THE LEANBAN PRIMER

Lean Software Development at the Team Level

Al Shalloway
James R Trott

Net Objectives
ESSENTIALS

The Leanban Primer: Lean Software Development at the Team Level
Al Shalloway and James R Trott

Net Objectives Press, a division of Net Objectives
1037 NE 65th Street Suite 362
Seattle WA 98115-6655
Toll Free: 1-888-LEAN-244

Find us on the Web at: *www.netobjectives.com*
To report errors, please send a note to *info@netobjectives.com*

The Leanban Primer is a condensed version of materials offered by Net Objectives. It is intended to support those who have already received training. It is appropriate for any role on a Lean-Agile team.

Notice of Rights

Notice of Liabilities

The information in this book is distributed on an “As Is” basis without warranty. While every precaution has been taken in the preparation of this book, neither the authors nor Net Objectives shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer or hardware products described in it.

ISBN 978-0-9713630-3-8

10 9 8 7 6 5 4 3 2 1
Printed in the United States of America

First Edition

Contents

Part III: Roles and the Team 35

Part V: Quality 133

Preface

The Leanban Primer collects in one resource the good practices we have learned and observed as we have trained thousands of teams in Lean and Agile software development, including SAFe®, Scrum, kanban, and XP.

This is a "primer." It is designed to supplement training, such as the courses described at *www.netobjectives.com/training*. It offers guidance through concise descriptions and checklists and visuals rather than trying to instruct you in essential concepts. It is organized to help you find what you need when you need it, following the normal course of a project.

This primer is intended to support the Leanban team (business analysts and developers and testers) and the Team Agility Master who work together to create a product. It addresses everything they will be doing in their work. Certainly, there is more to Lean-Agile than this: both upstream (in product planning) and downstream (in release and support) as well as managing the value stream. We chose to focus on the team level to keep the primer at a reasonable and useful size.

Leanban covers many principles and practices. There are several flavors of Agile methods in the marketplace, Scrum being the most popular at the moment. To make this primer widely useful, we chose to use the following terminology. The techniques are similar even if the words are different. Substitute as you see fit.

- **Iteration.** The time-boxed period during which teams focus on producing demonstrable deliverable. Scrum calls this a "sprint."
- **Team Agility Master.** The role responsible for the process and health of the team. In Scrum and SAFe®, this is known as the Scrum Master although the role has more responsibilites than what is common with Scrum. In kanban and other Agile methods, it may be known a project manager.
- **Product Vision.** The short statement of the vision that is driving the product work, expressed in business and customer value terms. This is often known as the project charter.
- **Populate.** When someone adds an item of work to the backlog of work to be done by the team, we say they "populate the backlog" or "place the item onto the backlog." Some methods call this "loading the backlog" but that phrase is not widely used.

Continuous Improvement

Quality improvement is central to Lean and Agile. This is true in this primer as well. As we learn more, the primer will improve and expand. We will continue to improve and expand. As you find areas for improvement or correction, please send us an e-mail. Our contact information is at *www.netobjectives.com*.

Acknowledgements

Many people have contributed significantly to this primer. In particular, we wish to acknowledge Alan Chedalawada and our colleagues at Net Objectives who are some of the best thinkers we have found in this area. Andrea Bain has been so helpful with design. We wish to acknowledge Curt Hibbs who has been tireless in proofing and contributing valuable ideas.

The concepts in this primer build on the shoulders of giants in the industry. These concepts represent the practical combination of Lean, Agile, Scrum, XP and technical principles and practices applied to software development. Some of the concepts and methods present have been created by Net Objectives (especially in the area of release planning) and some represent the guidance of others. Space does not allow us to acknowledge everyone, but here are some of the most important in this field:

Christopher Alexander
Jane Cleland-Huang
Mark Denne
Daniel Jones
Corey Ladas
David Mann
Mary and Tom Poppendieck
Don Reinertsen
Peter Scholtes
David Womack

Introduction

This primer is designed to reinforce the Lean-Agile thinking that you are developing as you use Scrum, kanban, XP, or other Agile team methods to develop software products. To get the most use of this primer, you and your team should have already had Leanban or Lean-Agile training by Net Objectives or other competent trainers.

Lean-Agile is driven by "Business value." The goal is to meet the customer's value and quality objectives and to deliver as much value to the customer as the customer can consume, as quickly as possible, in the most efficient manner possible, in a sustainable way.

The next section offers a brief description of essential principles and practices. Following that are sections on roles, planning, analysis, and estimation, a review of quality and continuous process improvement techniques, and some notes on special cases.

This primer is designed for teams who want to use Leanban. Team members will find most of the information in this primer to be useful as they do their daily work.

Lean-Agile involves more than team practices and involves roles in addition to the team; for example, Product Manager, the Business owner, the Team Agility Master, and management. These roles interact with the team but have their own set of practices and techniques they use. The people filling these roles will find some of this primer useful to their daily work, but they are not the primary audience.

Part I: Understanding Leanban

Leanban is a team-level offering that makes higher level Lean-Agile tenets actionable on a day-by-day basis. It involves a number of concepts that everyone must learn. This part describes these concepts.

The Leanban Advantage

Leanban provides a consistent approach to implementing the Minimum Business Increments that have been selected for development.

Here are some of the important advantages of Leanban.

- **Leanban is business driven.** All teams must focus on delivering value as defined and prioritized by the Business. It increases quality, reliability, and velocity
- **Leanban suggests core practices.** Teams get started quickly by following a core set of practices and then Lean-Thinking to adjust practices and add new ones based on their needs.
- **Leanban is tailored to each team's situation.** Each team's situation is unique. Leanban considers the team's current situation, including whether they are cross-functional, require iterations, and how disciplined they are.
- **Leanban provides a consistent approach across the organization.** This enables individuals to move around easily, facilitates learning across teams, and improves management understanding.
- **Leanban takes a systems-approach.** Teams deliver business value by working in concert; they must not simply focus on their own work. Leanban provides the mindset for coordination across teams.
- **Leanban provides teams a way to evolve to better practices as they learn.** Leanban provides guidance to teams to evolve as practices become inappropriate. It promotes positive change based on achieving the desired outcomes of a practice.
- **Leanban is based on proven principles.** Building on good practices improves professionalism and avoids dogma.
- **Leanban facilitates change.** Leanban supports change at a sustainable pace guided by Lean practices. It is important to determine what degree of change is appropriate without forcing change or avoiding change.

What is Leanban?

Scrum and eXtreme Programming (XP) were the first generation of Agile approaches. Kanban and the Kanban Method were the second. Each of these are consistent with subsets of Lean-Thinking; each emphasizes different Lean principles and each manifests Lean to different degrees.

Leanban is the third generation approach to Lean-Agile at the team-level. It makes higher level Lean-Agile principles actionable on a day-to-day basis. It differs from its predecessors in that from the beginning, it explicitly manifests Lean principles while also incorporating proven Agile practices. Leanban is not a mere hybrid of Scrum and kanban. It springs directly from Lean principles and it uses Scrum and kanban and other approaches as they help to manifest Lean. Leanban encapsulates Agile principles and practices with Lean principles and practices.

Leanban is adaptive. It solves the important dilemma of needing a consistent approach while enabling each teams to tailor their process to fit their contexts. It uses Lean-Science as an umbrella and provides a core set of practices that will work for virtually all teams. Leanban provides guidance to the teams on how to tailor other practices as required by their own, unique, situation. Because Leanban is based on a combination of principles and practices, it includes advice on how to adopt new practices when current ones become less than optimal.

At the team level, Lean recommends these practices:

- Improve flow across the entire value stream.
- Collaborate to the greatest extent possible to remove delays. Create cross-functional teams as much as possible. Of course, some skills must be shared between teams.
- Work together within the team without delay. As much as possible, use colocated, cross-functional teams.
- Work on small batches to improve feedback. Decompose work into small pieces of functionality that can be validated (often called vertical slices).
- Use a shared backlog when multiple teams need to work on related features at the same time.
- Manage work-in-process to improve flow.
- Build quality in. Use test-first methods at least at the acceptance test level. Continuously improve your methods.

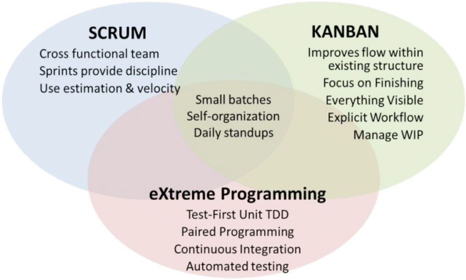
Leanban as an integrative approach

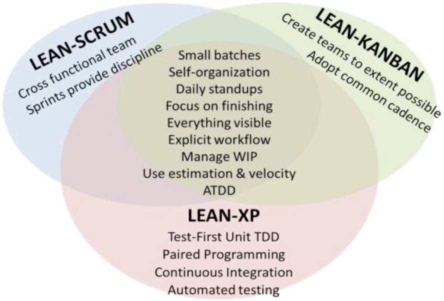
It is not sufficient merely to try to integrate Scrum and kanban. It requires incorporating their principles and mindsets to meet the needs of the current context. There are three reasons why this is important.

- There are some practices not in Scrum or kanban that virtually all teams should incorporate.
- We want a single mind-set from which to select practices. In other words, all teams should be following the laws of software development as well as being management friendly.
- When selecting Agile methodologies, the best approach is tailored to meet the current needs of the team

Leanban uses Lean-Thinking to guide teams to incorporate Agile practices into their work. Lean-Thinking reconciles the mindsets behind the various Agile methodologies: what is common, what can be incorporated from one to the other, and what is unique.

The figures below illustrate how this would apply to the three primary Agile methods. The first figure shows the practices of Scrum, kanban and XP as commonly understood. There is not much overlap. The second figure shows the greater overlap and the distinctives that result from using Lean-Thinking.





There is much more overlap and the distinctions are obvious. The difference between Lean-Scrum and Lean-Kanban boils down to four practices:

- Whether you use estimation and velocity
- Whether you can achieve cross functional teams
- Whether you have iterations or are purely flow based
- Whether you start by creating cross-functional teams and the roles of Product Owner and Team Agility Master (Lean-Scrum) or by seeing where you are and looking for organizational changes that are easy to achieve (Lean-Kanban)

Begin with what we have learned from Scrum, kanban and XP and use the practices that will benefit all teams. Apply Lean principles to decide upon practices non-universal practices and to help create additional practices as needed.

Implementing Leanban

How you start with Leanban depends upon where you are. Are you currently using Scrum or kanban? Or is Leanban going to be your first transition to Agile? Here are some considerations.

Moving from Scrum to Leanban

Moving from Scrum to Leanban is accomplished by adding the following practices:

- Create an explicit workflow within the sprint (that is, discuss how stories are to be done)
- Manage WIP within this workflow
- Implement Automated Test-Driven Development (ATDD) to some degree. At a minimum have test specifications for a story prior to its implementation.

Moving from Kanban to Leanban

Moving from kanban to Leanban is accomplished by adding the following practices:

- Use estimation and velocity if needed
- Break work items down into small stories at the beginning of the development value stream
- Create cross-functional teams to the extent possible
- Implement Automated Test-Driven Development (ATDD) to some degree. At a minimum, have test specifications for a story prior to its implementation.

Moving from Waterfall to Leanban

Moving from Waterfall to Leanban is accomplished in this way:

- Use multi-functional teams *or* creating an explicit workflow across the various groups involved in product definition, construction, and delivery to the market.
- Follow the steps in “Moving from Scrum to Leanban” or “Moving from Kanban to Leanban.”

Leanban Practices for Multiple Teams

Leanban focuses on individual teams. There are times when these teams must coordinate with each other. Lean-Agile prescribes three practices that are essential for coordinating the Leanban teams and enabling quick feedback.

Coordinate the work with backlog management

It is important that teams work in a coordinated fashion. If one team builds a piece of a feature while other teams work on other segments when time comes to integrate things it is highly likely that the components will not work as well as expected. The situation now is that the teams need to collaborate but the team that built their component first is now busy on other things and the delay between when they first wrote their code and now means it will take them longer to fix anything that is wrong.

It is essential that teams work in a coordinated fashion to both improve collaboration and eliminate the delays between getting out of synch with related teams and discovering errors in understanding what is happening. One method to achieve this is to take the product backlog and have the product owners of the teams create coordinated backlogs for their teams so every team is working on related features at the same time.

Coordinate the teams with a common cadence

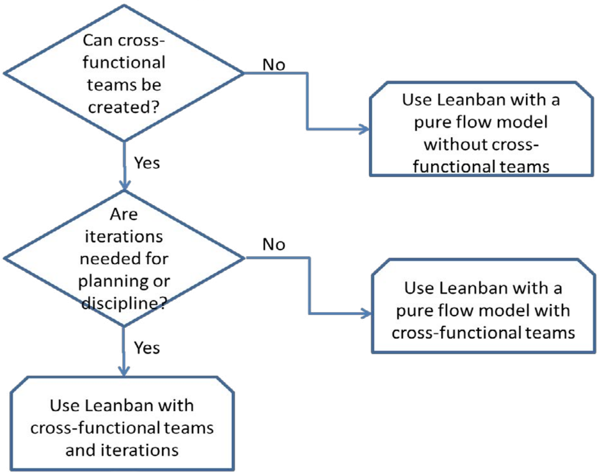
Teams must use the same schedule for delivering working code. Maintaining a common cadence allows the teams to discover and resolve integration errors without delay. The best cadence for integrating code from multiple teams is continuous integration. When that is not possible, a common cadence across all teams enables consistent points of synchronization.

Consider creating temporary cross-functional teams for the duration of the project

Cross-functional teams are extremely powerful. They are more efficient and are usually more effective because the resulting collaboration leads to more creative ideas. Of course, it is not always possible to create permanent cross-functional teams. It may be that members of one team are spending most all of their time supporting another team. In that case, it can be useful to assign them temporarily to the team they are supporting. This type of relationship often happens when one team is building or modifying a shared component on behalf of another. Temporarily dedicating a few members of a supporting team to join a team that they support can remove the dependency on the supporting team and allow the rest of that team to focus on its own work.

Where to Start

Use this decision aid to help you know where to start.


Can cross-functional teams be created?
No
Use Leanban with a pure flow model without cross-functional teams
Yes
Are iterations needed for planning or discipline?
No
Use Leanban with a pure flow model with cross-functional teams
Yes
Use Leanban with cross-functional teams and iterations

Roles in Leanban

This section mentions the roles in Leanban. Roles are described more fully in "Part III: Roles and the Team" on page 35. When considering roles, it is good to remember that roles are not people! They are responsibilities that are held by people.

The following roles are required for Leanban:

- Product Owner (PO)
- Team Agility Master
- Development Team

Additional roles can include:

- Product Manager
- Specialized roles

Use whatever terms you want for these roles. For example, some companies who have a Technical Program Manager role which essentially combines the roles of Team Agility Master and Product Owner.

Product Owner. The Product Owner represents all customers and manages the product backlog. Responsibilities include sequencing the product and iteration backlog; assisting the team in decomposing work items into stories; ensuring that items are being worked on in the proper order; and making the business value from which the stories sprang clear to the team.

Team Agility Master. The Team Agility Master role combines coaching, facilitation, and focusing on continuous improvement by the team. It extends the traditional Scrum Master role by combining Agile tenets with Lean Thinking. The Team Agility Master shepherds the team, creates a trustful environment, facilitates team meetings, asks the difficult questions, removes impediments, makes issues and problems visible, keeps the process moving forward, and socializes agility within the greater organization.

Development Team. The development team includes Business Analysts, programmers, and testers. There may or may not be different people filling the role of programmer and tester. The separation is due more to the fact that people already often have these roles and it may be disconcerting to change them. The roles relate more to the skills people have. Both roles are responsible for developing quality code, quickly, reliably and sustainably.

Here are some important concepts:

- The **Team** estimates size of backlog items, makes design and implementation decisions, commits to increments of deliverable software, and delivers it. The team tracks its own progress, is autonomous and self-organizing, dynamically adjusts its plans as needed/appropriate, and is accountable as a team for delivering as promised.
- A **Swarm** is a temporary group of team members that works together on one story to bring it to completion.
- **Testers** seek to discover why defects are happening as well as testing for defects.

Product Manager. This role aligns Business with development/IT. The Product Manager forms the product vision, improves ROI, manages customer and stakeholder expectations, road-mapping and release planning, prioritizes the product backlog, provides clear, testable requirements to the team, collaborates with both customer and team to ensure goals are met, and accepts product at the end of each iteration. Note: This role is probably outside of the scope of Leanban but it is good to know about.

