SATORI CONSULTING

LEAN AGILE POCKET GUIDE

Software Product
Development Methodology
Reference Guide

PURPOSE

This pocket guide serves as a reference to a family of lean agile software development methodologies and tools that we find most valuable for those who lead software product development projects, programs and improvement efforts.

Objectives

Using the practices in this guide enables technology development organizations to:

- Reduce time to market for new products, systems, processes, and technologies.
- Improve customer and end-user satisfaction with project outcomes
- Reduce the risk of failure, wasted investment, and runaway projects
- Improve productivity, quality, and teamwork
- Continuously improve delivery from one project, phase, or iteration to the next.

CONTENTS

This guide presents the principles that underlie and unite the agile and lean approaches to software development then moves to an explanation of agile with Scrum and kanban. Often development teams select a subset of the agile practices and create their own hybrid methodology based on some combination of these practices and traditional “plan-driven” or “waterfall” approaches.

ORIGINS OF AGILE

Beginning in the mid-1990s, a number of consultants independently created and evolved what later came to be known as agile software development methodologies.

Agile methodologies and practices emerged as an attempt to more formally and explicitly embrace higher rates of change in software requirements and customer expectations. Agile methodologies include Adaptive Software Development, Crystal, Dynamic Systems Development Method, Extreme Programming (XP), Feature-Driven Development (FDD), and Scrum.

Although there is no one agile "methodology," scrum has become the standard for coordinating the activities of agile project teams. The term was coined by Takeuchi and Nonaka to describe the "hyperproductive" product development practices they observed in Japanese and U.S. companies.

AGILE VALUES

The agile manifesto offers an important starting point for an organization that is consider adopting Agile practices. While the methods, techniques, and terminology have evolved since 2001, the core values of Agile have not. The Manifesto for Agile Software Development outlines a set of values and principles.

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value the following:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan,
  That is, while there is value in the items on the right, we value the items on the left more."

AGILE PRINCIPLES

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development.
- Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work that does not get done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

LEAN PRINCIPLES

Lean principles can be applied to projects of any size and scope, and are a prerequisite for scaling agile in large complex enterprises. Lean accounts for work that occurs both upstream and downstream of the software engineering process, while agile/scrum is focused primarily on the work of developers.

Value, Value Stream, Flow, Perfection

Lean principles include the following:

1. Value is specified from the standpoint of the end customer.
2. A value stream describes each step in processes and categorizes them with regard to the value added (e.g., value adding, necessary non-value adding and non-value adding steps).
3. Flow organizes processes so products move smoothly through the value-creating steps.
4. Pull involves each customer calling output from the previous step, on demand.
5. Perfection entails continuous improvement of processes for meeting customer needs with zero defects.

SCRUM 101

Agile software development is a team-based approach to satisfying the customer through frequent iterations of working software, each of which provides incremental benefit to the business. Scrum refers to the strategy that rugby teams use to get out-of-play balls back into play by passing the ball within a team, moving as a unit incrementally up the field. Agile with scrum is an iterative and incremental approach with the following characteristics:

- Each iteration or "sprint" is a self-contained, mini-project, with activities that span requirements analysis, design, implementation, test, and customer acceptance.
- Each iteration builds towards a new "release" (which may be only an internal release) that integrates the software developed across the team.
- Agile methods incorporate feedback obtained before and after iterations from both customers and the business.
- Frequent deadlines reduce the variance of a software process and can increase predictability and efficiency.
- The predetermined iteration length serves as a timebox.
- The scope of each iteration is chosen to fill the time allowed.
- Rather than increase the iteration length to fit the chosen scope, the scope is reduced to fit the iteration length.
- Iteration lengths vary between 1 and 4 weeks, and intentionally do not exceed 30 days.
- Research has shown that shorter iterations have lower risk, higher productivity and higher success rates.

USER STORIES

A user story describes functionality that will be valuable to the customer or user of a system or software. They are used for planning software development work associated with a product roadmap. Thus, stories:

- Comprise a narrative description of the goal and task for a specific user role in the form of "As a [user role], I would like to [action] so that I can [goal]".

- Include acceptance criteria for testing the code results to ensure they are valid and meet the expectations of the user.

