work. Every project, every situation and every resource is different and one needs to adopt software development processes that factor in these project-specific elements.

The degree of formality would change based on 3 parameters, as described below and represented in Exhibit 1.

1. Project specifics

Look at the criticality of the project to the business, and also the complexity and size of the project and derive your processes. For example, for a critical project, you may have to do formal requirement sign-offs and avoid ad-hoc changes to the software. For a complex project, you may require formal and detailed low-level designing. For a big project, you may not want to spend too much time on detailed designing, but may come up with architecture or design blueprints and guidelines that you can circulate to all the developers.

2. Team working relationship

Look at how long you have been working with the customer and the past relationship. If you have formed trust relationship with the customer, you may want to reduce the formal nature in the process and make it more agile

3. Developer expertise

If you have more experienced resources, you may not want to detail the design for them. However, if you have less experienced resources, you may want to do a low-level design for them.

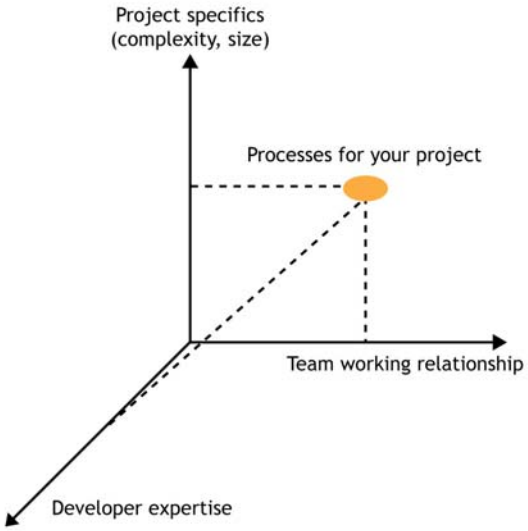

Exhibit : Not one size fits all

Thus, if you tune your processes as per your project, situations and resources, you can reduce the process overhead considerably. The derived methodology will be somewhere in between the CMM methodology and agile methodologies.

Process Framework

Exhibit 2 describes the process framework in detail.

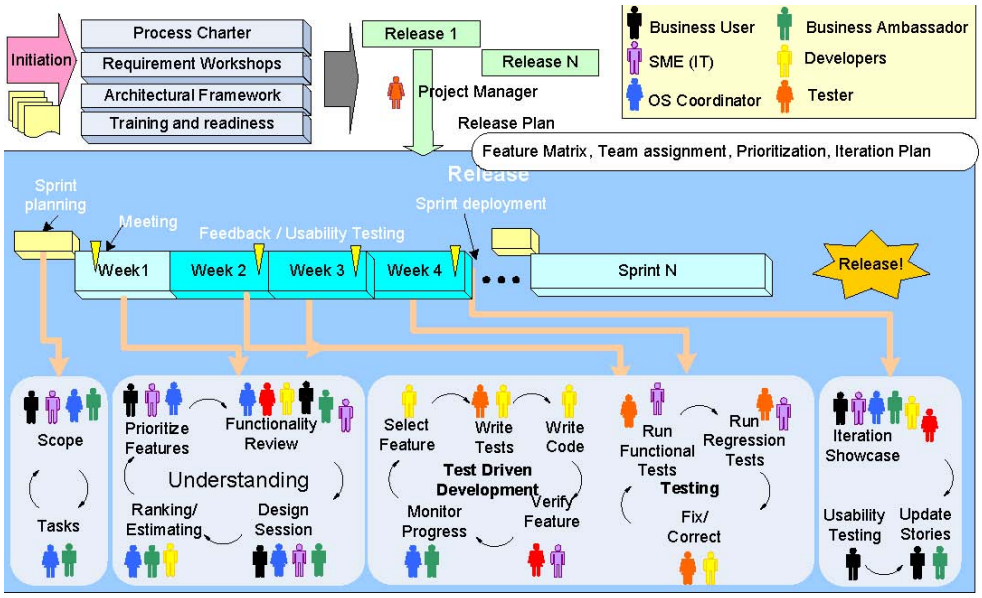

Exhibit 2: Process Framework

Initiation Phase

This phase is a pre-cursor to the detailed planning phase. This phase is more of a formal phase. It sets the foundation for the project. This foundation helps the later phases be more rapid and agile. It consists of the following activities:

- **Define process charter:** As explained in the previous section, derive your processes by considering project specifics, team working relationship and developer expertise. This will determine the degree of formality that you want to bring in to your processes.
- **Requirement workshops:** Conduct a detailed requirement workshop with business users and collect as many requirements as possible. Document these requirements in a document called 'Requirement Specifications'. Sometimes, while implementing innovative project concepts, it may not be possible for business users to come up with detailed requirements. In that case, define the initial hypothesis to start the work. The requirements will evolve during the first few implementation iterations. It is important to start the work with well thought-out requirements. While implementing agile methodology, it has been noticed that sometimes, teams do not spend enough time on brainstorming to arrive at the requirements at the start of project. This gives rise to many changes and rework later on. It makes the project complex and slows down the response to the changes. Thus, it actually works against the agile philosophy. This phase therefore, ensures that ample time is spent on requirements before starting the implementation. This process is similar to traditional (CMM-like) processes. The important difference is that requirements are not frozen or signed off. They are open for changes as may be deemed necessary later on.
- **Architecture workshops:** Conduct an architecture workshop for known requirements to define a base architectural blueprint and design guidelines. The base blueprint can be defined at the organization level and then tailored for each project when necessary. The blueprint defines architecture pattern (e.g. MVC), design decisions (e.g. 3-tier, Web Service interfaces, data layer etc.) and nomenclature (e.g. XXXManagerClass, etc.) that are to be followed. It provides guidelines for developers that enable them to write the code even without providing the detailed design. (It is assumed that the developers are experienced enough to understand the blueprint and code accordingly. If that is not the case, you will need to tune the process.)
- **Training and readiness:** Identify the training needs for the developers and complete the training before the start of the project. The training should include hands-on exercises. At the end of the training, the developers should be very comfortable with the chosen technology. Once the training is complete, carry out an internal certification to ensure that all the developers are ready for the project. The developers can not afford to learn on the job if they are expected to provide agile response to changes.

Planning Phase

This is a very important phase. The success of your project actually depends on how well you plan. Divide all the known requirements into smaller work units (or features). The rule of thumb is that each work unit should not require more than a week to implement for 2 to 3 developers (10 to 15 person days of efforts). Group all related work units into iterations called 'sprints'. Each sprint should be of 4 to 6 weeks size, with team size of approximately 6 to 8 resources. The criterion for defining a sprint is that we should be able to put each sprint package into production independently.

Smaller groups have better agility than larger groups, hence, sprint team size is restricted to 6 to 8 resources. Also, the small duration of sprint ensures frequent visibility to business users and opportunity for them to give feedback.

Now, create a release plan with pre-defined dates and group sprints under each release as per the priority. For example, you may create a release plan for quarterly release and then group 3-4 sprints in each release. The sprints can also be executed in parallel. Again, the general rule of thumb is to keep a leeway in release date for accommodating new requirements that may occur in each sprint.

Thus, after the planning phase, the work is divided into smaller sprints. Each sprint is time-boxed and also the production release date is fixed. Later on every time a new requirement of change is introduced, the scope of each sprint or release will be negotiated, but the release date to production will not be changed whatsoever.

Sprint Execution

Let's discuss team structure first, as shown in Exhibit 3. Details of the structure and associated roles are provided in Exhibit 4.