Product Managers spend about 80% of their time focused on markets and customers and 20% on teams. Product Owners spend 80% of their time focused on teams and 20% on markets and customers.

Specialized Roles. Leanban recognizes that there are often specialized roles that may not be available to all of the teams. Examples include Architect, Graphic Design, and Analytics. We would prefer not to have these specializations because they often cause bottlenecks; however, they are often unavoidable or can only be avoided at exorbitant costs. The work for people filling in these roles will usually need to be managed via a personalized kanban board.

Leanban Practices

Leanban looks across multiple software methods to discern fit-to-purpose and to apply practices thoughtfully. It does not insist on purely following one approach when that means being limited in using good practices from others. Borrowing from Scrum, kanban, and XP, Leanban prescribes these core practices:

- **Small batches.** Work items being worked on directly are small (1-3 days). The amount of work being done across the value stream is organized to provide quick feedback. This requires that the started work items be as small as possible and focus on quick business value delivery and/or quick feedback.

  Leanban follows the mandate of focusing on the delivery of business value in small increments. The workflow starts with the definition of the Minimum Business Increment that can be built, deployed and consumed. Each MBI is broken down into vertically sliced features and added to a common backlog. The different teams pull items from the backlog, break them down into stories, and implement the stories in small batches.

  At the team level each, story typically should take no longer than three days to complete from start to finish. Stories that will be worked on by a more than a single team member can be larger than stories that will be worked on individually.

- **Self-organizing teams**. Although teams must work within the context of the overall organization, they should still self-organize their own work within that context.
- **Daily stand-ups.** The team has a stand-up meeting every day, in the same place and at the same time. All team members need to be present as well as the Team Agility Master. Who leads is at the discretion of the team. All work items in progress are quickly reviewed. See "The Daily Stand-Up" on page 86.
- **Focus on completion of work items not starting new ones**. Team members attempt to complete open items instead of opening up new ones. This enables the team to maintain a meaningful cadence. It also limits WIP naturally.
- **Make all work visible.** Anything done by the team is visible on the board or in a tool in a manner that the following aspects can be seen: Its status, if it is blocked, and who is working on it.

- **Include management in what the team is doing.** Take the attitude that their understanding will be helpful. It is a holdover from the early days of Agile to keep management at arms-length. Management is as much likely to be committed to the work being done as the teams.
- **Make all workflow rules explicit.** Team members follow a well-defined sequence of events by which collaborative problem -solving is attained. Everyone on the team knows what the team's work policies are and agrees to follow them. The policies are not necessarily written down. They should be reflected on the board (virtual or physical). These policies can be changed at any time by a team-wide decision.

  Explicit workflow rules help people understand their responsibilities. They also allows for trying things out and seeing if improvements can be made. They are the basis for the Lean approach, "Plan-Do-Check-Act," as well as WIP management.

  People discuss the workflow not to limit their creativity but to understand what everyone is doing. In other words, discussions ensure alignment.
- **Explicitly manage WIP throughout the process.** Establish limits on the number of items that can be being worked on. WIP management can include limiting:
  - The number of projects people are working on
  - The amount of work that will be done in the iteration
  - The number of items in process at any one moment
  - The number of items allowed at any step in the workflow

  WIP limits can be refined gradually. You can start high and then lower them to the appropriate levels so as to remove delays in the workflow and to identify problems quickly.
- **In high-level planning, use estimation and velocity.** In a development or IT group, planning can be essential. Using estimation and velocity enables better inter-team coordination through more informed projections of when work will be completed. Estimates enable some amount of high-level planning even if they cannot be used for managing work.
- **Minimize cycle time.** In a maintenance organization that deals with urgent production bugs, minimizing cycle time is more important than estimating and measuring effort.

- **Use Acceptance Test-Driven Development (ATDD).** Acceptance Test-Driven Development is one of the best practices to be used. There are different degrees to which you can adopt it, but it should be used to some degree. See *www.netobjectives.com/atdd* and *www.netobjectives.com/blogs/how-start-acceptance-test-driven-development* for more information.

In addition to these core practices, your situation may call for other or additional practices including:

- **Use cross-functional teams as much as possible.** Using cross-functional teams is one of the best practices available; however, they are not always easy to achieve. See *www.netobjectives.com/files/resources/articles/ValueOfTeamsAndHowToCreateThem.pdf* for details.

  Start by creating cross-functional teams if possible. Balance this with the reality that too much organizational change can cause problems. If you cannot create cross-functional teams or are worried about changing team structure, start with where you are and use a flow model. From this starting point you can continue to improve your teams' organization as people become more familiar with the principles of Lean.

- **Use a common cadence with or without iterations.** You may use iterations for planning and/or for the discipline it provides. If you choose not to, you must use cadence in order to provide opportunities to synchronize with other teams and roles.

Then, use Lean to drive your particular approach:

- **Lean-XP.** Test-first; strive for TDD when you can (starting with ATDD); paired-programming; strive for continuous integration and automated testing.
- **Lean-Scrum.** Cross-functional teams if you can; use iterations to provide disciplines and a common cadence.
- **Lean-Kanban.** Create teams to the greatest extent possible; adopt a common cadence.

Adopting and Abandoning Practices

Few practices are universally applicable. Variations in software development needs are too large for universality to prevail. Unfortunately, most approaches don't specify when their practices should be applied and how to abandon them and adopt new ones when necessary. People typically need guidance when beginning to adopt a new practice. This guidance should be based, at least in part, on the situation people are in. Examples include having cross-functional teams and working on too many things.

After people have started and are becoming at least competent at a beginner's level they will want to take more control of their approach and tweak it to their specific context. This requires thinking about the objectives for the practice and then determining whether they it can be realized in a different way. It is fine to move from one practice to another as long as the team honors the purposes and objectives of each. The danger of not understanding the "why" behind the "how" is that the team might abandon effective practices just because they are difficult. Naïve assumptions about practices create problems.

For example, some teams have difficulty closing out their iterations and decide to abandon iterations and call it "kanban." That is not a good practice and it is not kanban. First, the team must understand the objectives for iterations, such as:

- Small work items for better planning
- Consistent cadence
- Keeping team members in close collaboration
- Providing helpful pressure to ensure that work is finished.
- Team discipline
- Ensuring testing does not lag behind programming because stories must be completed within the iteration

If a team has abandoned iterations and their testing lags behind programming, they may benefit from adopting iterations within Leanban.

Summary of Leanban Practices and Value

The following tables summarize Leanban practices and the value they provide.

Alternative methods of getting value

Practice	Value Provided	Alternative Method of Getting Value
Time-boxing	Cadence for input, output, demonstration, and retrospection Discipline Small batches Visibility in and out Velocity Planning method Focus	Can have independent cadences. Must bring discipline to each story since they may take longer than should without it. Use small batches / stories. Use visual controls throughout workflow. Measure velocity via cadence. Plan ahead if valuable. Take a value-centric approach.
Cross-functional team	Limits WIP Reduces hand offs Improves feedback Minimize delays Improves collaborative problem-solving Improves learning	Attending to flow while using as close to a true team structure as possible can achieve these values
Use of a Product Owner	Reduces unneeded features	An equivalent "one-voice" is needed regardless of method.
Finish stories quickly	Minimizes delays Reduces WIP Provides quicker	Time-boxes or discipline to complete stories quickly. Decompose to small stories. INVEST

Practice	Value Provided	Alternative Method of Getting Value
Remove structural impediments	Minimizes delays Provides quicker feedback	Manage WIP Kaizen / intentional process improvements Shorten feedback cycles
Balance workload	Reliable flow of value	Pull work based on velocity Manage WIP

What practices achieve

Practice	What It Achieves
Explicit workflow	Enables everyone to know how we do software in our context Facilitates learning.
Daily Stand-ups	Keeps people informed (might not be strictly needed if the team is colocated).
Make everything visible	Facilitates learning and management. Helps detect challenges. Reduces the tendency to add unneeded reviews
Common cadence / iterations	Enables early synchronization of different teams through code integration and testing.
Build incrementally and iterate on the increments	Encourages short feedback cycles and learning.
Focus on finishing	Avoids too much WIP and helps look for opportunities to collaborate.
Do continuous integration	Detects out-of-synchronization errors.
Estimate work items and compute velocity (unless a maintenance group)	Validates understanding of items being worked on by the teams. Facilitates planning.
Work in small batches	Encourages faster feedback. Makes it easier to avoid workflow delays. Enables people moving around as needed.

Practice	What It Achieves
Use small stories	Improves clarity and specificity. Encourages faster feedback. Makes it easier to avoid delays. Enables people moving around as needed.
Manage WIP	Eliminates delay and speeds up feedback. Reduces multi-tasking.
Create cross-functional teams to the extent possible	Eliminates delay, speed up feedback and learn faster. Enhances horizontal flow of value..
Use test-first methods	Creates better understanding of what is needed. Improves collaboration and resource balancing between development and test. Facilitates automation of test.
Paired programming	Encourages collaboration, shared knowledge of the code base, and increased discipline.

Artifacts in Leanban

Leanban uses the standard process artifacts used in both Agile and Scrum. In particular:

- Minimum Business Increments, features, stories, tasks
- A kanban-style board (Kanban-style boards are more effective even when doing iterations)
- A product backlog
- A iteration backlog if iterations are being used
- An impediment backlog
- Burn down and burn up charts as appropriate
- Cumulative flow diagrams
- Scatter diagrams

Part II: Lean-Agile Thinking

This section summarizes some of the basic drivers of Lean-Agile thinking. The Net Objectives website offers an extensive collection of resources about Lean-Agile thinking applied to software development.

Basic Motivations

Agile software development implements the principles of Lean software development at the local team level. Lean thinking extends the benefit across the enterprise. Together, Lean and Agile create a framework that:

- Enables the team to discover what the customer needs and to deliver value quickly and repeatedly
- Empowers the team to remove impediments that are in the way of delivering value quickly
- Encourages continuous refinement of the approach being used to deliver more value
- Optimizes the flow of value delivery across the enterprise

The primary characteristics of Lean-Agile thinking include:

- Building software in iterations
- A product backlog that is prioritized to deliver the most business value
- Cross-functional teams where people use their combined skills to get work done, using a facilitator
- High-bandwidth communication as much as possible
- Daily Stand-up meetings for the whole team
- Simple visual controls at the work site to report progress and issues
- Risk mitigation: addressing issues, impediments, and dependencies early
- Continuous process improvement and best practices

Reasons for Going Lean-Agile

Need	How Lean-Agile Addresses the Need
Realize Business value quickly	The Business prioritizes work to increase revenue/ benefit, to increase market position and to increase feedback (and reduce risk). The product team is a partner in the value stream.
Gain clarity on customer needs	Work on aspects of features about which customers are *clear*. Use short iterations to increase feedback and grow knowledge. Minimize speculation; let the design evolve with experience over time. This reduces the tendency to over-design and minimizes complexity.
Manage projects better	Metrics are reported using simple visual controls (also known as "information radiators"), with an intense focus on value creation, current work, and reducing impediments. Radiators are managed locally and are visible. Work is planned and re-planned based on current knowledge. This works because the planning horizon is short. Testing is central to development. The goals of testing are to ensure high-quality code at each iteration, find bugs, discover root-cause issues that lead to defects, and create acceptance tests for each feature and each story. Short deadlines and close contact with the customer ensure a constant sense of urgency without panic. The is rarely surprised.
Motivate teams	Short feedback cycles and close contact with the customer results in frequent quick wins. Catching mistakes early allows more time to deliver value. Teams and individuals gain a greater sense of accomplishment, progress, and purpose.

Twelve Foundational Principles of Agile

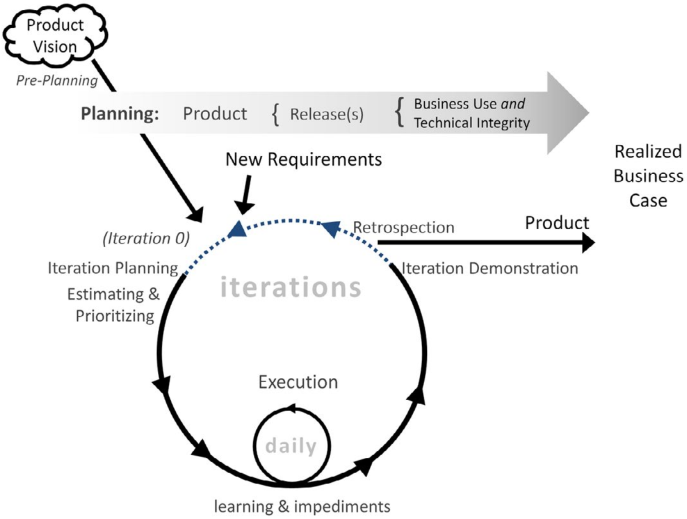
Principle	Description
Satisfy the customer	The highest priority is to satisfy the customer through early and continuous delivery of valuable software. A product backlog is prioritized to realize the most Business value.
Welcome change	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver frequently	Deliver working software frequently, according to how quickly the customer can consume it.
Whole team, daily	The Business and developers must work together daily throughout the project
Motivated people	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
High bandwidth communication	Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team.
Measure of success	Working software is the primary measure of progress.
Sustainable	The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Technical excellence	Continuous attention to technical excellence and good design enhances agility.
Simplicity	Practicing the art of doing just enough and no more than is necessary. Minimizing the amount of work not done.
Emergent design	The best architectures, requirements, and designs emerge from self-organizing teams.
Continuous improvement	At regular intervals, the team reflects on how to become more effective and then adjusts its processes and behaviors to do this.

Lean-Agile Follows a Regular Life-Cycle

The life-cycle of Leanban begins with the product vision, runs through a process to gather and prioritize overall requirements and then iteratively develops and releases product so that the Business can realize value quickly as they learn more about what the customer needs.

Planning and re-planning is done continuously along the entire value stream from vision to support. Depending on size and complexity, organizations may use a multi-tier approach to manage their overall portfolio of products, projects and programs.

Lean-Agile principles help Scrum to focus this by driving all effort to delivery Business value.



Lean Principles and Agile Practices

Lean has two primary emphases: Achieve fast-flexible-flow and discover what is needed. These principles work together to remove delays and to minimize the amount of work in process. When Work-in-Process (WIP) is high, the team will thrash and there is a risk of greater waste if directions have to change.

Principles and Practices: Achieve Fast-Flexible-Flow

Lean Principle	Agile Practices	
Optimize the whole	Product vision	Product backlog
	Colocation	Business value burn-up
	Release planning	Attention to Business value
	Story point sizing	Feature/Story sequencing
	Line of sight on release and iteration goal	
Eliminate waste	Process improvement	Kaizen
	After Action Review	Root Cause Analysis
Build quality in working software	Validation-centric	Acceptance TDD
	Story/Task done-ness	Continuous integration
	Integrate development team and QA	
Deliver without delay	Iteration planning	Backlog management
	Iteration commitment	Value-creation metrics
	Establish a regular iteration cadence (1-4 weeks)	Iteration review

Principles and Practices: Discover what is needed

Lean Principle	Agile Practices	
Defer commitment	Pull Just-In-Time Stories	Story unfolding
Create knowledge	Information metrics Daily accountability Iteration retrospection Impediment tracking Community of practice	Dashboards After Action Review Dependency tracking Velocity tracking Collaboration
Respect people	Develop reasonable descriptions of standard work for leaders and team members Provide visibility of responsibilities so it is clear who does what. Develop team norms that team members treat each other with respect and that the team (not individuals) succeeds or fails as a group.	

Lean-Agile Involves a Commitment to Continuous Improvement

A process describes how a team of people work together. The team is responsible for their processes. The team must agree together to follow their processes and, when they discover that something is not working, to stop and change it based on their knowledge of their own conditions, and then follow resume following the process again. Processes exist to support the team and not the other way around.

Processes are never perfect nor are they universally applicable.

Managers facilitate a spirit of continuously improving processes: empowering people to improve and expecting that it is done. They hold teams accountable for results. Managers guide teams by asking good, thought-provoking questions to get people to think clearly about their problems.

Process improvement activities need not take long. But they do require a commitment and *discipline* to stop, adjust, and then go on. This builds the team's confidence in itself, its ability to solve problems, and its ownership of its own processes.

Tip. Continuous improvement is not easy. Often, teams over-focus on the need to deliver on the iteration goal and don't see process problems. The Team Agility Master and managers serve the team by calling attention to process problems and facilitating retrospections.

Lean-Agile Involves Oversight of the Flow of Value

Oversight concerns the control of time, money, and quality relative to the flow of value. It is the responsibility of value stream management, the Product Manager, the Product Owner, the Team Agility Master and the team. Short planning horizons make this oversight easier and more effective.

