Assignment 1

An analysis of Values & Principles for Agile & Lean

Sai Pranav Koyyada
saky19@student.bth.se
9904299311

September 2020

Contents

1 Introduction

Software process models have matured along with the magnified complexity of software to address practical problems. Over the past years, several software process models have emerged. Agile and Lean are two mainstream process models adapted widely by multiple organizations. This report is an analysis of values and principles for Agile Lean. Section 2 explains Agile, its Manifesto, values, and principles defined in the manifesto as its subsections. Section 3 talks about Lean, value in Lean, and its principles as subsections. Section 4 draws a comparison between the principles of Agile and Lean. Section 5 examines Scrum and Kanban methods in Agile and Lean ecosystems. And section 6 investigates practical problems while adopting Agile and Lean.

2 Agile

Agile is construction adapting to change, thus supporting to sustain the turbulent environment and uncertain demands. It is a retrospective to investigate the surroundings and impediments with a spur to define and adapt refinements. Agile is an umbrella for a set of frameworks and practices rooting on the values and principles of the Agile Manifesto. [1,2,3,4]

2.1 The Agile Manifesto

Before formally recognizing Agile, independent teams from diverse organizations developed methodologies resembling the functions of Agile to compliment respective environments. In 2001, 17 developers collaboratively drafted the manifesto of Agile. Agile Manifesto was a compilation of commonalities and differences of diverse approaches of software development unanimously agreed by the 17 independent developers. It holds 4 values and 12 principles to abide by for Agile software development with an Agile mindset. From grass-root adaptations, Agile turned mainstream with different organizations introducing Agile to their respective approaches.[1,2,3,4]

2.1.1 Agile Values

Following are the 4 Values of Agile in accordance with The Agile Manifesto.

1. **Individuals and interactions over processes and tools.** [1,3,5]

   Agile teams must be cross-functional to leverage control over the solution. Expectations from the system can be met with optimal resources by individuals from the appropriate background and collaborative effort among them. Process and tools only assist the living assets of the project. They can only enhance or minimize the resource expenses but not ship the complete solutions without comprehensive individuals with exemplary interactions.[5]

2. **Working software over comprehensive documentation.** [1,3,5]

   The individual perceptions of the solution can be comprehensively matched with working software. The demonstration visually translates complex proposals to the consumable workflow of features, thus making the solution evident and predictable. Documentation can be a secondary component in the Agile toolkit used for recording details in finer granularity and future recalls.[2,5]

3. **Customer collaboration over contract negotiation.** [1,3,5]

   Contracts are not a replacement for communication. Customers can entitle value for individual features of the solution. Without the collaboration of customer, requirements will constantly change with no definitive pause or value addition. Customers should be involved in the development throughout the project.[5]

4. **Responding to change over following a plan.** [1,3,5]

   Following a plan constructs a rigid project structure denying change. Changes must be adapted to leverage value into the project. Without change the software remains stagnate. It is critical to respect feedback from costumers and deliver their priorities over a predefined plan.[5]

2.1.2 Agile principles

Following are the 12 principles of Agile in accordance with The Agile Manifesto.

1. **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**[1,2,4]

   Delivering components persistently improves the perception and predictability of the solution. Customers appreciate getting value in the initial phases of development to review and understand the broader view of the solution. Staging the features early helps project teams assess critical feedback from customers and reflect upon the progress. Irrespective of the completion of the solution, it is always encouraged to ship even a minimal feature holding evident value early to the end-user.[2]

   Although the snag of delivering early is the scope of the solution shipped is minimal and far from complete. Thus, some stakeholders cannot evaluate the solution. Following the situation, Enterprises always barricade the release of the solution with minimum expected features. This chokes the flow of the development with delayed feedback from customers and reverting considerable progress if there is a major change to be implemented. This change management requires to initiate a new set of contract negotiations with the stakeholders. Under these circumstances, it is difficult to evolve the solution over time as the project team remains stuck in rinsing code for changes and repeating the same due to missed early feedback.[2]