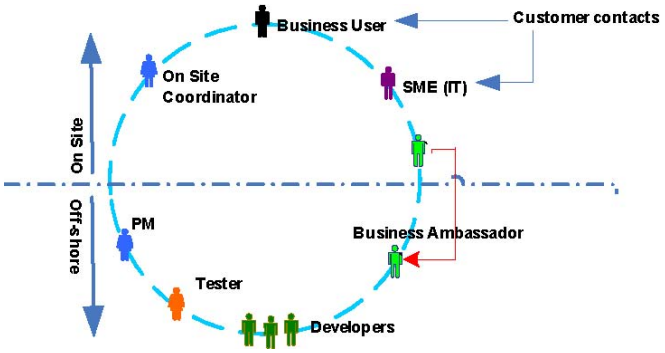

Exhibit 3: Sprint team structure

Role	Organization	Location	Responsibility
Business User	Customer	Onsite	Defining requirements Review and feedback Common across sprint
Subject Matter Expert – IT	Customer	Onsite	Point of contact for service provider's team Takes decisions on behalf of customer organization Technical review and feedback Common across multiple sprints
Project Manager	Service Provider	Offshore	Planning and tracking progress Status reporting Ensure people motivation Conflict resolutions Performance tracking Common across multiple sprints
Onsite coordinator	Service Provider	Onsite	Project issue resolutions Coordinating with business user for understanding requirements and changes Coordinating with SME-IT for technical decisions Common across multiple sprints
Business Ambassador	Service Provider	Offshore	Senior resource with domain background Can shadow customer business users during offshore working hours Helps the development team in functional understanding Works with testing team for verification and validation of the software product Is onsite during the initial phase in requirement gathering; Later moves back to offshore team Can be common across multiple sprints
Developer	Service Provider	Offshore	Dedicated for each sprint Develops code and unit tests Does impact analysis of changes by collaborating with colleagues Collaborates with customer point of contacts (with onsite coordinator) for scope/time change as appropriate, to accommodate higher priority changes
Tester	Service Provider	Offshore	Dedicated for each sprint Develops automated test cases by working with developers Continuous feature wise integration testing in parallel with development Regression testing after code freeze

Exhibit 4: Sprint team structure and responsibilities

Sprint Execution Phases

Phases for sprint execution are described below:

1. Spring planning (2-3 days)

This is a very short phase prior to sprint. Business ambassador and onsite coordinator touch base with customer business user and SME-IT to review the functionality planned for the sprint and confirm the priorities. If there are any changes required in the functionality, those changes are included in the planned list. The meetings for detailed discussion on requirements are scheduled as listed below.

2. Sprint understanding (1 week)

The entire sprint team participates in this phase. There are meetings planned with onsite and offshore team everyday. The work timings need to be shifted by mutual consent to ensure sufficient time overlap (say 3 hours) between the onsite and offshore teams.

During the meeting, onsite business user and IT SME discuss the functionality in detail and discuss the feature priorities. The business ambassador works with them to create many story board diagrams, workflows, screen wire frames, etc. The focus is on making sure that the developers and testers know the rationale behind the requirements.

After the meeting, the developers and testers work together on design for these requirements. They brainstorm on the requirements and Exhibit out the technical issues with implementation of the requirements, and also estimate the requirements if possible. The next day, they share the issues with the onsite team and discuss probable solutions. This continues for a week. After a week, everybody is pretty sure about the requirements and what is expected of them. The scope of the development is also clear. Depending upon the scope of the work, the team may together decide to drop some of the features from the sprint or add new features as required.

3. Test-driven development (2 to 3 weeks)

The developers and testers select each feature for development. Together they write the test case for the feature. The developer then writes the code for the features and does the unit testing. This is followed by the testers who verify the feature in development test environment by running the unit test case. The business ambassador and onsite coordinator help the team wherever necessary.

4. Regression testing (1 week)

After the development and unit testing of each feature is done, the code is moved to the testing environment. The testers run the regression test cases to make sure that all the sprint requirements are implemented. They report defects immediately to the developers. The developers fix the defects in development environment and each day the code is moved to testing environment at determined frequency.

As the functionality change is limited for a sprint, the testing complexity is controlled. Also, as the group is smaller and closely located, the testing efficiency is at highest.