At this level...	Oversight involves these activities
Overall product management	The Product Manager works with the Business to prioritize projects, capabilities, and features within the planning horizon. The Product Manager also sets goals for initiatives and releases. Product Manager advocates for the product to ensure clear understanding and progress.
Project	The Product Manager creates the product charter and explains it to the team. The product charter is displayed in the team's work area. At releases and iterations, status is reviewed against the product charter. Daily review of business value burn-up (feature completion) and impediment burn-down. Create the project board for the team.
Iteration	Daily Stand-ups cover progress relative to priorities. Visual controls are updated daily by the team and the Team Agility Master so that status is always visible to the team and to management. The Product Manager and other management make a practice of "going to the Gemba" (where the team is working) to get a sense of what is going on rather than simply relying on reports. Local teams assume responsibility for governing themselves in terms of process and process improvement and standards and practices to use. Continuous builds assure that high quality code is always maintained.

Foundations

Respect People. People hold the knowledge about processes, problems, and improvements. Teams use their local knowledge, guided by enterprise standards, to create processes to deliver product. Each team member must deliver on his or her own commitments and depends on others to do the same. The Product Manager trusts the team to deliver on its commitments.

Continuous Process Improvement. Processes exist to serve the people, to help them get their work done. No process is perfect; when problems arise, the team is responsible to ***stop***, ***change the process***, and ***start again***. Without affixing blame to any one person.

Focus on Value. Team decisions are based on what delivers business value quickly. The Business prioritizes what features are necessary to satisfy needs of the Business and its customers.

Remove Impediments. The team is responsible for identifying and, as much as possible, removing anything that impedes their ability to deliver value. If it is beyond their scope of control, then management assumes responsibility. Impediments are tracked.

Principles

Optimize the Whole. Focus on the entire portfolio of products that are in the "value stream" for the enterprise. Do what is needed to improve the overall flow of the most important value. At the project level, business analysts, developers, and testers work together to make high-quality code. Bugs are not tolerated.

Eliminate Waste. Eliminate anything in the product development process that creates waste, anything that does not add business value or customer value, that has been started but has not been put into production, that delays development, or that is more than what the customer requires (or goes beyond delighting the customer).

Build Quality In. Quality is *built into* the product, not tested in. It is part of design. Design patterns can inform design choices that create sustainable, less complex, more flexible code. Quality assurance and testing exist less to catch bugs than to identify causes of bugs so that processes and systems can be corrected. Ideally, code development will be test-driven or at least tests should be considered before writing code.

Create Knowledge. Each development project, environment, and team is unique. Each iteration offers a learning opportunity. Knowledge that is not transferred or is not embedded in processes is lost. Be intentional about discovering lessons and facilitating knowledge exchange in a safe, blame-free way. Learn what customers know and let the requirements emerge based on what they learn while doing.

Use Visual Controls. Progress charts, product vision, rules, and instructions, are visible everywhere. Teams use regular, common meeting rooms for high bandwidth communication.

Test Early, Test Often. Validate that the team understands what the Business needs, that the product produced is what was required, and that the product is of sufficient quality. Move testing as early as possible into the development process. No feature or story should be created without a corresponding acceptance test. See "Quality: Testing" on page 144.

Foster a Team of Peers. Communication between roles must be unrestricted and transparent, based on mutual respect and personal safety. All roles are responsible for product quality, must act as customer advocates, and must understand the business problem they are trying to solve.

Practices

Use Agile Development. Teams should follow their particular agile framework for product development. Ideally, the whole team receives training in the framework before they begin so that everyone is on the same page. Every team has a Team Agility Master assigned to them to facilitate their work.

Deliver Incremental Value. Small, quality deliverables show progress, promote learning, and improve team morale. They help detect risks, bugs, and missing requirements early. They force key decisions when feedback can still make a difference. Keys to frequent delivery include: keeping the scope of an activity small, working on deliverables in a "just-in-time" manner, leaving options open by eliminating premature decisions, and using continuous build and test. Plan, execute, and measure progress and team velocity based on the delivery of increasing value.

Foster Open Communication. Historically, many organizations and projects have operated purely on a need-to-know basis, which frequently leads to misunderstandings and impairs the ability of a team to deliver a successful solution. Lean-Agile requires open and honest communications, both within the team and with key stakeholders. A free-flow of information not only reduces the chances of misunderstandings and wasted effort, but also ensures that all team members can contribute to reducing uncertainties surrounding the project.

Stay Agile, Adapt to Change. The more an organization seeks to maximize the business impact of a technology investment, the more they venture into new territories. As the organizations learns about the needs of the marketplace, requirements will certainly change. It is impossible to isolate a project from these changes. All core roles must remain available throughout a project so that they can contribute to decisions arising from these changes. The team works together to address these issues based on the knowledge each member has. The contribution of all team roles to decision-making ensures that matters can be explored and reviewed from all critical perspectives.

Practice Good Citizenship. Team members are stewards of corporate, project, and computing resources. This attitude manifests itself in many ways: conducting the project in an efficient manner; optimizing the use of system resources; reusing existing resources. Team members respect each other's time and skills and strive to live up to commitments made to each other. Each team member is responsible for an environment that is clean and accessible, agreeing on a discipline that helps them sustain the environment.

Essential Skills for Agile Developers

When an organization is transitioning to Agile, developers often raise many concerns, especially how they are supposed to write code in an environment in which requirements are changing on them and there is little or no architecture. Addressing those concerns involves gaining knowledge of Agile development and acquiring certain skills that are essential to thrive in an Agile environment.

Each of the following skills is easy to learn and yet provides huge improvements to the developer's code. The resources section describes a number of on-line resources to help you acquire and improve them.

Skill	Description
Minimize complexity and rework	This is more of an attitude than a specific skill set. It provides an alternative to the common extremes of hacking (causing much rework) and over-design (causing greater complexity than needed).
Keep the system in a running state	Finding errors is the developer's biggest time waster. When several changes occur and then something goes wrong, it is hard to know what change caused the error. By making incremental changes to detect errors immediately, this saves a lot of wasted time searching for errors.
Programming by intention	Programming by intention was made popular in eXtreme Programming. It is easy to do and provides de-coupling, cohesion, encapsulation and clarity. If there is one practice you can (and should) easily pick up, this is it.
Separate use from construction	This eliminates the situation in which integrating new code into established systems exceeds the cost of writing the code in the first place. It makes the developer responsible for abstraction and encapsulation. It provides discipline for hiding the specific implementation you are using.
Commonality and variability analysis	A straightforward technique to identify the concepts in your problem domain. While developers are capable of thinking in terms of abstractions and implementations, few do. This provides a model to further the proper use of object-oriented modeling.

Skill	Description
Encapsulate that!	Encapsulation lives at all levels. We are often unsure of how we should implement something. We often make a choice which proves bad and requires rework or we overbuild increasing complexity. If we encapsulate our code and designs we can more efficiently make needed changes when we understand what is really needed.
Think about tests up-front	Think about tests up-front even if you do not write them up-front. It takes no work, increases understanding, and prevents many errors.
Shalloway's Law and Shalloway's Principle	Shalloway's Law states that if you have to find more than one thing when a change is made, you will miss something. Shalloway's Principle says to avoid situations where Shalloway's Law applies. If using object-oriented programming, then the proper use of object-orientation and encapsulation means that you never have to find more than one thing when a change is required.
When and how to use inheritance	Inheritance has a long heritage of improper use. The seminal book *Design Patterns: Elements of Reusable Object-Oriented Software* provided a better way to use it.
Refactor to the open-closed	While refactoring can mean fixing code, it also is used to improve designs that were good when they were written but decayed because of new requirements. This technique improves the design of quality code as more knowledge about the system's needs become available.
Needs vs. capabilities in interfaces	Interfaces provide a different method of design. Instead of designing from the perspective of what we have, we should be designing from the perspective of what is needed. This follows the concepts of design patterns as well as several object-oriented design principles.
Continuous integration	Continuous integration is a low cost method of detecting errors quickly. It is essential if you are writing tests up front. See "Continuous Integration (CI)" on page 149.

Part III: Roles and the Team

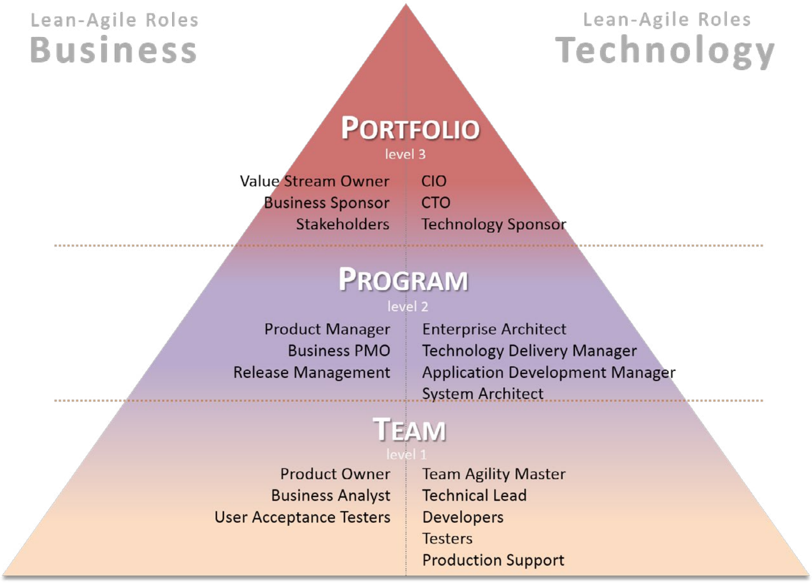
In Lean-Agile, the delivery of value involves the effort of people from all across the value stream: executive, management, and front line teams. People from the Business and technology work together to discover and deliver work that customers can use and can realize value from it.

Leanban focuses on the Team level. For completeness, this section also mentions Portfolio level and Program level roles.

This section provides the following:

- **Roles by level.** A summary of Portfolio, Program, and Team level roles.
- **Responsibilities and standard work for some of the roles.**
- **Team of Peers.** The special notion in Lean-Agile that teams are composed of "peers" working together and what that means. Sometimes, this requires using "swarms" and "teamlets."

In Lean-Agile, the delivery of value involves the effort of people from all across the value stream: executive, management, and front line teams. People from the Business and technology work together to discover and deliver work that customers can use and can realize value from it.



Roles by Level

The roles shown here represent a general model used by many enterprise-level organizations. Your company might use different names or combine them in different ways. The important thing is that the responsibilities of these various roles are being covered.

The Portfolio Level

The Portfolio Level includes a diverse group of stakeholders working across organizational boundaries. The Portfolio Level is responsible for priority, the order of work, and deciding what is active. Roles involved in the portfolio include the Value Stream Owners and the sponsors. The Business engages with the technology organization at the Portfolio Level.

In many organizations, Portfolio Level work is carried out by a PMO.

Role	Focus of the Role
Value Stream Owner	Continual realization of the highest Business value Optimal cycle time from idea to realization
Business Sponsor	Realizing Business value and ROI Providing final approvals and resources/funding
Technology Sponsor	Technology and process to realize Business value incrementally with quality solutions
Stakeholders	Representing interests, constraints, and requirements for the organization

The Program Level

The Program Level is the center of responsibility that is focused on an application area. Work is comprised of releases, enhancements, production support, and maintenance requests. This level also is responsible for program ecosystem and operational metrics.

Role	Focus of the Role
Product Manager	Defining the scope for Business discovery. Prioritize and incrementally realize Business value, ROI.
Business PMO	Visibility and progress of portfolio Book of Work and/or program backlog. Project administration and oversight on behalf of the Business Business standards for the organization
Technology Delivery Manager	Continual incremental delivery of quality solution(s) Knowledge and learning leadership
Application Development Manager	Development of software functionality with integrity Extensibility and maintainability that can be delivered incrementally System evolution, architecture and design Code quality (developer standards) and managing dependencies and/or collusion points Determining appropriate required skills/ expertise for technical development

The Team Level

The Team Level is composed of development teams and shared services. Each development team is composed of the developers, testers, analysts and SMEs required to produce and implement a Business value increment, for quality assurance, and continuous incremental improvement. A team's work is composed of the stories and tasks for a specific release, enhancements, production support, and maintenance requests.

Skills needed in a team typically include analysis, design, code, build, deploy, testing, acceptance, validation, and implementation. Extended skills needed might include: Architecture and design, build and release management (environments), SEO, and subject matter/domain expertise (technical, Business, and customer).

Role	Focus of the Role
Product Owner	Acceptance, validation, and adoption of : MBIs and features Serving as an interface between the Business and the team: • Representing enterprise Business value to the development team • Representing the team's perspective to the Business and to the Program Level
Team Agility Master	Instilling and nurturing Lean-Agile practices Coaching for visibility, transparency, and Lean-Agile practices Continuous incremental process improvement and team efficiency
Business Subject Matter Expert	Acceptance criteria Validation and implementation of increments Contributing to requirements
Business Analyst	Synthesizing/refining requirements Business analysis Acceptance tests
Technical Architect	System architecture and design

Role	Focus of the Role
Technical Lead	Design and code quality Defining appropriate skills for technical development
Developer	Designing, writing code, unit test Developing use cases as needed or appropriate Incremental value delivery, quality, and continuous improvement
Tester	Early involvement with developers and analysts to support ATDD and TDD Representing testing discipline within the team Developing and execute test cases Coordinating testing activity to ensure a last 'safety net' prior to deployment to production
Production Support	Providing operational and production input. Preparing and execute transition to production
Database Administrator and **Database Developer**	Supporting the creation and implementation of database projects Performing day-to-day administration and maintenance of database servers. Note: These roles are often limited; they are not expected to participate in Daily Stand-ups or retrospections unless they are a member of the core team.

Product Owner

The Product Owner is the pivotal role in enabling the seamless flow of development from value to product.

The Product Owner (PO) is the person who connects two key functions and associated groups of people: the ones who identify what value is needed (the Business) and the ones who determine how and implement it (development team).

- To the development team, the Product Owner represents the enterprise Business value and the needs of Program Level stakeholders , defining and prioritizing the team backlog.
- To the Business, the Product Owner represents the development team: the value that has been realized and the needs they have.

The Product Owner is the proxy for the customer, the go-between for those who define value at the Minimum Business Increment level. The Product Owner is like the transmission of a car.

- The engine of the car is those who define the value. The wheels are those who implement value.
- The Product Owner connects the engine to the wheels, so to speak, by answering the development team's questions about the value they will implement. The Product Owner is the team's main source for insights into customer and stakeholder motivation. The Product Owner helps the team know just what the end customer or the enterprise is expecting of their products.
- The Product Owner connects the wheels to the engine by feeding back implementation issues to those who define the value. The definers of value may be able to redefine it in a way that is simpler or less expensive or even simply possible to implement. This also lets the definers of value know when something is going to be delivered later or at higher cost due to technical difficulties.

Responsibilities

The Product Owner provides the *critical connection* between those who define value and those who implement it. The Product Owner makes it possible for the teams to produce the value that the enterprise needs.