The core value of Agile "Customer collaboration over contract negotiation" addresses this situation. Project teams must maintain a continuous flow of delivering valuable features to consume change optimally and support the evolution of the solution over time. Therefore the priority of every project should always be to reach the customer early with valuable features and maintain a continuous delivery cycle to promote changes.[2]

2. **Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.**[1,2,4]

   Change is a primary component in the Agile inventory. But change is not always welcomed due to the resources invested in the solution to make progress. It charges a mental toll on the project team and backs them from taking pride and ownership of the solution. Changes must be embraced as learning than a mistake. Mitigating changes early is critical. Riding into to late phases of a project with a major change overhead can toss the sanity of the team. Regardless, change has to be adapted taking its positive aftermath into account.[2]

3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**[1,2,4]

   Shorter time-boxed iterations facilitate optimal change control. Delivering functional features of the solution in shorter runs support capturing critical feedback from the customer. With sequential smaller builds, the scope of change can be restricted. This makes changes more consumable and controlled minimizing the possibility of major breakages late in solution completion. Working in shorter sprints accommodates reasonable work distribution thus reducing the overall effort and helps to maintain the morale of the team.[2]

4. **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.** [1,2,4]

   Propagation of thoughts and transfer of knowledge is effective through face-to-face communication. Documentation is a secondary tool for recording information in finer granularity and future recalls. Face-to-Face conversations inherit a sense of community. Project teams and stakeholders experience ownership by contributing their thoughts towards building the idea. Also, they can envision the solution in close cohesion with each other's perspective.[2]

5. **Business people and developers must work together daily throughout the project.**[1,2,4]

   Agile promotes collaboration with the Business people to understand the value proposition of the solution. Although in practice business people are not available throughout the project. Therefore, it is crucial to run time-boxed iterations delivering builds with functional value. The backlog with potentially significant value must be prioritized to ship value early and assess feedback from the Business people. Thus, with minimal time expense of both Business people and developers greater productivity can be achieved.[2]

6. **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**[1,2,4]

   Project teams must understand the value proposition of the solution. This induces pride and ownership for the solution, allowing team to contribute constructively towards improving. The morale of the team must be maintained by respecting and rewarding them for what they build. Without a value or respect for the solution and or poor rewards drive teams to lose motivation to build the solution. There should always be a trust and collaborative environment around the team members to promote knowledge transfer and quality of the software.[2]

7. **Working software is the primary measure of progress.**[1,2,4]

   Status reports are politically biased entities supporting chosen individuals. Consuming the context of the report also varies on the perception of individuals. With a working software, progress is evident. The solution communicates the status of the project without an explicit need to define or order textual reports.[2]

8. **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**

   Planning and estimating the resources and effort required to deliver the promised functional set of the solution in each iteration allows project teams to schedule the pace of the development. Following a defined plan to maintain sustained development pace thus helps in holding a reserved time to acknowledge late breakage or unforeseen issues without charging an extra mental or physical toll on the project team. Also, it offers a window to retrospect the sprint and defines the future roadmap for progress.[2]

9. **Continuous attention to technical excellence and good design enhances agility.**

   Scalability and maintainability is an inherent bonus from appropriate code design to fit the expectations of the solutions. Changes are more consumable provided the solution supports them. Therefore, continuous improvement of technical knowledge of the project team is critical to enable better coding practices and solution architecture to allow changes without choking the pace of development.[2]

10. **Simplicity—the art of maximizing the amount of work not done—is essential.**[1,2,4]

    Adding gold-platting to existing features can sometimes backfire. Keeping the deliverables optimal and restricted strictly to the backlogs maximizes throughput. Adding code on code with extra dependencies can only create a domino effect of cascading changes and breakages. Therefore estimating and prioritizing backlogs is critical to maintain a sustained pace and eliminate unwanted wastage.[2]

11. **The best architectures, requirements, and designs emerge from self-organizing teams.**

    Self-organizing teams build a collaborative ecosystem with equal responsibility share contributed for planning, inception, and implementation. With better control over the decisions and design of the solution, these teams, with their cross-functional capabilities, estimate the effort, decide the deliverables, and manage changes. The sense of ownership constructively reflects on the quality of the product. [2]

12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

    Agile teams adapt and learn to build better solutions with constant introspection. They run a retrospective post each iteration to understand the complications and impediments witness during the previous run and identify other underlying factors denying maximum potential delivery. They mitigate and change in all aspects by understanding the problems to improve and excel in coming iterations.[2]

2.2 Connection between values and principles defined in Agile Manifesto

Principles are extensions of values elaborating on the intricacies of adapting values in practice.