5. Iteration showcase (1 week)

After the sprint regression testing, the sprint deliverable is showcased to customer business user and IT subject matter expert. The whole team participates in this phase. Again the work timings might need some adjusting to ensure overlapping between onsite and offshore teams. In these meetings, business users and SMET review the software work product and provide their feedback. The smaller changes are accommodated immediately. Other changes are included in the scope of next sprint.

6. Knowledge management (2-3 days)

At the end of each sprint, the knowledge management phase is introduced. This is to make sure that the documents regarding the sprint are up to date. Most of the documents are not created up front before the development, but are actually created later for future references. Documents required and details to be included are defined at the start in the process charter. Using a collaborative portal platform like Wiki is a good way to create such documents.

7. Parallel sprint execution

The sprints can be executed in parallel as well. It not only adds to the complexity of project management, but can really speed up the development. Parallel sprint execution is shown in Exhibit 5 below.

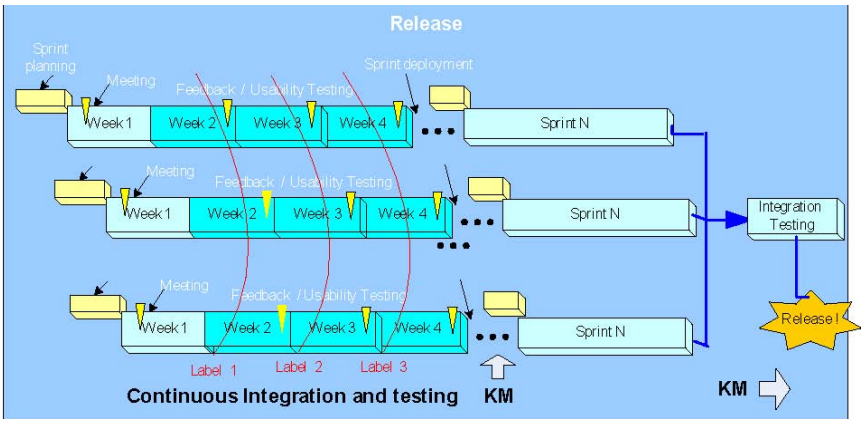

Exhibit 5: Parallel sprint execution

While planning parallel execution, care must be taken to create as independent a code base as possible. There could be some files shared across the sprints, but these should be kept to a minimum to avoid merging complexity later on.

To avoid last minute surprises, the code base should be merged at defined frequencies and tested in separate environments. A very strict configuration management supported by tools is required to manage the complexity. In case of parallel sprint execution, it requires a phase of integration testing before the release. This is to make sure that all the integration issues between the sprints are resolved.

Tool Support

Many tool vendors offer tools to support the agile methodology. Some of those tools (e.g. Microsoft VSTS) allow you to customize them to support the methodology specific to your organization. When evaluating tools to support agile methodology, the following parameters need to be considered:

- **Developer productivity:** The tool should provide an integrated environment for the developers. It means a one-stop-shop for all developer needs. The developers should be able to view the tasks created for them in the IDE, checkout/check-in code, run automated unit tests, create defect reports and even fill timesheet and update status in the plan. Such tools considerably improve the developer's productivity as the process overheads are reduced.
- **Support for collaboration tools:** The tool should provide integrated support for messaging, online team meetings and workflows. This helps developers across locations to easily collaborate with each other.
- **Strong configuration management / release management**: This is essential in case of parallel sprint executions. Some tools let you define the rules for stricter configuration management. For example, before checking in any code, you can put a rule to compile the entire code base. Only if the compilation is successful, the developer is allowed to check-in. Similarly, you can set rules to create daily builds and move the code automatically to testing environment etc. Some tools like Telelogic Synergy allow task-based configuration management. It allows you check-in and check-out code related to each task.
- **Project metrics collection:** Tools should support easy metrics collection and reporting of data for regular tracking. Microsoft VSTS provides the ability to conExhibit metrics parameters and reports.
- **Knowledge management:** Collaborative portals like Microsoft Share Point or Wiki can be used for knowledge management. In case of Wikis, any developer who has been granted access can change any page just by clicking the 'Edit page' button. It does not require any review or approval. You are basically trusting developers to make the right changes. The Wiki software usually maintains the vital versioning information that allows you to track the history of the documents, and restore previous version in case it is required. It is an ideal platform for trust based collaborative authorship of project documents.