The Product Owner role is responsible for:

- Integrating the team into the value stream (or program)
- Understanding the priorities of the Product Manager
- Managing the team backlog, including populating, prioritizing, maintaining, and monitoring it
- Driving the team at a sustainable pace
- Assessing and accepting development deliverables
- Defining acceptance criteria and assessing whether the product meets the acceptance criteria
- Participating in product demonstrations and retrospectives
- Releasing the product
- Helping to resolve impediments

Standard Work

Accountability	Which involves....
Integrating with the development team	Being available to the team, colocating with the team as much as possible. Ideally, there is one team or at most two teams per Product Owner. Walking the floor and look for issues / delays / improvement opportunities Serving as liaison with the Business and Business SMEs Driving the team at a sustainable pace • Understand and help to allocate capacity for enabling work • Writing stories to represent the requirements and the pace • Explaining the stories to the team with "just in time" elaboration Participating in or observing team meetings • Iteration Planning (includes team commitment to backlog items) • Daily Stand-ups, when possible • Iteration retrospectives • Product demonstration and reviews Helping the team to resolve or escalate impediments as appropriate, providing the Team Agility Master with status of impediments reported by the team Protecting the team from distractions and outside influences, including loaner requests, multiple projects, and production support (where possible)

Accountability	Which involves....
Integrating the team into the broader program	Coordinating dependencies with other Product Owners. Acting as the designated communicator for the team to the value stream (or program) level, discussing with the Product Manager • Team-level prioritization decisions • The implications of implementing the desired value • Non-architectural implementation issues Assessing the Business value of the team objectives. Attending planning events and the Product Demonstration and Review Events.
Managing the team backlog	Populating the backlog. • Decompose features in the Program Backlog into User Stories that go into the team backlog. Each User Story should be sized to be done in one iteration; preferably in two or three days. Prioritizing the backlog . • Work with the team to apply a prioritization method such as Weighted Shortest Job First to put the backlog items into order of priority. • Observe the team as it picks stories from the team backlog and adds them to a "commitment list" for the iteration. • Observe the team as it chooses additional “stretch objective” stories that they will try to finish in the iteration. Maintaining and modifying the backlog. • Refine product backlog to maximize ROI • Add new items or modify existing items based on feedback from stakeholders and learning by the team. • Understand and establish capacity for enabler work

Accountability	Which involves....
Monitoring iteration execution	Understanding and help setup visual controls Reviewing information visibility charts Reading the information visibility boards for signs of problems with the iteration for signs of failing agility Remaining aware of whether the team's other responsibilities (changes, internal projects, unplanned work) are reducing their ability to complete the committed backlog items.
Assessing and accepting product	Understanding priorities of the Product Manager Specifying acceptance criteria for each story in the backlog Accepting or rejecting backlog items at the Product Demonstration and Review at the end of the iteration Deciding with the team about carry-over work
Working with release management	Working with Release Management to release as appropriate Writing stories required for release

Alternatives

Traditional Agile organizations often combine the Product Owner and Product Manager roles into a single role that they call “Product Owner.” Lean-Agile divides these two roles to reduce complexity, enable proper focus on the market on the one side and proper focus on the product team on the other side, and to bring up conflicting interests that often get lost   when the two roles are combined. In most circumstances we do not recommend a classic Agile Product Owner-only approach.

Traditional waterfall organizations often add the responsibilities of the Product Owner to the Program/Project Manager (PgM) role. This is counter-productive. The PgM role is already quite busy with monitoring, reporting, and other traditional program management tasks that do not include product ownership and advocacy. There is never enough “left-over” time available for a PgM to add Product Owner tasks and do them adequately.

The mindset is completely different: the PgM is typically taking a command and control perspective “over” the teams, while the Product Owner is operating within a peer perspective.

These and many other factors combine to make it counterproductive to combine the PgM and Product Owner roles.

Team Agility Master

The Team Agility Master is responsible for shepherding the team, creating a trustful environment, facilitating team meetings, asking the difficult questions, removing impediments, making issues and problems visible, keeping the process moving forward, and socializing Lean-Agile within the greater organization.

Responsibilities

The Team Agility Master champions the needs of the team to the organization and for championing the needs of the organization to the team.

The Team Agility Master is the team's facilitator, shepherding the team as a servant-leader by creating a positive and safe environment, improving team cohesion, being creative and focused, asking questions that focus the team on their processes and norms, and helping to develop the product backlog.

The Team Agility Master communicates status on behalf of the team, bringing issues and impediments to the team, Product Owner, or management, and helping to develop the product backlog.

The Team Agility Master helps the team coordinate with other teams in the Program.

The Team Agility Master role is responsible for:

- Ensuring the process is understood and followed
- Facilitating the teams efficiency and health
- Ensuring visibility on progress
- Facilitating the Daily Stand-up and process improvements
- Helping to remove impediments and keeps the product and the process moving forward
- Providing transparency of process and Work-in-Process
- Facilitating iteration reviews and planning meetings
- Serving as the primary source for proactively radiating project information. As a practical reality, the Team Agility Master is responsible for maintaining the team's project board, ensuring it is up to date before the Daily Stand-up

- If the team uses a project management portal, the Team Agility Master is responsible for maintaining it, implementing security and accounts, and creating reports
- Helping the Product Owner develop the product backlog
- As appropriate, facilitating knowledge sharing and lessons learned across the organization

Standard Work

Accountability	Which involves....
During Planning	Preparing for the upcoming iteration planning session. With all appropriate stakeholders, address the following: • Product vision • Product backlog • Risks / Dependencies • Release plan • Artifacts and deliverables for the project • Iteration length Facilitating the Iteration 0 planning meeting. Supporting the team's ability to commit by communicating team capacity. Negotiating commitment between the Product Owner and the team. Helping team create information visibility boards. Establish the progress charts: Product Burn-up charts and Iteration Burn-up and Burn-down charts. Agreeing on Daily Stand-up location and times. Setting up work areas. Helping the team develop communication rules. Updating spreadsheets and database information. Show number of points that have been committed for the iteration.

Accountability	Which involves....
Daily Work	Reflecting on how the iteration is going, looking for tasks taking too long. Facilitating Daily Stand-ups and ensuring that people clearly address the team and stay focused. Radiating information to show progress clearly and predictably. Updating visibility tools with the latest task and story progress. Posting updated charts prior to the Daily Stand-up, current as of yesterday. Identifying and displaying all impediments and issues that affect team velocity and iteration successes. Clearly showing activity and progress toward resolving impediments. Engaging team members to identify impediments, build rapport, and improve team cohesion. Shielding the team from external interfaces and distractions. Facilitating team meetings and process improvement workshops. Monitoring team cohesion and sustainability. Upholding the Agile process and challenging the team to identify and implement process improvements. Socializing Leanban within the organization.

Accountability	Which involves....
Iteration Closing	Helping to demonstrate the product increment to the Product Manager and other appropriate stakeholders. Facilitating the Iteration Retrospective and identifying actionable improvement goals for the next iteration. Placing improvement stories into product backlog. Sharing the findings from the Iteration Retrospective, including the process improvement stories, with other Team Agility Masters in the organization so that the knowledge can be applied more widely and refined as more is learned.

Practices

Accountability	Which involves....
Create the ecosystem	Helping to build a peer team culture Helping to establish roles Helping to create teams (see "Starting a Lean-Agile Team Checklist" on page 157) Maintaining governance reports Encouraging colocation as much as possible (see "About facilitating teams that are not colocated" on page 51)
Support the value stream	Helping to assess team production capacity Helping to assess readiness to pull stories Participating in governance meetings Facilitating planning events

Accountability	Which involves....
Manage workflow	Facilitating Iteration 0 at the start of a release Maintaining the team's project environment • If using kanban, maintain the kanban board • If using iterative (Scrum), maintain the Agile life-cycle tool • Maintaining visual controls and metrics • Manage the team backlog At the end of the iteration, • Facilitating the Product Demonstration (see page 92) • Ensuring that the team is attending to carry-over work (see page 90) • Facilitating the iteration planning meeting (see page 77) • Helping the team review the iteration.
Help the team improve continuously	Facilitating the Daily Stand-up for the team every day (see page 86) Facilitating improvement sessions for the team using After-Action Review, kaizen, or Root Cause Analysis

Alternatives

In some organizations, project managers play the role of Team Agility Master; however, they must be careful to focus on improving the team's process over against just schedule and budget.

If you choose to use the term "Scrum Master" for this role, be aware that the responsibilities for this role go beyond what is commonly understood in Scrum and other Agile methods. This is why Net Objectives prefers the term, "Team Agility Master" to "Scrum Master."

About facilitating teams that are not colocated

Communication involves written, verbal, and non-verbal / body-language elements. Timeliness and context are important. When teams are not colocated, they are missing the elements that make communication effective. Unfortunately, colocation is not always possible. For example, a project may require using multiple, large development teams or teams may be distributed across multiple sites (local or across countries). This requires special attention by the Team Agility Master and the team.

Here are some special considerations.

- As much as possible, try to locate project team members near each other.
- Use dedicated broadband communication as much as possible: audio, video, story-boarding.
- Use an Agile tracking tool for managing product backlog, impediment backlog, stories, tasks, and tests so that teams can collaborate asynchronously.
- The team has to be disciplined in creating and using artifacts, code, and discussion repositories.
- Teams must define usage criteria for code and tools and must be consistent across all teams on the project.
- Teams must coordinate their schedules for collaboration, planning, Daily Stand-ups, and reviews.

Business Analyst

The Business Analyst role represents the Business to the team, providing business knowledge and expertise. This includes describing Business knowledge and business rules, answering questions about customers, performing analysis that will become stories, and writing acceptance tests.

Responsibilities

The Business Analyst is a member of the team, swarming on work items together with other members of the team if they have skills that will help the story to be completed. Business analysts are commonly involved in acceptance testing.

Here is a quick overview of the team's responsibilities.

- The team drives the product from a tactical perspective. It is autonomous and self-organizing and is accountable to the Product Manager for committing to and delivering product increments within the duration of the iteration.
- The team estimates the size of backlog items, makes design and implementation decisions, and transforms stories into high quality software assets.
- The team works with the Team Agility Master to track its own progress, raise awareness of impediments, and devise options to maintain forward progress.
- The team is responsible for establishing processes and standards, continuously improving them, and then following those processes and standards.

Standard Work

Accountability	Which involves...
During Planning	Participating in sizing, prioritizing, and decomposing stories into implementation stories and tasks At a high level, identifying enabling and systematic evolution issues that must be addressed Identifying stories which might need to be addressed in the upcoming iteration Using their insight to identify skills needed to achieve the iteration commitment Collaborating with the Product Manager and committing with the team to deliver an increment of the product
Daily Work	Focusing efforts on delivery of the highest priority stories and tasks while identifying dependencies Building rapport with team members to increase team collaboration Maintaining a positive and safe work environment, and increase trust Contributing to development and testing by swarming on stories and tasks Helping to resolve impediments that are within the team's span of influence Contributing to the Daily Stand-up Clearly marking effort on remaining tasks and who the owner is As necessary, rewriting stories and tasks to level workload, handling dependencies and impediments, and accelerating results As necessary, re-sizing stories based on any new information or on what has been learned Discussing issues raised during the Daily Stand-up about which the analyst can help Being aware of team cohesion and sustainability Ensuring the test strategy is implemented effectively and in a timely manner

Accountability	Which involves...
Iteration Closing	Demonstrating the product / product increment to the Product Owner, Product Manager, and other appropriate stakeholders Participating in the Iteration Retrospective. Identify and commit to actionable improvement goals for the next iteration based on what worked well and what can be improved

Practices

Accountability	Which involves....
Planning	Developing estimates at beginning of iteration and in story points Decomposing features into stories and tasks Committing to the backlog
Ecosystem	Working within the team's project environment to ensure it meets the needs of the team including tools, logistics, and team room
Produce code that meets acceptance criteria	Working with the Tester to ensure code is tested and that code passes all types of tests
Assuring flow of work	Swarming with other team members on stories and task Managing WIP Working with the Team Agility Master to maintain the kanban board or Agile life-cycle tool and to update the team backlog
At the end of the iteration	Participating in the Product Demonstration Attending to carry-over work Participating in the Iteration Planning Meeting Participating in the Iteration Retrospective Working with Release Management
Help the team to improve continuously	Participating in Daily Stand-ups as a full participant. Resolving impediments to progress.

Developer

The developer role is responsible for working with other members of the team to build potentially releasable features in each iteration. The developer collaborates with business analysts and testers on stories and tasks, ensuring that the results meet all functional and acceptance tests. The developer helps the team resolve impediments, getting help as needed.

Here is a quick overview of the team's responsibilities.

- The team drives the product from a tactical perspective. It is autonomous and self-organizing and is accountable to the Product Manager for committing to and delivering product increments within the duration of the iteration.
- The team estimates the size of backlog items, makes design and implementation decisions, and transforms stories into high quality software assets.
- The team works with the Team Agility Master to track its own progress, raise awareness of impediments, and devise options to maintain forward progress.
- The team is responsible for establishing processes and standards, continuously improving them, and then following those processes and standards.

Standard Work

Accountability	Which involves...
During Planning	Participating in sizing, prioritizing, and decomposing stories into implementation stories and tasks. At a high level, identifying enabling and systematic evolution issues that must be addressed Identifying process and environment issues which might need to be addressed in the upcoming iteration Using their insight to identify skills needed to achieve the iteration commitment Collaborating with the Product Manager and committing with the team to deliver an increment of the product

Accountability	Which involves...
Daily Work	Focusing efforts on delivery of the highest priority stories and tasks while identifying dependencies Building rapport with team members to increase team collaboration Maintaining a positive and safe work environment, and increase trust Contributing to development and testing by swarming on stories and tasks Developing tests and code that conform to the code quality standards agreed to by the team Helping to resolve impediments that are within the team's span of influence Contributing to the Daily Stand-up Clearly marking effort on remaining tasks and who the owner is As necessary, rewriting stories and tasks to level workload, handling dependencies and impediments, and accelerating results As necessary, re-sizing stories based on any new information or on what has been learned Discussing issues raised during the Daily Stand-up about which the analyst can help Being aware of team cohesion and sustainability Ensuring the test strategy is implemented effectively and in a timely manner Ensuring the test strategy is implemented effectively and in a timely manner for SAT, CAT Controlling WIP to ensure flow of development
Iteration Closing	Demonstrating the product / product increment to the Product Owner, Product Manager, and other appropriate stakeholders Participating in the Iteration Retrospective. Identify and commit to actionable improvement goals for the next iteration based on what worked well and what can be improved

Practices

Accountability	Which involves....
Planning	Developing estimates at beginning of iteration and in story points Decomposing features into stories and tasks Committing to the backlog
Ecosystem	Working within the team's project environment to ensure it meets the needs of the team including tools, logistics, and team room
Produce code that meets acceptance criteria	Doing Detailed Design Doing Systems Analysis and Design Working with the Tester to ensure code is tested and that code passes all types of tests
Assuring flow of work	Swarming with other team members on stories and task Managing WIP Working with the Team Agility Master to maintain the kanban board or Agile life-cycle tool and to update the team backlog
At the end of the iteration	Participating in the Product Demonstration Attending to carry-over work Participating in the Iteration Planning Meeting Participating in the Iteration Retrospective Working with Release Management
Help the team to improve continuously	Participating in Daily Stand-ups as a full participant. Resolving impediments to progress.

Tester

A key Lean-Agile principle is to aim for perfection, to improve constantly. It is everyone's job. This informs role of the testing: it is done early in the process and is the responsibility of the team. Every feature and every story must have one or more acceptance tests. The outcome of testing is to deliver fairly well-perfected code where defects have no serious consequences.

Responsibilities

Testing does help to discover bugs. But more importantly, the Tester role is a full partner in the team, to help discover the causes of errors and eliminating them. This involves root-cause analysis, looking at processes, infrastructure, and the understanding of customer requirements.

A person in the Tester role must understand the context for the project and help others to make informed decisions based on this context. A key responsibility is to help the business analyst create acceptance tests up front.

The tester helps set up a testing environment that is consistent with the team's quality objectives. The tester then works with developers and business analysts to help them create and conduct tests with complete coverage.

Here is a quick overview of the team's responsibilities.

- The team drives the product from a tactical perspective. It is autonomous and self-organizing and is accountable to the Product Manager for committing to and delivering product increments within the duration of the iteration.
- The team estimates the size of backlog items, makes design and implementation decisions, and transforms stories into high quality software assets.
- The team works with the Team Agility Master to track its own progress, raise awareness of impediments, and devise options to maintain forward progress.
- The team is responsible for establishing processes and standards, continuously improving them, and then following those processes and standards.

Standard Work

Accountability	Which involves....
Ecosystem	Validating that the team understands what the Business needs, that the product produced is what was required, and that the product is of sufficient quality. Move testing as early as possible into the development process. No feature or story is created without a corresponding acceptance test. Acquiring, building-out, and configuring tools and environments for the team such as: • Agile life-cycle management tool • Source control management • Continuous integration server • Test frameworks
During Planning	Participating in sizing, prioritizing, and decomposing stories into implementation stories and tasks At a high level, identifying enabling and systematic evolution issues that must be addressed Identifying process and environment issues which might need to be addressed in the upcoming iteration Using their insight to identify skills needed to achieve the iteration commitment Collaborating with the Product Manager and committing with the team to deliver an increment of the product

Accountability	Which involves....
Daily Work	Focusing efforts on delivery of the highest priority stories and tasks while identifying dependencies Building rapport with team members to increase team collaboration Maintaining a positive and safe work environment, and increase trust Contributing to development and testing by swarming on stories and tasks Developing tests and code that conform to agreed-upon code quality standards Helping to resolve impediments that are within the team's span of influence Contributing to the Daily Stand-up Clearly marking effort on remaining tasks and who the owner is As necessary, rewriting stories and tasks to level workload, handling dependencies and impediments, and accelerating results As necessary, re-sizing stories based on any new information or on what has been learned Discussing issues raised during the Daily Stand-up about which the analyst can help Being aware of team cohesion and sustainability Ensuring the test strategy is implemented effectively and in a timely manner Ensuring the test strategy is implemented effectively and in a timely manner for SAT, CAT
Iteration Closing	Demonstrating the product / product increment to the Product Owner, Product Manager, and other appropriate stakeholders Participating in the Iteration Retrospective. Identify and commit to actionable improvement goals for the next iteration based on what worked well and what can be improved

Practices

Accountability	Which involves....
Planning	Developing estimates at beginning of iteration and in story points Decomposing features into stories and tasks Committing to the backlog Understanding the context for the project and help others to make informed decisions based on this context. Helping the Business Analyst create acceptance tests up front. Developing the testing plan which must • Satisfy all types of tests • Address issues of quality including Acceptance testing, automated testing, and impediments to progress and quality
Ecosystem	Working within the team's project environment to ensure it meets the needs of the team Configuring tools for ATDD, TDD, testing, and continuous integration
Produce code that meets acceptance criteria	Working with the Product Owner and Business Analysts to implement ATDD Working with the developers to ensure code is tested and that code passes all types of tests. Facilitating the team in using continuous integration including tools, process, and usage Ensuring stories are Done-Done-Done which means: • The system runs on the developer's computer as expected. • The system is verified by running unit tests on a common machine. • The system is validated as being of deliverable quality with functional tests.