Value #1: **"Individuals and interactions over processes and tools"** is inherited by principle #4: **"Business people and developers must work together daily throughout the project"**, principle #6: **"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."**, principle #9: **"Continuous attention to technical excellence and good design enhances agility"**, principle #11: **"The best architectures, requirements, and designs emerge from self-organizing teams."**, and principle #12: **"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."**. Individuals are living assets of the project and thus, must interact through effective modes to facilitate retrospectives on the process, tools, and solutions directly contributing towards refining project behavior. Also, individuals collaboratively can define appropriate design to build robust solutions supporting change with optimal resource expense.

Value #2: **Working software over comprehensive documentation** is acknowledged by principle #1: **"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."**, principle #3: **"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale"** and principle #7: **"Working software is the primary measure of progress"**. Working software propagates the idea of a project with visual feature workflow compensating for the knowledge gap of different stakeholders. Delivering this artifact early and continuously captures feedback to assert the progress. Documentation takes a backseat to record information with finer granularity and future reference.

Value #3: **Customer collaboration over contract negotiation** is referenced by principle #1: **"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."**, and principle #4: **"Business people and developers must work together daily throughout the project"**. Customers dictate the value of features. Customer collaboration is critical for feedback to assert the progress and direction of development. Without customer feedback and the project loses its value thus, demoralizing the team and deteriorating the quality of the solution delivered.

Value #4 **Responding to change over following a plan** is rooted by principle #2 **"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage"**. Change is inevitable. Changes must be embraced as learning than mistakes. Change has to be adapted taking its scale and positive aftermath into account. Restricting to a pre-defined plan will only repel changes thus, deteriorating the quality and value of the solution.

Principle #8 **"Agile processes promote sustainable development.**

The sponsors, developers, and users should be able to maintain a constant pace indefinitely" is an adaptation of all the values since it acknowledges the development and change management from customer feedback with sustained pace thus, drawing on individuals and interactions in the team and distributing each iteration to deliver working software.

3 Lean

Lean production practices introduced by Toyota Production Systems translates as Lean software development as a synthesis of a system of practices, principles, and philosophy for building software systems for customer use. The primary focus of Lean software development is to deliver value than restructure the software development process.[6] Therefore, Lean as a principle supports Agile organizations following the independent development process with the pro-lean subculture to add value for customers.[7]

3.1 Lean Value

Value in Lean is an element the customer is willing to pay for. Value for a customer is diverse and based on choice. But in commonality, entity mediating or assisting in practical problem-solving with reduced steps can be referred to as having value for the customer. Besides, customers claim value in entities functioning as desired and offer an incentive in the aspect and at the time they wish for. The customer expects the entity to deliver the highest quality function at a justifiable price.[7,8]

Identifying value is critical for customer satisfaction and sustainability. Customer collaboration is vital for asserting progress through feedback. Lean demands amplified learning to understand the requirements and improve the process for better value addition. Delivering features early offer insight about customer opinion and value helping project teams identity the value function addressing customer pain point. The Quality Function Deployment(QFD) method assists in capturing customer requirements and translating them into system specification value addition.[9]

Value Stream Mapping(VSM) will facilitate a defined pipeline to stream changes and add value with accurate estimations and minimized cost to the customer. It also adds the capability to design functions compatible with the stream to minimize or eliminate wastage completely. With a steady stream of value, customers can collaborate at the greater scope and refine value continuously supporting project sustainability.[10]

In summary, value is intangible relation addressing customer pain in exchange for trust and effort. Lean focuses on identifying and delivering value for customer satisfaction and organization sustainability.