Key Characteristics (INVEST):

- ***I****ndependent* of one another to enable their prioritization with as few dependencies on other stories as possible.
- ***N****egotiated* through verbal communication and can be modified and enhanced over time.
- ***V****aluable* to the customer or user
- ***E****stimatable* by software developers in terms of time and effort
- ***S****mall* so that each can be described on paper the size of an index card.
- ***T****estable* with criteria that enable verification that the coding is "done".

MORE ABOUT STORIES

- User Stories are not use cases or detailed requirements, as the intent is to shift emphasis towards verbal communication between developers and customers or their proxies and move away from requirements techniques that rely entirely on written documents.
- They can be split into smaller stories as their priority becomes higher and development work is likely to commence.
- They are easily understood by non-technical people
- They do not include non-functional / technical requirements, which should be labeled “constraints”.
- Several stories may be related to one another by a larger story called and “epic”.
- A given software release can be described in terms of related stories, epics or themes.
- Bug fixes should also be described as stories and included in the release plan based on their priority relative to other stories.

RELEASE PLANNING

A series of software releases over time can be expressed through the development of a product roadmap. Planning for a typical release cycle allows the organization to manage the phases of software product evolution by envisioning the application's key features, gathering information about likely cost and duration, reserving resources, and planning for the integration of major system components.

Typical Artifacts

- Business case
- Rough order of magnitude estimate of time and cost
- Customer and business requirements
- Vision document
- Project charter
- Preliminary stories

Depending on the project, this stage may include preliminary drafts of:

- Software development plan
- Use-case model
- Software architecture document

THE PRODUCT BACKLOG

The product backlog is the full set of desired features and functions expressed as a list of stories that the team has yet to deliver. It is allowed to grow and change as more is learned about requirements and customer needs.

The product owner is responsible for sorting the stories in priority order based on value to the customer, value to shareholders, degree of operational risk if not done, or some combination.

Sizes and Priorities

The product backlog consists of:

- Stories in priority order from highest to lowest
- Estimated size of each story relative to others based on an evaluation by the development team

When the product owner identifies a subset of stories that describes the next increment of new application features to be developed, that subset of stories from the product backlog becomes a release backlog.

When a team identifies the handful of stories that they will convert into working code during the next sprint, that small set is considered the story commitment.

PRODUCT OWNER

The product owner provides requirements for each feature, function or story in the product backlog and prioritizes the order in which they should be addressed. The product owner is also the person who will accept or reject the team's functional deliverables at the end of each sprint and release. Close collaboration with the business through the role of the product owner is key to Scrum's ability to be responsive to the business and its customers.

Being a product owner requires steady involvement with the development team, but the role does not consume 100% of their time. Participation is naturally higher during release and sprint planning; yet their role is to answer questions that arise over the course of each sprint.

Unlike most waterfall projects, developers on a Scrum project do not answer to a project manager, but instead to the product owner.

The product owner directly represents the parties providing the funds to maintain the project and the team, and is focused on maximizing the value of development work.

By controlling the project's backlog, the product owner decides what will be built, in what order, and when it will be pushed into production, subject only to technical constraints that the developers articulate.

SCRUM MASTER

The Scrum Master is responsible for monitoring and improving the team's performance, including its work processes and communication practices.

They are skilled facilitators who intervene when the team moves off track. Their role is not to dictate but to foster self-organization.

The scrum master is often responsible for the following:

- Facilitation of planning and daily scrum meetings
- Capturing and removing impediments identified by the team
- Tracking the team's productivity as measured by velocity
- Leading team retrospectives at regular intervals to facilitate continuous improvement.

SPRINT PLANNING

The Sprint Planning meeting consists of two segments:

1. Selecting stories for the upcoming iteration
2. Preparing the sprint backlog.

Prior to the meeting, the Product Owner prepares the Product Backlog. The Scrum Master schedules and facilitates.

First Meeting

The goal of the first segment of the meeting is for the team to select the product backlog items (stories) that it believes it can turn into an increment of potentially "releasable" product functionality. The result is the team's Sprint Commitment

Second Meeting

The goal of the second segment of the meeting is for the team to plan the work involved to accomplish the backlog items they've committed to. The result is a series of tasks that need to be accomplished over the course of the sprint.

The result of the sprint planning meeting is the Sprint Backlog.

SPRINT BACKLOG

The Sprint Backlog consists of the following:

- Tasks required to complete the product backlog items selected for the next sprint
- Task estimates
- Task assignments

The sprint backlog may also contain tasks that are not yet assigned.