Accountability	Which involves....
Assuring flow of work	Swarming with other team members on stories and task as needed Managing WIP Working with the Team Agility Master to maintain the kanban board or Agile life-cycle tool and to update the team backlog
At the end of the iteration	Participating in the Product Demonstration Attending to carry-over work Participating in the Iteration Planning Meeting Participating in the Iteration Retrospective Working with Release Management Helping the team to know how much testing resource is available in the next iteration. If testing has not been finished by the end of the iteration, then working to move testing early in the iteration.
Help the team to improve continuously	Participating in Daily Stand-ups as a full participant. Resolving impediments to progress.

Database Administrator

The main goal of the database administrator role is to support the creation of database projects as well as deploying database project changes to production. This is in addition to the database administrator's traditional role of performing day-to-day administration and maintenance of database servers.

The database administrator is not expected to participate in the Daily Stand-up or retrospections unless a member of the team.

Standard Work

Accountability	Which involves...
During Planning	Advising the database developer, identify enabling and systematic evolution issues that must be addressed for data environments. Helping to identify all skills necessary to achieve the iteration commitment.
Daily Work	Focusing efforts on delivering the highest priority stories and tasks that pertain to the database and data environment while identifying dependencies.
Iteration Closing	Participating in demonstration of product to the Product Owner, Product Manager and other appropriate stakeholders.

Database Developer

The database developer's main goal is to implement all database development tasks within the planned time frame. The database developer is also responsible for cost estimation, supervision of feature implementation, and providing database expertise to developers.

Database resources are often quite limited. In Lean-Agile projects, a lead developer often assumes the role of a "surrogate database developer/analyst." This role then acts as the primary intermediary between the team and the database developer, collecting requirements, managing the relationship, and reporting progress. This approach creates higher quality and richer communication with the limited resources.

The database developer or surrogate participates in the iterative database development life cycle, along with the database administrator and the team members.

The database developer is not required to participate in the Daily Stand-up, but may participate in retrospections. The database developer may play an advisory role in story planning.

Standard Work

Accountability	Which involves...
During Planning	Advising the database administrator and Technical Owner. Identifying enabling and systematic evolution issues to address for data environments. Helping to identify all skills necessary to achieve the iteration commitment.
Daily Work	Focusing efforts on delivering the highest priority stories and tasks that pertain to the database and data environment while identifying dependencies. Ensuring all database development tasks are implemented within planned time frame. Providing cost estimates. Providing expertise to developers and analysts.
Iteration Closing	Demonstrating product to the Product Owner, Product Manager, and appropriate stakeholders. Possibly participating in retrospections.

A Team of Peers

The team drives the product from a tactical perspective. It is autonomous and self-organizing and is accountable to the Product Manager for committing to and delivering product increments within the duration of the iteration.

The team estimates the size of backlog items, makes design and implementation decisions, and transforms stories into high quality software assets.

The team works with the Team Agility Master to track its own progress, raise awareness of impediments, and devise options to maintain forward progress.

The team is responsible for establishing processes and standards, continuously improving them, and then following those processes and standards.

Here are some foundational Lean-Agile principles about the team.

Principle	Description
Equal authority	The team members are peers with a common focus, shared responsibility and open communications. Each role is accountable for a specific share of the quality of the overall solution with special emphasis on horizontal flow of value.
Roles and skills	Team membership is based on roles and skills rather than positions. Each person has a primary role and periodically may occupy other roles on a team depending on their skills and the needs of the release.
Delivering value	The team works from a common vision for the product and for releases, as laid out and prioritized by the Product Manager. The goal for each release is to deliver the minimum set of releasable product that will help the Business. Teams commonly swarm on stories to focus on one or just a few stories at a time.

Principle	Description
Frequent feedback	The team's activities are set up to get feedback early and often from all stakeholders to help them learn what the requirements are. The idea is to work on what is known rather than making customers speculate about what they do not yet know. Alternatives and changes are much easier earlier than later.
High-bandwidth communication	Rich communication is preferred to encourage fidelity and speed in decision-making and to minimize documentation that is truly non-value-added. Teams use tools to promote this communication.
Deliver product	The primary focus of the team is a working product that delivers the features required by the Product Manager. That is what they commit to and they are expected to deliver.
Team owns their process	The team is responsible for its process. Teams employ their knowledge about their current situation and about what is required to do their work, respecting mandatory enterprise standards. The team agrees to use and improve its own processes. If something is not working, the team should stop, improve, and then go forward.

Swarming and Teamlets

The team's focus is on building and delivering product. People are expected to bring their skills to bear, regardless of role, to work on whatever is currently active. For example, a Business Analyst might volunteer to help with acceptance testing of a piece of functionality.

This idea of people joining to work together based on their skills rather than their roles is called "swarming" and is an essential team skill. When people agree to swarm on a work item, they form a "teamlet." The teamlet may assign a "Story Captain" to help the teamlet stay on track.

Teamlets are generally formed at the Daily Stand-up, when the team reviews progress that has been made, and decide what needs to be worked on next. They are fluid, forming and disbanding based on the requirements of the work items at hand.

Swarming Rules	Description
Focus on only one story at a time	Within an iteration, teams should have only a few stories open at a time because the focus is on burning down stories. Do not dissipate energy by focusing on too much at once.
The swarm is the priority	While individuals may work on other tasks, their priority should be the stories they swarm on.
Each team member with the skill should join the teamlet	Each team members who has skills to contribute to burning down a story, and has capacity is expected to join in the teamlet, even if it is not exactly in their job description to do so.

Part IV: Concept to Consumption

In Leanban, work proceeds from the initial concept through development, and on to delivery when the customer can consume the product. Only then is the value realized.

This section provides overviews of the following:

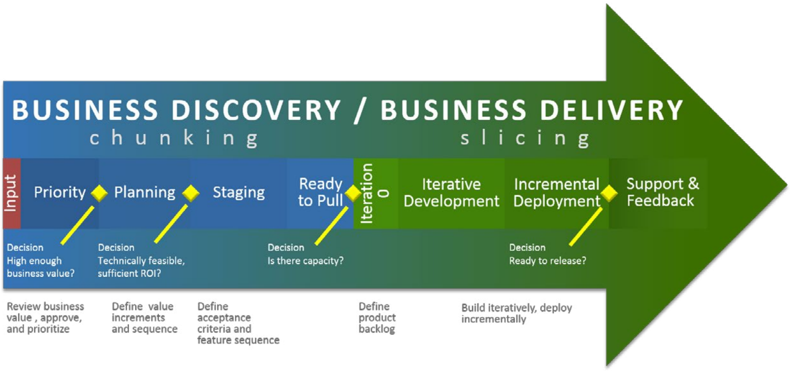
- The activities of discovery and delivery with an emphasis on team activities for delivery
- The progressive unfolding of requirements and the elements involved: Capabilities, Minimum Business Increments, Features, Stories, and Tasks
- Estimation and velocity

In preparing for battle, I have always found that plans are useless, but planning is indispensable.
– Dwight D. Eisenhower

Overview: Business Discovery and Business Delivery

In Leanban, planning and re-planning are done continuously along the entire value stream from initial vision to support. Lean-Agile principles help Scrum to focus this by driving all effort to delivery Business value.

The goal is to plan as much as, but no more than, the Business is clear about, to avoid guessing, and to incorporate lessons learned and customer feedback. This avoids building features that are not (yet) needed or building features with the wrong behavior. It recognizes that users often do not know what they need until they have something concrete to work with. It allows plans to adapt to changing market conditions and business priorities. And it allows teams to deliver sooner something that the customer finds valuable rather than having to wait.


BUSINESS DISCOVERY / BUSINESS DELIVERY
chunking
slicing
Input
Priority
Planning
Staging
Ready to Pull
Iteration 0
Iterative Development
Incremental Deployment
Support & Feedback
Decision High enough business value?
Decision Technically feasible, sufficient ROI?
Decision Is there capacity?
Decision Ready to release?
Review business value , approve, and prioritize
Define value increments and sequence
Define acceptance criteria and feature sequence
Define product backlog
Build iteratively, deploy incrementally

Phases of Business Discovery and Business Delivery

Lean-Agile describes the value stream of discovery and delivery in eight phases with four gates. These are illustrated in the arrow figure.

Note: Moving from Business Discovery to Business Delivery involves progressive unfolding of requirements as more is learned. See "Progressive Unfolding of Requirements" on page 101.

The outcomes for the stages are described in the following table.

Stages and Outcomes of Discovery and Delivery

Stage	Responsible	Outcomes
Priority	Business Sponsor Value Stream Owner Stakeholders	Business Value established Approved for planning Prioritized (relative to other approved items) Product Manager identified Book of Work updated Product Vision
Planning	Business Owner Product Manager Business PMO Technical Owner Leads	MBIs defined Approved for staging Prioritized and sequenced based on ROI (relative to other approved items) Product Owner assigned Business SMEs identified Book of Work updated
Staging	Value Stream Owner Product Manager Business SMEs Business PMO Technical Owner Leads	MBI(s) refined, sized, sequenced based on ROI Features defined and sequenced Acceptance criteria defined Business backlog established

Stage	Responsible	Outcomes
Ready to Pull	Product Manager Product Owners Business PMO	Product Manager available Product Owners available Business backlog defined Required resources available Team(s) with available capacity are identified
Iteration 0	Product Manager Product Owners Business PMO Representatives for all skills required Team Agility Masters	Program backlog established for all work: Business and technology Feature sequence refined including technical dependencies Initial top-line in story points High level architectural specified Done: Criteria, tests, and documentation Logistics: Team(s), visual controls, meetings, coordination
Iterative Development	Product Manager Product Owners Representatives for all skills required All team members Team Agility Masters	Iteration backlog established of stories committed by the teams for next two weeks Right-sized stories defined, sequenced, sized in story points Tasks defined in hours Initial iteration Burn-up and Burn-down charts

Iterations

Agile uses an iterative approach to product development: building product features in 2-4 week increments (iterations). Iterations begin after the Iteration 0 and release planning have been done. They follow a routine cycle of activities:

1. Conduct Iteration 0, if needed, to set up for the iterations
2. Plan the Iteration, estimating and prioritizing the work to be done
3. Daily execution:
    - Daily review and adjustment
    - Develop and test continuously
    - Handle impediments as they arise
4. Demonstrate the product at the end of the iteration
5. Release to customers as appropriate
6. Hold a retrospection on the iteration to learn how to improve
7. Incorporate new requirements
8. Prioritize and estimate work for the next iteration based on all requirements
9. Start again

Checklists

Checklists for Iterations begin "Starting a Lean-Agile Team Checklist" on page 157.

"Iteration 0"

In most projects, before the first iteration of a release can be started, it is good to do some "pre-work" to set up stories in the iteration, environment and tool configurations. This is called "Iteration 0" and may take up to 1.5 times the length of the other iterations.

Purpose

Iteration 0 meetings are a conversation between stakeholders and the team before the first iteration of a release. This ensures the team is ready with people, process, and tools in order to begin making progress in iterations. It is a time for conversation about scope and requirement. They will revisit these during subsequent planning meetings.

Prerequisites

These should have been done much earlier:

- Business Prioritized
  - Business value has been established and prioritized
  - Capabilities are sequenced and approved.
- Business Planning
  - MBIs are defined.
  - MBIs are prioritized and sequenced based on ROI.
  - Technical feasibility has been assessed.
  - MBIs are sized according to T-shirt size.
  - MBIs are approved to continue.
  - Business SME's are identified.
  - Possibly, Product Owner is assigned.
- Business Staging
  - MBI(s) have been refined.
  - Features are defined and sequenced.
  - Acceptance criteria are defined for each feature.
  - Business backlog has been established.
  - Features are approved and sequenced to continue based on ROI (relative to other approved items)

These should have been done just before Iteration 0:

- Ready to Pull
  - Product Manager and Product Owner are available
  - Business backlog is defined: features prioritized and estimated and high level architectural milestones are specified
  - Team(s) are identified and capacity is available
  - Required resources are available
  - Iteration 0 Facilitation plan is finished
  - Carry over stories from previous iterations, if any, are identified
  - Team velocity over past increments, if known.

Approach

In Iteration 0, the whole team does the following:

1. Introductions are made (even for well established teams).
   - Who is on the team (roster and who is required)
   - The resources required for the team(s)
   - Review of team environment: workflow, visual board, and policies for stories
2. Identify technical components. Input them into team backlogs (technology features and/or stories)
3. Review and define production documentation and artifacts. List the current production documentation maintained.
4. Review and define high level architecture and design.
5. Finalize feature sequence by Business value and technical discrepancies.
6. Identity any critical areas and dependencies that must be addressed for progress to be made.
7. Review and define complexity factors for relative sizing in story points.
8. If there are problems or impediments, rank them, prioritize them, and identify stories to mitigate them.
9. Confirm all items are complete in the Iteration 0 Checklist.

Outcomes

The outcomes of this practice include:

- Product Manager has approved initiation of Iteration 0 activities.
- Action items for the Product Manager action items.
- The team is prepared with skills and tools to begin work on the iterations.
- Stakeholders and team have communicated effectively about the objectives of the release and what is expected.
- The team builds a sense of ownership for their processes.
- Impediments to progress that are outside of the team's control are naturally escalated to management and others who can do something about them.

The team will be able to answer these questions:

- How do we know we are done planning?
- What documents do we have?
- What do we know?
- How do we know we are ready for the first iteration?

Checklists

See "Iteration 0 Checklist" on page 159 .

Iteration Planning

Iteration planning determines the work that the team commits to be completed in the iteration by adjusting the predicted velocity and managing the number and priority of assigned, deferred, and/or new stories.

Purpose

The length of an iteration enforces a common cadence and creates boundaries for work. With a fixed iteration length, the Product Owner and the team adjusts scope and cost to fit what can be done in the iteration.

At the beginning of a project, there may be an initial allocations of features or stories to increments based on value or dependencies. However, as the project unfolds, these are revisited at the beginning of each iteration to ensure that the scope of the iteration is reasonable given the current team capacity and that it includes high value work based on the changing environment at the stakeholder and business value level.

Prerequisites

Inputs to Iteration Planning include:

- Backlog of user stories, preferably elaborated (refined, sized, and with initial-acceptance criteria)
- Carry-over stories from previous iteration
- Team velocity over past increments
- The Team Agility Master has prepared to facilitate the meeting
- Completed Iteration 0

Approach

In iteration planning, the whole team does the following:

1. Review and understand iteration goals.
2. Review current velocity and adjust as needed. Take into account holidays, time off, corporate events, team member availability, and change in complexity of or experience with stories.
3. Review stories already planned for the iteration, Discuss, size, split as needed, and ensure stakeholder-clarified acceptance criteria.
4. Remove or add stories within velocity constraints.
5. Elaborate any new stories as before.
6. Identify and account for any dependencies with other teams (may require interactions with other teams or the system architect.
7. Ensure selected stories are within highest Feature/MBI priority.
8. Rank stories in priority order (rules for prioritization may vary depending on status of project or environmental changes).
9. Create and estimate tasks within each story.
10. Identify lessons from previous iterations that the team wants to incorporate into this iteration.

Variances

Here are variances to iteration planning:

- Tasks and estimations may happen in a follow up session.
- Individuals may task and estimate.