3.2 Lean principles

Following are the 7 principles of Lean software development:

1. **Optimize the Whole** [7,8]

   Lean is built on understanding the intricacies of the ecosystem the software is residing and mediating for. Customer value definition is critical given that a software system rarely holds standalone value without a greater context of the problem it is resolving for an individual. Optimizing the system to cohesively adjust to its problem environment delivers immense value to the customer. A value stream encapsulating the thorough use case of the system must be set up to investigate feature leaks and estimation errors with local measures alongside reducing the expense of crossing boundaries within and outside the organization.[8]

2. **Eliminate Waste** [7,8]

   Lean classify an entity with no contribution towards value or knowledge addition as waste. Waste in the context of software development points to unwanted features, partially done work, handovers, and multitasking. Defects are direct consequences of waste in large codebases, therefore restricting progress and value addition. Writing less code and delivering nothing but value is a step towards eliminating waste. It is the focal point of Lean software development to improve the quality of the system shipped to the customer.[8]

3. **Build Quality In** [7,8]

   Lean promotes top-down programming to support feature integration on the fly without waiting time. The system must be designed to accept integration irrespective of its progress. Holding smaller components for integration until core system completion can invite late breakages. Building quality into the system during the development is crucial then to push it for the later phases and compromising due to schedule and budget overheads. Components must synchronize to function upon integration with minimal or no refactoring. Also, integration must be automated to assist development emphasis then operations management.[8]

4. **Amplify Learning** [7,8]

   Software systems are knowledge translation to mediate or assist problem-solving in its host ecosystem. Knowledge must be constantly acquired to understand the dynamics of the environment and appreciate the value proposition of the customer. Experimenting and collaborating are the two primary resources for knowledge acquisition transfer. Extracted knowledge must contribute to minimize the cost and improve the quality and value of the software system.[8]

5. **Defer commitment**

   Development can commence without complete specifications. The ability to adapt and change must be harnessed to minimize cost and leverage value stream for customers. Critical and expensive-to-change decisions must be delayed. Explore alternatives and assess possibilities before confining to a plan, thus offering flexibility to experiment and learn. There should options available to fall back upon failure.[8]

6. **Deliver Fast** [7,8]

   The software should be a pipeline with a steady flow of changes for value addition. The system should be broken into smaller batches for continuous and steady delivery depending on the capability to work. Delivering early asserts progress with critical feedback from customers, thus adapting changes in consumable chunks and promoting value over negotiating changes and choking the development flow.[8]

7. **Engage/Respect Everyone** [7,8]

   Cross-functional teams should be promoted to plan and control the development process, thus inducing a sense of ownership for the system, critical for value addition. Engaging everyone facilitates knowledge transfer and quality refinement benefiting the software system. Also, it builds a community where individuals respect and trust peers. Decision-making should seep to the lowest level of hierarchy to support a seamless value stream. [8]

4 Comparing Agile and Lean principles

Agile and Lean principles demonstrate strong cohesion. Lean principles can help refine the process of development with reduced waste and optimizing as a whole. Agile principles pledge adaptability and inviting change. [7]

Agile is a product-focused methodology, while Lean improves the process. Thus, Lean can be a subculture for Agile organization helping them minimize cost and deliver value while adapting and changing by the feedback. Both Agile and Lean aim to deliver features to the customer early. Although, Lean pacing is managed and fluctuated at any given time by team speed and development flow. On the flip side, Agile emphasizes small batches with fixed deadlines to deliver features. Therefore, Agile offers a steady flow, whereas Lean is responsive and dynamic. Agile collaborates with customers via interactions. Lean puts the customer first by eliminating waste. It delivers customer value by shipping only relevant features. Agile is more structured with systematic reviews, structured meetings, and defined roles. In contrast, Lean builds a community with no specific roles and high peer trust. [2,7]

In summary, Agile and Lean are very same yet very different. Software products can benefit by adopting a Lean subculture under an Agile ecosystem to optimize resources and improve the quality of service and value delivered. [7]

5 Scrum and Kanban

Scrum and Kanban support Agile and Lean principles. Scrum commits to ship value in defined intervals with structured learning loops and release cycles. Kanban ships value when it is ready with no rigid structure applied to the development process.[11]