Implementation Guidelines

This methodology puts a lot of trust and responsibility on each individual to make the project successful. It is not easy in a customer-service provider relationship to implement the collective ownership environment. There has to be trust established between both the parties before you can attempt this methodology. Here are some guidelines for implementing this methodology in your project.

- **Generate buy-in:** Educate customers and developers about the benefits and risks in implementing this methodology and for generating buy-in from all the parties. All those involved need to be enthused about implementing this methodology; otherwise it is unlikely to succeed.
- **Work out contractual terms:** It is challenging to workout contractual terms for agile development. Project development using time and material contract terms is probably the best option. However, customers usually prefer "Fixed price" project for better risk management and budget allocation. In that case, combination of these two costing models can be experimented with.
- **Set-up process framework and supporting tools:** Set up the process framework, tools and templates as described in this paper. Tailor it to what is-most suitable to your customer's and your organization.
- **Cultural assimilation:** Agile methodology demands more collaboration between onsite and offshore teams. Every developer is empowered to discuss, negotiate and take decisions. In case of the onsite-offshore model, people from two different cultures need to interact with each other. This requires training for both the parties to understand each other's culture and learn to collaborate with each other
- **Start small:** To begin with, start using this methodology with small, not so critical projects. The team will require time to get used to a new way of life. Use the project learning to tune the process framework.

Conclusion

Service providers adopt methodologies like RUP and CMM for onsite-offshore development. These heavyweight software methodologies provide benefits such as people independence because of better knowledge management, quantitative performance measurements and quality assurance. However, those methodologies resist change and provide sluggish responses to changes in requirements. Agile development methodologies, on the other hand, cut flab, welcome changes and provide rapid responses to changing business requirements. A combination of these methodologies can provide the best of both worlds to onsite-offshore software development. The methodology needs to support following features:

- **Evolutionary:** Be iterative with smaller iterations of 4 to 6 weeks executed by smaller teams
- **Collaborative:** Should ensure frequent feedback mechanism
- **Test driven development**: Should support continuous testing
- **Continuous integration:** Should ensure integrated code base
- **Changed mindset:** Should encourage empowerment, collective ownership and trust

The methodology can not be 'one-size-fits-all' for all projects. Project specifics, working relationship of teams and developer expertise, should be factored in while fine-tuning the methodology to suit the project.

Reference Material

1. Manifesto for Agile Software Development (http://agilemanifesto.org)
2. The New Methodology (http://www.martinfowler.com/articles/newMethodology.html)
3. Continuous integration: (http://www.martinfowler.com/articles/continuousintegration.com)
4. MSF for agile development
5. CMMi http://www.sei.cmu.edu/cmmi/general/
6. Copyright for this white paper belongs to QAI

About the author

Amit Unde is a Lead Architect at L&T Infotech. He has more than 10 years of experience in various software development disciplines such as programming, software architecting and strategic program management. He has five years of international experience and has successfully managed large IT projects executed with the onsite/offshore model. Amit has studied engineering at Pune University. He is a Microsoft Certified Solution Architect. He is also a PMA certified Qualified Project Management Professional (QPMP) and has undergone IT project management training in the USA, at a course conducted by ESI International and certified by George Washington University.

About L&T Infotech

L&T Infotech offers domain-specific IT services and solutions. We are sharply focused on a set of industries that include manufacturing, banking and financial services, insurance, energy and petrochemicals and telecom (product engineering services). We are part of Larsen & Toubro, a multi-billion dollar engineering and construction conglomerate. We leverage this unique business-to-IT connect to offer the winning edge to our clients.

For more information, visit us at www.lntinfotech.com or email us at info@lntinfotech.com.