Iteration planning should be done as a collaborative exercise. Collaboration establishes a reasonable set of stories to implement within the increment given current team, project and stakeholder status.

Add, Revise, Remove Requirements

As customers gain experience with the product, they learn more about what they need. They may development new requirements, may decide they no longer need something they had asked for, or may decide that something is more important or less important than before.

At the end of each iteration, based on what the customer has learned from seeing and using the product, the Product Manager (acting as advocate for the customer) can:

- Add new requirements
- Revise existing requirements
- Remove existing requirements

After these adjustments are made, the collection of requirements are then prioritized and used as input for the next iteration.

Outcomes

Iteration planning results in a clear scope of work for the next iteration must be feasible for the current team.

Issues and Considerations

Topic	Discussion
Lack of a known velocity	When the team's velocity is not known, populate the backlog using size rather than velocity.
Testing was not done	If testing has not been finished by the end of the iteration, then the team cannot know how much testing resource is available in the next iteration. Move testing early in the iteration.
Team composition	If the team composition changes, then they cannot know what their capacity (velocity) will be for the upcoming. Treat their velocity estimate as a guess, just like it was at the first iteration.
Frequent interruptions	Establish a process to handle interruptions from management, other projects, etc. so that teams are not blind-sided by interruptions in the middle of an iteration. Perhaps schedule time at the end of the iteration for other work. If interruptions become too frequent, institute a single point of contact (an interruption bottleneck or a "bouncer") to limit the impact. The Team Lead or Team Agility Master can perform this function for the team.
People are not available	If all team members are not available to plan together, the plan will almost certainly be incomplete, inadequate, and erroneous. Adjusting during execution is wasted effort.

Checklists

See "Iteration Planning Checklist" on page 161.

Iteration Execution

The iteration is when the development work is done. The goal of the iteration is to complete the stories that have been agreed to and produce perfected and demonstrable product at the end.

Purpose

During the iteration, the team is focuses on its work and should be relatively free of interruptions. They should be identifying impediments, adjust their work processes based on what they are learning.

At the end of the iteration, the team demonstrates the product, reflects on lessons learned and how they need to improve process, and is ready for the next iteration.

Prerequisites

Before an iteration can begin, the iteration backlog must be populated, the project board and team environment set up and cleaned up; and the team commits to the plan.

Approach

In iteration execution, the whole team does the following:

- Controlling WIP
- Starting on stories
- Selecting tasks
- Updating the story board
- Completing stories
- The Daily Stand-up
- Product Demonstration
- Iteration Retrospective

Controlling WIP

One of the biggest factors in slowing down work, is to have too much of it going on at the same time. It leads to excessive task switching, forgetting important information, stress, and many other documented sources of delay. The practice of controlling the amount of Work-in-Process provides a way for people to focus on just the work they can effectively service at any given time.

For any system, there is an upper limit to the number of work items that can be processed effectively at any given time. This is called the "WIP limit." Generally speaking, the lower the WIP limit can be made, the greater the throughput of the system. A lower WIP limit means that the system is becoming more effective at processing small amounts of work at a time. This increased focus accelerates the work and improves flow.

The goal is to reduce the effective WIP limit. The more mature a system is, the lower its WIP limit can be made.

An approach to controlling WIP

Here is an approach to controlling WIP:

1. Prepare to become able to lower the WIP limit by removing delays in the process (using Value Stream Mapping) and reducing batch sizes.
2. Set up a visual control to make it clear how many items a person or a group of persons (e.g., "the testers") are being assigned to work on at the same time. A kanban board is good for this.
3. Set policy so that workers will not be pressured to exceed their WIP limits. Set up means to highlight when WIP limits are exceeded. A kanban board is good for this too.
4. Gradually decrease the WIP limit over time, to reveal bottlenecks in the system and remove them. This will increase overall team throughput.

Outcomes from controlling WIP

Over-tasking is one of the biggest drains on productivity (Weinberg, 2011). Over-tasking leads to bottlenecks. Bottlenecks are work steps in the value stream in which people cannot keep up with the amount of incoming work, so the people performing the bottleneck step fall behind and the work piles up in the work step before the bottleneck. This often happens in QA or testing step, for causes such as the testers have been divorced from the developers, too many errors are allowed during development due to process or tooling issues, or because too few people or too primitive tools have been allocated to testing. As bottlenecks are revealed, the organization/team should reduce them by a combination of making better use of the bottleneck resource (for example having developers and testers work together so fewer errors reach testing) and process improvement that reduces dependency on the bottleneck.

Starting Stories

When selecting stories, it is important that the amount of Work-in-Process (WIP) matches the capacity of the team. Having many stories open runs the risk of having many stories open at the end of the iteration and reducing team efficiency. This should be avoided as much as possible. Having stories that are in process at the start of an iteration causes many problems – not the least of which is you are never sure how much time it will take to complete them.

When defining a story it is useful to ask the question, "How will I know I've done that?" Getting an answer to this question gives the developer an example of what the story needs to accomplish. This helps the developer check their understanding of the story. Not answering this question will often result in work being done that will later be discovered to not have been what the customer asked for.

If the Product Manager “does not have time” to answer this question, try to come up with the best answer you can and ask if that is what is meant. This requires much less time on their part but will still give an example.

Selecting Tasks

Focus on tasks that will help to complete open, in-work stories.

Updating the Story Board

Team members should themselves update the story board for the following events:

- Open story (typically indicated by having a task in process)
- Active task (typically indicated by having a task in a different place than it started)
- Impeded task (typically indicated by placing a sticker on it or moving to an impeded row or column)
- Completed story waiting for customer validation (indicated by moving it to a completed row or column)
- Done-Done story (indicated by moving it off the board)

Completing Stories

Stories are completed when they are done-done-done. This means:

- The system runs as expected.
- The system is verified by running unit tests on a common machine.
- The system is validated as being of deliverable quality with functional tests.

Level	Test	Prod Mgr	PO	Cust-omer	Tester	Dev
Capability	Business Value / ROI realized	X				
Minimal Business Increment	Incremental Business Value / ROI realized		X			
Feature / Scenario	Completed and accepted		X			
User Story	Completed and accepted		X			
Customer / User	Customer acceptance test			X		
Functionality	Functional				X	
System	Integration					X
Component	Unit					X

Checklists

See "Iteration Execution Checklist" on page 163.

Tip. Protect the team from wasted time

Meetings are a big time waster in many companies. Team members should not have to attend meetings called outside the Scrum team unless the meeting is absolutely required (such as mandated employee management meetings). Meetings whose purpose is to report status can be eliminated through the proper use of visual controls and Gemba walks and through the end of iteration demonstrations.

If a team needs to be represented at a meeting, the Team Agility Master can attend.

The Daily Stand-Up

The Daily Stand-up is the activity of the team to decide together each day what team members will focus on. It is quick, focused, and highly collaborative. Daily planning is the responsibility of the whole team, each member talking with the rest of the team.

Purpose

The purpose of the Daily Stand-up is to communicate progress, identify impediments, and create teamwork. It is *not to solve problems.* Problem-solving can be done afterwards, offline.

Prerequisites

Before the Daily Stand-up can begin, the status of stories, tasks, and impediments must be updated on the team's project board.

Approach

Daily planning involves a brief stand-up meeting involving the Team Agility Master and the team, both technical and Business. The Product Owner should try to attend whenever possible, even if it is a call-in.

The goal of the Daily Stand-up is to ensure consistent progress in completing the work required for the current iteration. Priority is on completing in-work stories rather than starting new stories.

Here are the steps of the Daily Stand-up

1. Team gathers in designated area – all standing as they are able.
2. Team quickly scans the work completed for the previous day.
3. Review the most important work for the day is.
    - Team identifies the most important work. Team Agility Master asks who's working on it.
    - Discuss if the complexity of work changed and how much time is left on your tasks
    - Repeat until all team members have pulled work.
4. Are there any impediments. Team Agility Master captures them if identified. Team Agility Master ensures someone owns the raised impediment.

Ground Rules

The members agree to follow the simple rules of the Daily Stand-up.

Rule	Description
Be consistent	The Daily Stand-up meets every work day at the same time and in the same place.
Show courtesy	Team members show professional courtesy: show up on time, participate, and listen.
Be brief	The Daily Stand-up typically lasts for 15 minutes. Standing up reinforces brevity. Extra discussions and problem-solving is conducted after the meeting, when there is more time.
Hold a team-wide conversation	The Daily Stand-up is for the team's benefit. Each team member is expected to speak and speaks to the whole team. The team is not reporting to the Team Agility Master. The Team Agility Master is there to help facilitate the conversation but not to lead the session.
Answer all three questions	Each team member answer three questions: • What have you done since last meeting? • What will you do before next meeting? • What is blocking or slowing you down? Team members can raise issues and obstacles but not propose solutions.
Review impediments	The Team Agility Master reviews impediments and status.
Swarm	If you have the skills to help with a task and you do not have something else to do, you should volunteer to join the swarm.
Finish work	The priority is to complete stories that are in-work before starting new stories.

Outcomes

Stand-up meetings are efficient ways to ensure that all the developers and other stakeholders understand the value of what they are doing in relationship to the business/customer, and remain focused on the most valuable work. They enable early identification of issues so that they can be addressed before additional technical debt is accrued. There is an element of team building that comes from the interaction of peer pressure and peer support.

Issues and Considerations

Topic	Discussion
What if the team falls behind?	The iteration plan is just that: a plan. When the team determines it is falling behind, it should not just hope it will catch up. It should let the know so he/she can prepare to remove stories from the iteration if necessary. Be hesitant to open any new stories until those in process are done (a good practice anyway) in case not all of the stories get completed.
What if the team goes faster than expected?	If the team looks to complete its iteration ahead of time it should tell the this so he/she can prepare to add some new stories to the iteration if they can be completed.
What if someone identifies an additional task in a story?	When an additional task is discovered, you must determine if this will add time to complete the story. Sometimes it does, sometimes it does not. If the additional effort is insignificant, there really isn't much you have to do. However, if the extra time now puts a story in jeopardy, it is important to ask the if the story is still worth the extra cost.
How does the team make sure it is ready for the next iteration?	A little look ahead is required in order to be sure enough analysis has been done to be able to make good estimation and planning during the iteration planning day. A certain number of stories should be analyzed prior to the iteration to prime the pump.

Checklists and Ground Rules

See checklists and ground rules for the Daily Stand-up page 164.

Tip. One of the most important jobs of the Team Agility Master is to reflect on how the iteration is going. This includes looking for tasks that are taking too long.

Tip. The team may stop an iteration if they discover a significant barrier that cannot be resolved. The Product Owner and team must decide what to do and re-plan.

Carry-Over Work

Estimates are rarely perfect, so it is possible, even probably early on, that there will be some work remaining incomplete at the end of an increment. The team intentionally decides what work is recast as smaller tasks that are complete, what is redefined as additional tasks deferred into the backlog or the next iteration, and what work is dropped from the project scope.

Carry-over work should be addressed on the last day of each iteration.

Example: Web Self-Service

The Business identifies several actors that will use a new Web Self-Service Account Management system, including: Home Owners, Renters, and Business Owners. The Business decides that servicing Home Owners would be the most valuable and should be released first, followed by other actors.

Providing Account Management to Home Owners is part of the capability for Home Owner Self-Service.

Looking at this, the planning team composes Home Owner with the stories for Account Management to identify four scenarios: Home Owner Create Account, Home Owner Read Account, Home Owner Update Account, and Home Owner Delete Account (involving the support desk).

Looking at these, the Business decides that the first three are essential. Since Home Owner Delete Account requires help from the support desk, they judge that it is not essential for the computer system to do it automatically. They can continue using support desk personnel.

The Home Owner Create Account, Home Owner Read Account, and Home Owner Update Account comprise an MBI.

Tip. Sometimes, the customer may be satisfied enough with the product that she decides that it can be released as is so that the Business can begin using it. If everything else is ready, release it.

Approach

Identify which category unfinished work belongs in.

For features/stories not started or undeliverable, do the following:

1. Clone incomplete story with its sub-tasks.
2. Put new cloned story in future iteration or backlog, or drop from current scope and accrue as technical debt.

For features/stories partially completed and deliverable with acceptable work/rework, do the following:

1. Identify work completed.
2. Create new story for work completed with new acceptance criteria and appropriate story points.
3. Reassign completed technical tasks from old partially, completed story to new story just created.
4. Accept new story and sub-tasks within current iteration.
5. Create new story for incomplete portion of carry over story with new acceptance criteria and appropriate story points.
6. Reassign incomplete sub-tasks to new story.
7. Place in backlog or next iteration.

Guidelines

Everyone participates in reviewing carry-over work. The Team Agility Master is responsible for ensuring this activity is completed by the last day of the iteration.

Outcomes

The benefits of looking at carry-over work include:

- Remaining work is always accounted for. It is rescheduled for the next iteration, backlogged, or dropped from current scope.
- Gaining credit for the partial work completed on accepted stories.
- Sustaining a rational and intentional approach for rescheduling remaining work.

Product Demonstration and Review

At the end of each iteration and at the end of the release, the product is demonstrated to the key stakeholders, the Product Manager, the Product Owner, the Team Agility Master, and the whole team. If possible, try to include some actual users as well.

The Product Demonstration is not just a presentation but is a conversation between the participants about what has been done and what to do next.

Purpose

Demonstrations are the best way to keep stakeholders up to date on progress and also to obtain feedback on the developers' interpretation of features and stories. The cadence for demos need not be synchronized with the development or delivery cadences. Demos are usually scheduled near the end of an iteration, but they can be conducted at any time a feature or story is ready for evaluation and the stakeholders are available. It is much more important to demo early and get corrective feedback early, than demo late and find significant disconnects that create technical debt in terms of necessary rework.

Prerequisite

The essential prerequisite is that everyone is available for the demonstration: the key stakeholders, the Product Manager, the Product Owner, the Team Agility Master, and the whole team. If possible, try to include some actual users as well.

In addition, the following must be ready:

- Increment plan
- Finished functionality
- Reports showing completion (including test results)
- Demonstration script
- The Team Agility Master is prepared to facilitate the meeting

Approach

The Product Owner owns the demonstration. Oftentimes, the Team Agility Master facilitates the meeting. The demonstration is informal.

The walkthrough should cover more than simply the technical aspects: discuss what has been done with the people skills and training and the processes required to realize the feature. Adjustments to priorities (new, revised, removed) in order to achieve value in the next iteration.

Here are the steps:

1. The Product Owner gives an introduction to the demonstration. This includes:
   - The goals of the iteration
   - The commitment list and the stretch objectives
2. The team gives the demonstration. This includes:
   - A summary of what will be demonstrated
   - What functional tests were used to validate functionality
   - A demonstration the functionality
   - A review of what was added/corrected since the last demonstration
3. The Product Owner accepts the items,
   - Declaring which team-commitment items are complete: code-complete, integration-complete, verification-complete, and validation-complete.
   - Declaring which iteration goals were met
   - Accepting or rejecting the calculated team velocity (the story points of completed items in the team commitment list)
4. The team and the Product Owner agree to return incomplete items to the team backlog (see "Carry-Over Work" on page 90).

Issues and Considerations

Topic	Discussion
Useful stakeholders are not present	If stakeholders are not present, you cannot get proper feedback.
Managing expectations	When the product is only partially complete and not ready to be released, the stakeholders have to be very clear that it is not ready.
It is good enough	If the customer is satisfied enough, consider releasing it... *use it*!

Ground Rules

See "Iteration Implementation Checklist" on page 164.

Iteration Retrospective

Iteration Retrospectives are the structured reflective practice to learn and improve based on what has already been done. The purpose of retrospection is to examine the process the team uses, to build team commitment, and to transfer knowledge to the next iteration and possibly to other teams.

Retrospections must be done at the end of every iteration. A briefer version, the After Action Review can be done at any time, whenever there is value for the team to stop and learn from what has been done and change while it still helps work.

Purpose

During each iteration, during every activity, the people doing work should help each other think about quality issues. Be explicit about what is required without trying to solve the problem (which is the responsibility of people who will receive stories or tasks to do).

Improving quality involves discipline and mutual commitment. Lean techniques such as retrospections and After Action Reviews, keeping a clean environment, testing, using patterns, and attending to code quality, help but only when the team uses them regularly.

One of the principles of Lean-Agile is to *create knowledge*. With the permission of the team, the Team Agility Masters should be sharing lessons (sanitized if need be) with fellow Team Agility Masters through their community of practice. Many organizations find it helpful to include knowledge-sharing as part of Team Agility Master and team performance assessments.

Prerequisites

Team availability. Everyone who was involved in the iteration be available. The reason is that everyone has some viewpoint about what happened. If someone is missing, an insight will be lost.