Scrum and Kanban believe in delivering a working system against the document to evaluate progress. They follow an iterative approach to break the software into consumable chunks and release them to customers following their respective release methodologies. Customer collaboration is a staple function for both methods. Feedback is analyzed and reflected on to refine the value proposition. Changes are welcomed in a different format. While scrum control changes at the end of each sprint, Kanban hosts them at any given time. Changes can be translated to specifications and added to the current iteration depending on the size and priority in Kanban. Scrum focuses on amplified learning and Kanban respects everyone to define a peer trust ecosystem. It also hones build quality in value of lean. Although both of them strive to deliver fast, Kanban can fluctuate with its release cycles, whereas Scrum offers a steady value stream. Therefore Scrum has a sustained delivery pace as defined by Agile principles when Kanban shies away from this. None the less both adhere to minimize waste. They also are self-organizing teams to devise and control decisions on the project. There is a focus on individual and their interactions rather than the process or tools in these two methods.[11]

Scrum and Kanban are two distinct offering respecting Agile and Lean principles fit for different teams and different projects. Where scrum is organised and steady, Kanban wields continuous adaption as focus.[11]

6 Practical standpoint

A portion of Agile adapters has lost sight of Agile values and principles. Certain practices are replaced with fixed methods deviating from the Agile mindset.[4] The reason may root in practical complications in implementing the Agile methods.

For instance, change is not always appreciated in the corporate crunch.

Changes are treated as mistakes and developers are held responsible for resource expenses. It tosses the team into dismay with no respect or value for what they build. With an effort to control the resource expense and improve employee performance, organizations implement incentive game systems to complement their work function. Developers are appreciated for discovering and adding features while testers are applauded for finding more bugs. These practices deteriorate the prime value of Agile to choose individuals and interactions over processes and tools. They create a split and antagonizing ecosystem between developers and tester with developers hoarding code with additional components with no proven value addition and testers turn picky by pinpointing possibly everything. The software quality falls with unwanted codes and poor reporting.[2]

The above scenario can also be extended to the violation of Lean thinking, values, and principles. Peer trust is broken in such an ecosystem with poor reporting by testers. Instead of eliminating waste, developers are creating it just for pity bonus. There is no respect for any individual in the team. The only thing that exists is a competition to gain personal benefit with software as the medium. The care and value of the system are lost.

With the falling quality, the system rejects changes and denies building quality in thus dying in a competitive market. Organizations have to understand Agile and Lean as attitudes for building better value and trust for their products and brand. Compromising with this attitude can only drive the organization's death since the principles of Agile and Lean are built to complement and function in cohesion with each other. Failing to follow any principle alone can cause a complete structural collapse.[2,4]

References

[1] Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick (2001). "Manifesto for Agile Software Development". https://agilemanifesto.org/. Accessed on 26/09/20.

[2] Stellman, Andrew, and Jennifer Greene. The Agile Principles, O'Reilly Media, Inc, 2018.

[3] Stellman, Andrew, and Jennifer Greene. What is Agile?, O'Reilly Media, Inc, 2017.

[4] McDonald, Kent. "Agile 101". Agile Alliance. Accessed 26/09/20.

[5] "Examining the Agile Manifesto". Ambysoft Inc. Accessed 26/09/20.

[6] R.N. Charette, "Challenging the Fundamental Notions of Software Development", white paper IT Metrics and Productivity Inst., 2003.

[7] Poppendieck, Mary, and Tom Poppendieck. Lean software development: an agile toolkit. Addison-Wesley, 2003.

[8] M. Poppendieck and M. A. Cusumano, ”Lean Software Development: A Tutorial,” in IEEE Software, vol. 29, no. 5, pp. 26-32, Sept.-Oct. 2012, doi: 10.1109/MS.2012.107.

[9] Akao, Y.. “Quality Function Deployment : Integrating Customer Requirements into Product Design.” (1990).

[10] Manos, Tony (June 2006). ”Value Stream Mapping—an Introduction” (PDF). Quality Progress. American Society for Quality. p. 64 – via University of Washington.

[11] Max Rehkopf. ”Kanban vs Scrum”. Atlassian Agile Coach. Accessed on 26/09/20