Task progress is tracked and made visible to the team at all times using a task board or Kanban board.

Tasks may be “Done”, “In Progress” or “Waiting”.

Velocity is not measured by the number and/or size of the tasks completed; it is measured by the size of the user stories that have been developed over the course of the sprint.

DAILY SCRUM

The Daily Scrum is a short 15-minute meeting held everyday with all members attending, including the Scrum Master, Product Owner, Testers, and Developers.

The Scrum Master facilitates a round-robin update from each member.

Three Questions

Each team member answers the following three questions:

1. What did you do yesterday?
2. What will you do today?
3. What obstacles or impediments are in your way?

The Scrum Master notes all impediments identified.

A fundamental rule of the daily scrum is that there should be no problem solving.

Problem solving occurs after the meeting with those who need to be involved.

VELOCITY

The estimation of the level of effort using the abstract of “story points” or “t-shirt sizes” enables a software development team to measure its overall velocity each iteration.

Velocity is the sum of the sizes of each backlog item completed in a given sprint.

Measure of Team Output

The measure of velocity helps determine how much work the team tends to complete in a given sprint so that it can plan the appropriate scope to take on for the next sprint and/or release.

Velocity is plotted for each sprint as a time series to visualize the team’s progress over the course of each release.

Average velocity is used to project how many sprints will be required to complete the remaining items in the release backlog in the form of a Burn-down Chart.

BURNDOWN CHART

The Burn-Down chart is used in daily meetings to visualize the overall progress of the team and is a plot of the effort required to complete the remaining items in the release backlog.

The source of the data is the product or release backlog with size estimates for each story that remains to be completed in upcoming iterations.

Average team velocity can be used to project which stories are likely to be completed by the end of a given time period.

RETROSPECTIVE

The retrospective is a meeting attended by all team members to reflect on the last sprint. The product owner is included in this retrospective process as well as developers, for the team needs to reflect upon its effectiveness from both the customer's and the producer's point of view.

Reflection on Performance

The team discusses the following:

1. Did we achieve what we intended to achieve?
2. What worked well that we should carry forward?
3. What needs to be improved in the next sprint?

Skilled Facilitation

The session is led by the Scrum Master or another skilled facilitator who can:

- Encourage active participation from all team members
- Focus the group on problem-solving rather than blame
- Generate commitment from team members to identify and implement improvement action items.

KANBAN BOARD

The term Kanban is derived from the Japanese for “visual card” and is a component of Toyota’s Lean production process.

Visualization of the Work

A Kanban board consists of the following:

- Goals column on the far left that describes the team’s goals, to provide a reminder of the overall purpose of the project.

- Stories” column contains all of the stories that will be worked on, and a story remains in this column until it moves through the process toward completion.

- Each subsequent column represents the work steps necessary to get to the final state. These columns are defined by the team and could include “UI design,” “Development,” “Validation.”

- The top part of each column is for stories in progress for that step, the bottom is for stories that are complete for that step but have not yet moved to the next step.

- The final column holds the stories that are complete and of shipping quality.

VALUE STREAM MAPPING

Mapping the value stream is a way to start discovering the inefficiencies and impediments in the product development process.

From Concept to Cash

It is a depiction of all the steps required for a customer request or requirement to go from “concept to cash”.

The purpose of the Value Stream Map is to identify opportunities for reducing cycle time by removing or minimizing non value-added work.

Non-Value Added Work

Non Value-Added Work (waste) includes bottlenecks that create delays or add to calendar time and don’t add value to the customer and consume resources.

Value-Added Work is defined as the discovery, creation and transformation of knowledge into working code, features and systems that customers use to achieve their goals.

AGILE AND USER EXPERIENCE DESIGN (UX)

While Agile methods mainly describe activities addressing code creation, UX design methods describe activities for designing the product's interaction with a user. UX design methods allow software development teams to create software that is usable for the user.

Evaluating UX Designs

The phrase 'gap analysis' is sometimes used by developers to refer to an activity which involves inspecting the designs handed over by the UX designers, comparing it to the software already implemented and identifying mismatches between them.

Prioritizing Stories Based on UX Designs

Once the UX designs are delivered to developers and the developers feel they have enough information regarding the design to implement it as working software, the implementation work must be added to the Product Backlog. The UX designers are approached for design direction when unanticipated implementation issues arise..