Facilitation plan. The Team Agility Master has prepared to facilitate the meeting.

Approach

At the end of each iteration, the whole team conducts a retrospection, facilitated by the Team Agility Master. The key question is, "*If we could do it again, what would we keep doing and what would we improve?*"

To answer this, the team discusses the following:

- How well did we do in the iteration? Did we meet the iteration goals?
- How is the project progressing?
- What were your successes? (Ask everyone.)
- What processes / techniques would be good to use again?
- What were your challenges? What could have gone better? (Ask everyone.)
- What processes and techniques need to be improved?
- How is the team being? Are they raising impediments? Are they adjusting?
- What impediments are still present?
- On a scale of 1-10, what is the score of this iteration?

Some organizations boil these down into three broad questions: “What?”, “So What?”, and “Now What?”

The Iteration Retrospective should result in one to three stories for the next iteration reflecting a "vital few" improvements to the process.

Also, review "Continuous Improvement Before, During, and After" on page 133 and "Impediments to Progress and Quality" on page 134.

Guidelines

Guideline	Description
Facilitated	The Team Agility Master facilitates the discussion and does not solve problems.
Whole team participation	Everyone who was involved in the iteration should be present at the retrospection. Everyone speaks because everyone has an insight that may help foster understanding.
Blame-free	The goal is process improvement, not blame. Be honest about what happened. Critiques are allowed without recrimination. Use the normal rules for brainstorming. Uncover what actually happened.
Vital Few	While a team may generate a lot of ideas, have the team pick a "vital few" that offer the greatest opportunity for near-term improvement. For each one, create a story and assign it to an iteration. A retrospection is successful if it generates 2-3 stories focused on process improvement.

Facilitator Notes

The retrospection facilitator is responsible for the following:

- Inviting participants
- With permission, inviting people from other projects to observe so that they can carry lessons back to their teams, hearing subtleties not present in written reports.
- At the beginning, asking people to introduce themselves and their role (even teams that know each other)
- Reviewing the objectives for the iteration
- Creating an atmosphere of openness and don't be afraid to ask the unasked questions.
- Clarifying the distinction between facts and opinions
- Ensuring a blame free environment.
- In a big meeting, asking someone to take detailed notes
- Asking who else might benefit from these lessons learned

Issues and Concerns

Topic	Discussion
Growing stale	At some point in a team's life, retrospections become rote, stale. It ceases to be seen as useful to the team. Whenever you sense this, it is your responsibility to point it out. The team must consider together how to shake things up, to see how they can make it relevant again. Consider looking more widely or deeply about processes, based on lean principles.
Too many suggestions	Focus on the vital few. Track these as stories for the next iteration so that the team can see value being produced.
Too few suggestions	Teams often think too small. They are constrained by assumptions about what they are allowed to do.
Complaining	It is common for improvement events to devolve into whining sessions, complaining without an intention to do better. The facilitator can allow a little time for this but then must take strong action to head this off.
Lack of a properly trained facilitator	Retrospections must be facilitated by a trained facilitator who is willing to lead without trying to solve problems.

Topic	Discussion
Inviting others	Retrospectives must be safe environments for participants. This is an important part of the facilitator's job. This will depend on the maturity of the team and whether confidentiality is required in order to discuss a particular issue. In the early stages, it may be that membership should be limited to team members themselves. Deciding whether or not to involve outsiders in the retrospective, following this rule: *Will the additional person contribute to learning amongst the team and within the organization?* Should the Product Owner come to the retrospective? Yes, if... • The Product Owner has perspectives about the iteration that need to be shared • It will help to build relationship between the Product Owner and the team • The Product Owner is not involved in evaluating team members • The team feels comfortable with the Product Owner being there • The Product Owner has time (Product Owners are often very busy) As the team gains confidence, it could be that asking other Team Agility Masters to observe certain retrospectives. This is especially useful to help them carry lessons back to their teams. They get to hear subtleties first hand, ideas not present in written reports. But only if the team feels comfortable with it!

Outcomes

The benefits of the Iteration Retrospective include:

- A focused set of improvements that can be immediately put into practice.
- Building a habit of improvement. The team as a whole and individuals on the team get in the habit of looking for ways to improve.
- The team builds a sense of ownership for their processes.
- Impediments to progress that are outside of the team's control are naturally escalated to management and others who can do something about them.

Progressive Unfolding of Requirements

Lean-Agile thinking, analysis involves a progressive unfolding of levels of abstraction from capabilities to MBIs to features to stories and specific tasks (see "Overview: Business Discovery and Business Delivery" on page 70). Each level of abstraction describes with more certainty what the software product development team needs to do, when they need to do it, and how they will know when they have succeeded in order to deliver value to customers and to the Business. At each level of detail, it becomes easier to estimate the work involved.

Planning and analysis are activities the team uses to foster *communication* between team members, stakeholders, and customers.

Planning and analysis cannot be separated cleanly in Leanban. Each drives the other and each provides feedback. Teams constantly re-plan and add detail based on what has been learned. Give the team just enough so that they can start working on delivering value. *Never speculate* about requirements.

Lean-Agile thinking, analysis involves a progressive unfolding of detail from product vision to features to stories and specific tasks. Each level of detail describes with more certainty what the software product development team needs to do, when they need to do it, and how they will know when they have succeeded in order to deliver value to the Business. At each level of detail, it becomes easier to estimate the work involved.

Planning is the responsibility of the Product Manager who represents the voice of the customer and the voice of the Business to prioritize the work to be done. Planning is done all throughout product development:

- Product visioning
- Release planning
- Iteration planning
- Daily planning with the Daily Stand-up

Analysis involves writing capabilities, MBIs, features, stories, and tasks, the essential elements teams use to do work.

The following sections describe the processes of unfolding at each stage. It concludes with an approach to assessing how well requirements are understood.

Progressive Unfolding: Choosing a Requirement Format

Requirements have been written using many formats. In most cases, the "User Story" format is often best. This section describes how to write in the "User Story" format. It also offers guidelines for choosing another format.

Inputs

Inputs to this practice include:

- A requirement to be decomposed
- Information about the requirement to be decomposed: Level (Capability, MBI, Feature, Story), details about the value being developed (customer, Business), and the context in which it is being developed (interfacing systems, technology constraints, etc.)

Approach

The "User Story" format can be used to capture a single scenario, an example of some behavior of the feature, or even reminders or triggers to have a future conversation.

Choosing the best format for a requirement is based on judgment. Judgment comes from experience with requirements decomposition as well as having an understanding of what kinds of information will be needed by those implementing the next level of detail (either requirements at the next level of decomposition, or the software or human procedures that implement a team-level requirement).

Start by assuming the User Story format will work and try to write the requirement using it. If the User Story format does not adequately capture and communicate the requirement, consider alternate formats.

If the User Story format is not capturing every important aspect of the requirement, someone with experience at the next level of detail (or the one below it) should be consulted to find out what they will need to know to do their work successfully. Then pick or develop a format to replace or supplement it.

Tools and Techniques

Tools do not usually influence which requirements format to use. However, tools can help keep visible the context of a given requirement. This in turn can suggest what information needs to be recorded in the requirement. Consider a mind-mapping tool or a drawing tool that can be used to draw flow, System-of-System, and other relationship diagrams.

General Template: User Story

The essential attributes of a requirement written in the User Story format are:

- **ID.** An identifier used to refer to the story.
- **Description.** The intent of the story. Here is the most common template for a User Story:

  As a <*user*>*,* I want <*capability*> so that I get <*business value or functionality*>

  where:

  "<user>" means the role played by the person to whom the new value will be delivered. For instance, "As a bank teller," might begin a user story for a software capability that eases an aspect of the teller's work

  "<capability>" means what the system will do for the user. For instance, "I want to be able to scan in my customer's debit card number"

  "<Business value or functionality>" means what the new capability does for the user. For instance, "so that I eliminate the time typing in numbers as well as redoing an entry because of a typing mistake; I currently spend an average of 2 minutes extra work on each transaction because of these activities."
- **Validation Strategy.** Every story must have criteria so that the team can know, "Is this story done?"
- **Parent and Related Stories.** Where did the story come from and what else is it related to?

Alternatives

These are some of the main alternative formats used for requirements:

- **Use Case.** To capture more information about how the stakeholders are served, the before and after conditions of the system fulfilling the requirement, alternative and essential actions based on differing conditions when the user invokes the functionality in the requirement, etc. Use Cases can be used at any level, including capabilities expressing Business or stakeholder goals (these are sometimes called "Business Use Cases").

  The "Fully Dressed" Use Case format, based on the work of Alistair Cockburn, is:

  - **Title.** "An active-verb goal phrase that names the goal of the primary actor"
  - **Primary Actor.** Can be a user, or another system
  - **Goal in Context.** Statement of the requirement's goal and the conditions around the system when its functionality is invoked
  - **Scope.** Of the system into which the Use Case is being implemented
  - **Level.** Of abstraction of the "goal" or requirement
  - **Stakeholders and Interests**
  - **Precondition.** Assumed condition of the systems around the target system
  - **Minimal Guarantees.** Critical interests protected in any situation
  - **Success Guarantees.** State of the world around the target system, if the goal is achieved
  - **Trigger.** Event that kicks off the Use Case
- **Main Success Scenario.** Interactions between primary actor and target system, from trigger to goal
- **Exceptions.** What to do when the steps in the scenario are interrupted
- **Extensions.** Additions to the steps in the scenario, for different conditions than assumed in the main success scenario
- **Technology & Data Variations List.** Variations in steps

The main benefit of the Use Case format is as a list of additional information to add, to augment a User Story formatted requirement.

The main danger in the Use Case format is in making implementation decisions earlier in the development process than they are needed. For instance, it is tempting to specify the steps of interaction between the primary actor and the system in too much detail, before enough is known about the project to allow choreographing those steps for best effect.

Progressive Unfolding: Writing Capabilities

A capability is a formal, high-level description of a quality or ability that has been proposed at higher levels and then approved for development based on the expectation that it will help the organization meet major goals.

Here are examples of the kinds of goals that lead to a capability:

- Increasing ROI (Return on Investment)
- Improving public perception of the organization
- Growing market share
- Complying with government regulations
- Strengthening system infrastructure to allow implementing additional business capabilities in the future

In Lean-Agile, there are two kinds of capabilities:

- **Business Capability.** A description of a quality or ability that the Business requires in order to meet its goals. Usually, Business Capabilities flow from strategic concepts of the organization.
- **Architectural Capability.** A description of large technology initiative that addresses issues such as technology integration, obsolescence, scalability, performance, or taking advantage of new technologies or standards. Architectural Capabilities often cut across a portfolio.

The initial strategic concepts are typically written by someone with strategic oversight, like a Business Owner/Sponsor, or a Chief- or Enterprise-Architect. When a strategic concept is selected by enterprise leadership for development, someone is assigned to become the primary author of the capability.

- For Business Capabilities, the primary author is a Product Manager.
- For Architectural Capabilities, the primary author may be the senior architect (e.g., Chief or Enterprise) who originated the concept.

The author consults with the strategic experts who originated the concept and other stakeholders to write the capability statement in a standard format. The usual capability format consists of:

- A statement of the value and the vision ("why") behind it
- The criteria the stakeholders will use to judge the success of the implementation of the capability's value
- Limitations and constraints placed on the development of the capability (including Non-Functional Requirements)
- (Optionally) A lightweight business case outlining how developing the value will benefit the organization

Approach

Inputs to writing capabilities include:

- High-level concept statement(s) identifying a strategic capability that, if implemented, is expected to further major organizational goals
- Consultation and supporting information from the originators of the strategic enterprise-level concept, other executive-level people, and stakeholders such as customers and their customers (generally coordinated with the first-line customer's agreement), users, and regulators

Steps in writing capabilities are:

1. Obtain all written and verbal information about the strategic-level capability.
2. Identify all stakeholders of the capability and develop channels of communication to allow communicating with them about the capability as it is being written.
3. Develop an understanding of what the customers and other main stakeholders want (about which they already know) and needs (about which they may not yet recognize).
    - Go to the customers' and other major stakeholders' place where they do the work your new capability will serve.
    - Observe how they do their work, especially that which the intended capability will serve, and the kinds of challenges they face in that part of their work.

4. Sharpen the understanding of the potential target audiences for the capabilities by doing the following:
   - Develop "archetypal users," otherwise known as "putting a face on your customers." There should be an archetypal user for each market segment. (A "market segment" is a group of users who have similar needs to each other; one market segment is distinguished from another by differences in the common needs of each group.)
   - Give each archetypal user a profile, including:
     - ◊ A name
     - ◊ Personal details and backstory (similar to those of many people in the market segment)
     - ◊ Specific usage scenarios they face at work: Currently and if they had the capabilities available
     - ◊ The archetype for the most-important market segment is called the *primary* archetype; the others are called *secondary* archetypes.
   - If funding for implementing the capability is reduced, then begin by eliminating the secondary archetypes. Try to achieve the primary archetypes as much as possible.
5. For each archetype, do the following. Try to do these in parallel because each one will tend to inform the others.
   - Write the "value statement" of the capability.
   - Describe the criteria for stakeholders accepting the implementation of the capability. These are Business-oriented acceptance criteria rather than technical- or implementation-oriented acceptance criteria.
   - Bound the scope of the capability implementation by identifying limitations and constraints to be placed upon it.
6. Ensure that the archetype, as you are defining it, is implementable in an optimal way.
   - Coordinate with stakeholders such as
     - ◊ Value Stream Owner
     - ◊ Product Manager
     - ◊ Technology Owner
     - ◊ Technical Leads

  - Refine the archetype to make it as implementable as possible without adding implementation details. Consider:
    - ◊ Implementable with available technologies
    - ◊ Expandability
    - ◊ Modifiability
    - ◊ Optimal dependencies and operability with systems around the capability's implementation system

7. Repeat steps 5 and 6 until the entire capability and its parts satisfy all stakeholders as well as are practical to implement.

Here is an *advanced topic*. It is optional and outside the scope of this practice. Fine-tune the capability by doing the following:

1. Prioritizing the different stakeholders
2. Prioritizing the different aspects of the capability by doing a weighted sum of their importance to each stakeholder, then their importance again across all (weighted) stakeholders
3. Using methods designed for this kind of work, including the Analytic Hierarchy Process (AHP) and Quality Function Deployment (QFD)

Template: Capability

The capability must express the value of the capability, acceptance criteria, and scope. It may also describe non-functional requirements.

Value Statement. Here is a common format for a vision statement.

> **FOR** <target customer/market segment>
> **WHO** <statement of need>,
> **THE** <product name> **is a** <product category>
> **THAT** <product key benefit, compelling reason to buy>**.**
> **UNLIKE**< primary competitive alternative>,
> **OUR PRODUCT** <primary differentiation>

Acceptance Criteria. These are the bases on which the Business Owner and other major stakeholders will judge whether or not the implementation of the capability has satisfied their needs. Satisfaction of these criteria will be formally evaluated and approved or disapproved in the system demo at which the implementation of the capability is exhibited.

Scope (clarifying the limits of development and the constraints to be placed upon it)

- **In scope.** What kinds of capabilities are meant by the stakeholders to have included in the implementation of the capability.
- **Out of Scope.** What kinds of capabilities are not intended by the stakeholders to be implemented in the capability (primarily, capabilities that might otherwise be assumed to be included but should not)

NFRs (Non-Functional Requirements): Characteristics of the final product that are not functional (that is, not doing specific actions), yet are considered essential by the major stakeholders. Examples:

- **The "-ilities."** Reliability, modifiability, maintainability, etc.
- **Limits (such as for performance).** For example: must-support number of simultaneous users, maximum response latency, etc.

Alternative: A Lightweight Business Case

For many years, the way to justify business action has been to write a business case. This has been a heavyweight process and document that took days or even weeks to create; far too time-consuming to use at the level of most capabilities.

Now a lighter-weight approach is becoming the norm for business case development. It is called "Business Model Generation" (BMG). The format for a business case from the Business Model Generation approach is called a "Business Model Canvas" (BMC).

The Business Model Canvas focuses on a few, key issues, and simplifies their exploration. This makes it practical to apply the power of value-based thinking to smaller tasks that are nevertheless significant to the health and success of the enterprise. Capabilities fit into this level.

A lightweight business case for a capability will select the aspects of the Business Model Canvas that are most relevant to the capability specified by the capability.

Look at the following models and choose a subset that fits the needs of the organization. While all of these are good, any particular format may be heavier-weight than what you need or conversely may omit concerns important to you.

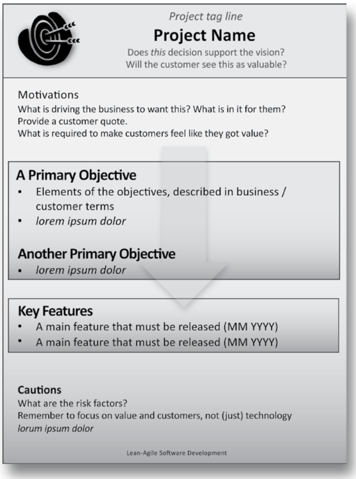
- The BMC model is described in the book *Business Model Generation*.
- A powerful subset of the BMC model useful at the capability level is explained in the book *Value Proposition Design*.
- A subset is found in the SAFe method described at *www.scaledagileframework.com/epics/*.

Alternative: The One Page Summary

Writing a "pro forma press release" is a fun, short exercise for creating the product vision and release vision. The Product Manager brings the essential description of goals and drivers and helps to guide the work.

The whole team engages in the conversation about what should go into a "press statement" to be released at the end of the effort.

The result is a deep understanding of the release by the Team. When finished, the One Page Summary should be converted into a poster and displayed in the team room.



Project tag line

Project Name

Does *this* decision support the vision?
Will the customer see this as valuable?

Motivations
What is driving the business to want this? What is in it for them?
Provide a customer quote.
What is required to make customers feel like they got value?

A Primary Objective

- Elements of the objectives, described in business / customer terms
- *lorem ipsum dolor*

Another Primary Objective

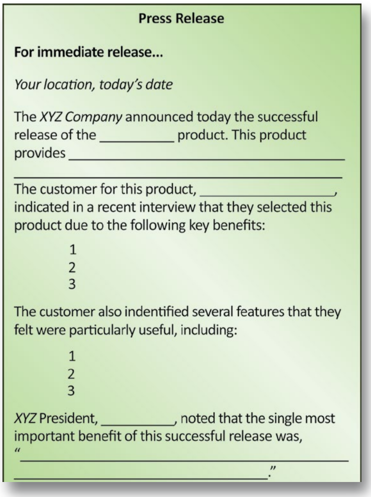
- *lorem ipsum dolor*

Key Features

- A main feature that must be released (MM YYYY)
- A main feature that must be released (MM YYYY)

Cautions
What are the risk factors?
Remember to focus on value and customers, not (just) technology
lorum ipsum dolor

Lean-Agile Software Development



Press Release

For immediate release...

Your location, today's date

The *XYZ Company* announced today the successful release of the __________ product. This product provides ____________________________________

____

The customer for this product, __________________, indicated in a recent interview that they selected this product due to the following key benefits:

1
2
3

The customer also indentified several features that they felt were particularly useful, including:

1
2
3

XYZ President, __________, noted that the single most important benefit of this successful release was,
"____________________________________
________________________________."

Progressive Unfolding: Capability to MBIs

The key to decomposing a capability into an MBI is to understand what an MBI is.

An MBI is a requirement covering the smallest grouping of capability that is worth delivering to a customer. This is in effect the indivisible "atom" of business value. Delivering anything less than this package of capability will not be perceived by the customer as providing them with what they want or need. They may even be irritated by receiving new functionality that they must think about and deal with, but which does not enable them to do anything more that they care about or want to use the system to accomplish.

An example of this is a swimming pool. You can list the features of a swimming pool—filtration, a diving board, slide, lights, robot cleaner, steps, concrete apron surrounding it, pool edge bar —and some non-functional features—shape, depth, length, width, color—and have an impressive list. Do all of them have to be delivered in one big package?

If the homeowner has any limitations to his or her budget, or if they would like to start using the pool sooner than an all-up, bells-and-whistles pool would take...the question becomes, what is the smallest set of features that would satisfy the homeowner as a first pass? Everyone wins...the homeowner doesn't have to save up as much money before starting their project (even if that money is just the down payment on the loan), they get a usable pool in a shorter span of time (an acceleration of value), and they might even be able to pay for the rest of the pool with future upgrades while enjoying the basic pool.

This example shows several important things about MBIs:

- MBIs are all about value...what will the stakeholder recognize as benefiting them? That's what we want to deliver to them; not some pieces they cannot use yet. For instance, shipping a homeowner a pool slide, by itself, before they have the pool structure or concrete apron around it is unlikely to make them happy. This same question can be asked about features: Will the stakeholder appreciate it if they were to receive feature X by itself...or some group of features that do not add up to usable value for them?
- MBIs can often best be identified by jumping down one more level of detail and thinking about potential features that might go into the MBI. This is not a definitive list, but it is a way of thinking about the scope and boundaries of the MBI.

- MBIs allow delivering more value to the customer, earlier; not less (as you might assume from the name “minimal”). Delivering the minimum acceptable amount means less work and less time between the “order” and the “delivery” of useful functionality. It also often allows earlier starts. The customer receives what they want, as soon as possible.
- MBIs are all about boundaries...what is absolutely needed to make the MBI into something the customer will recognize as value? What can we make not go into the MBI, and the customer will still be pleased?

Decomposing a capability into MBIs means thinking deeply about the stakeholders and what they really want and need. Then it becomes relatively easy to decide which MBIs can be identified that, together, will cover the value of the entire capability.

Approach

Inputs to decomposing capabilities to MBIs include:

- A capability that has been approved for further development
- Consultation and supporting information from the Product Manager, architect, and stakeholders

Steps in decomposing capabilities to MBIs are:

1. Choose a capability in the Portfolio Backlog, that has been approved for development based on portfolio backlog prioritization.
2. Obtain all written and verbal information about the capability to be decomposed into an MBI.
3. Develop a deeper understanding of the archetypal users who will be the focus of the MBI you will write. This includes both their wants (things they consciously know about) and needs.
4. Choose the stakeholder(s) to be the recipients of the capability that will be specified by the MBI, and the major aspect of value in the capability to deliver to them.
5. Write the MBI. This is normally done in User Story format.
   - Define the minimum value based on what you know about the stakeholders to whom the MBI will be targeted.

   - If the scope of the MBI is not clear top-down, then do the following:
     - Create a list of features for a thorough implementation of a major value area from the capability.
     - Remove features until reaching a minimal list of features that you believe the stakeholder(s) will perceive as value.
     - Validate your assumptions about the MBI stakeholder(s) by asking them if they would view a package of this list of features as useful to them. Ask them clarifying questions: Is there a subset that would be useful? and If this package is not useful to you by itself, what could we add to it to make it useful?
   - Repeat until the functional scope of the MBI has been defined.
   - Attach to the MBI any important supporting information that helped with defining the MBI.

6. Ensure that the MBI, as you are defining it, is implementable in an optimal way.
   - Communicate with more implementation-oriented roles.
   - Edit the MBI to make it as implementable as possible, without adding implementation details.
   - Capture any bounds, limits, or constraints to be applied on the implementation.
7. Develop Acceptance Criteria for the MBI in conjunction with relevant stakeholders.
8. Repeat steps 5 to 7 until the entire MBI and its parts satisfy all stakeholders as well as are practical to implement.

Template: MBI

The MBI format typically consists of a value statement in the User Story format. See "Overview: Business Discovery and Business Delivery" on page 70.

Discussion

The objection is sometimes raised that, "we don't want to deliver the minimum; we want to deliver the maximum!" This ignores the fact that the time to develop functionality goes up exponentially with the amount being developed. That is, the more you do at once, the longer it takes to do every part of it and therefore the whole. This is due to many things, including more people to communicate with between the parts, less-frequent all-up integration, longer times to find problems and therefore to correct them, and so forth.

The challenge is not how to deliver more capability at the same time; no customer really cares about that. The challenge is how to deliver more capability over time. The key to delivering more is to do less at a time but more often.

Progressive Unfolding: MBI into Features

The key to decomposing an MBI into a Feature is understanding what a Feature is.

A Feature is a significant piece of functionality that is part of an MBI, is cohesive within itself (i.e., everything about it works together to do one thing), but will, if delivered by itself to a customer without the rest of the MBI, not be seen as enabling them to accomplish anything that they care about.

So, for instance, in "Progressive Unfolding: Capability to MBIs" on page 113, we introduced the example of a basic swimming pool as an MBI. The MBI was defined by identifying the minimum set of features that, together, would fulfill the homeowner's desire to have a basic swimming pool.

One of the features in a swimming pool MBI would be the capability to filter the water. During construction of the pool, someone would deliver a filtration system to the owner's back yard to be integrated with the rest of the pool's features.

However, suppose that someone delivered just the filtration system to the owner. The filtration system is a significant piece of cohesive functionality that is necessary towards delivering the overall basic pool MBI. But the owner is not likely to be pleased to have a filtration system by itself. He or she will either have to find a place to store it out of the weather, taking up room needed for other things, or leave it out in the yard, perhaps getting harmed by the weather, or perhaps getting covered with a tarp...in either case certainly being an eyesore.

What is a welcome and significant piece of functionality, in the context of its MBI, becomes a liability if delivered by itself. This is the important difference between a Feature and an MBI.

Approach

Inputs to writing features include:

- MBI to be decomposed, drawn from the value stream backlog
- Consultation and supporting information from the Product Managers, architects, and stakeholders

The steps to decomposing an MBI into features are:

1. Choose the next MBI from the prioritized backlog.
2. Obtain all written and verbal information about the MBI.
3. Review the information about the archetypal users (from the MBI's supporting information) who will be the main focus of the Feature you will write. This includes both their wants (things they consciously know about already) and needs (what they may not consciously recognize yet).
4. Write the Feature. This is normally done in User Story format. Attach to the Feature any additional important supporting information you discovered while defining it.
5. Ensure that the Feature, as you are defining it, is implementable in an optimal way. Communicate with architects, Product Owner, leads. Capture any bounds, limits or constraints to be levied on the implementation (e.g., performance).
6. Develop Acceptance Criteria for the Feature, in conjunction with stakeholders.
7. Repeat steps 4 to 6 until the entire Feature and its parts satisfy all stakeholders as well as are practical to implement.

Template: Features

The essential attributes of a feature are:

- **ID.** An identifier used to refer to the feature. Use an ***intention revealing name.***
- **Business Value.** *(optional / helpful)* Indication of the value to the business. The Business decides on the scheme to use: a priority number, an estimate of ROI, an estimate of cost savings or risk reduction.
- **Description.** A few words to describe the feature. Here is a good template:
  - Who or what area, including the scope and scale (for example: all managers, only active customers)
  - The capability / functionality needed
  - When is it needed (urgency)
  - How we know we have achieved the value
  - The method or approach for validation
  - Acceptance tests and criteria to validate value realization
- **Size.** The size, in "points" of the feature. Must be able to be completed within the release.

Progressive Unfolding: Features into Stories

The key to decomposing a Feature into stories is understanding what a story is. A story is simply a piece of a Feature, small enough to be developed by a single team. If the Feature is very large, it may be helpful to create relatively large stories from it. These will all be implementable by a single team, but may require a complete iteration or even multiple iterations to develop.

If you have developed team stories, you will need to take an additional step and decompose each of these into a yet-smaller size, called "right-sized stories." Right-sized stories can be implemented by one team in roughly three days.

The approach used to decompose a Feature into a Story is essentially the same as that used to decompose an MBI into a Feature. A Feature covers functionality that is too limited to deliver value to the customer or stakeholder. So does a story.

Approach

Inputs to decomposing features into stories include:

- A Feature to be decomposed, drawn from the Value Stream Backlog
- Consultation and supporting information from the Product Manager and team members

The steps for decomposing features into stories are:

1. Choose the next Feature from the prioritized backlog.
2. Obtain all written and verbal information about the Feature. This will come from the Feature description as well as the people who contributed to writing the Feature.
3. Write the Story. This is normally done in User Story format, but consider other formats if needed. See "Progressive Unfolding of Requirements" on page 101.
4. Attach to the Story any additional important supporting information you discovered while defining it.

5. Ensure that the Story, as you are defining it, is implementable in an optimal way.
   - Identify concerns such as implementablity with available technologies, expandability, modifiability, and dependencies and operability with systems.
   - Capture any bounds, limits or constraints to be levied on the implementation (e.g., performance).
6. Develop Acceptance Criteria for the Story, consulting the Product Manager, Product Owner, and stakeholders as needed.
7. Continue with steps 3 to 5 until the entire Story and its parts satisfy the Product Owner and it is practical to implement.

Template: Stories

The essential attributes of a story are:

- **ID.** An identifier used to refer to the story.
- **Description.** The intent of the story. It should be descriptive enough to allow a business analyst to write it several iterations after it was initially conceived. Here is a template:

  As a <user>, I want <capability> so that I get <business value or functionality>

- **Size.** Stories must be "right-sized" so that a teamlet can complete it within 1/3 to 1/2 of the iteration, ideally in one to several days.
- **Value.** Can be expressed in "Business Value."
- **Validation Strategy.** Every story must have criteria so that the team can know, "Is this story done?"
- **Feature and Related Story.** Where did the story come from and what else is it related to?
- **Story Captain.** *(optional / helpful)* Add this to the card to indicate the team member responsible for shepherding the story to completion.

Types of Stories

Type	Description
Business Capability Story	A capability of the Business that must be developed for the feature to be developed. This would involve such aspects as processes, documentation, training, or equipment.
Analysis Story	Research that needs to be done to describe the feature more fully, including studies, interviews, root cause analysis
User Story	How users interact with the feature, including requirements, test cases, high-level system design
Deployment Story	How the feature is introduced, supported, and marketed. Also known as a "Change Management" story. Includes the run book, configuration settings, environment specifications, application disaster recovery
Environment Story	Physical, logistical, or virtual configurations that must be established before the feature can be developed or deployed.

How well are the requirements understood?

The Lean-Agile approach to story decomposition progresses based on how well they are understood. It is helpful to assess a story's maturity level against a set of maturity criteria called INVEST: Independent, Negotiable, Valuable, Estimable, Small, Testable. Here are criteria for assessing maturity.

Maturity Level	INVEST Factors
Red Level	Independent • Story is defined at a high level. • Story may be dependent on other stories. Negotiable • Features may not be clearly defined. More clarification is needed prior to creating tasks. • Acceptance criteria may not be identified. • Prototypes/wireframes have not been started. • Copy has not been provided. Valuable • Story has not been approved by the Product Owner (or stakeholders). Estimable • Story Point estimation has not been done. • Impact to other IT teams is not understood. • All infrastructure/environment setup is not in place for the work to begin. • Developer or tester does not understand underlying code framework at high-level. • Team is not ready to commit. Small • Story may not complete in an iteration. Testable • User Story is not testable.

Maturity Level	INVEST Factors
Yellow Level	Independent • Story is granular. • Story is independent of other stories. Negotiable • Features are clearly defined. More clarification is needed prior to creating tasks. • Acceptance criteria may not be identified. • Prototypes/wire frames are in progress. • Copy has been provided but not signed-off. Valuable • Story has not been approved by the Product Owner (or stakeholders). Estimable • Story Point estimation has not been done. • Impact to other IT teams is understood. • All infrastructure/environment setup is not in place for the work to begin. • Developer/Tester understands underlying code framework at high-level. • Team is not ready to commit. Small • Story may not complete in an iteration. Testable • User Story is not testable.

Maturity Level	INVEST Factors
Green Level	Independent • Story is granular. • Story is independent of other stories. Negotiable • Features are clearly defined. Easy to decompose story into tasks. • Acceptance criteria has been clearly identified. • Prototypes/wire frames are complete. • Copy has been provided and sign-off. Valuable • Story has been approved by the Product Owner (stakeholders). Estimable • Story Point estimation has been done. • Impacts to other IT teams is understood. • All infrastructure/environment setup is in place for the work to begin. • Developer/tester understands underlying code framework at high-level. • Team is ready to commit. Small • Stories can be completed in an iteration. Testable • User Story is testable.

Example of Story Progression

Consider this MBI: As an XYZ Bank credit card customer, I want to log in and see my card account on my phone.

- **Red Level Maturity.** As an XYZ Bank credit card customer,
  - I want to log in and see my card account on my phone.
- **Yellow Level Maturity.** As an XYZ Bank credit card customer,
  - I want to view my card balance on my phone.
  - I can see my card balance using my smart phone.
- **Green Level Maturity.** As an XYZ Bank credit card customer,
  - I want to be able to see my credit card balance updated real-time via my Android smartphone.
  - I can see my credit card balance.
  - My credit card balance is updated real-time within one minute of a transaction.

Progressive Unfolding: Stories to Tasks

A task work item communicates the need to do some work. Each Team member creates tasks unique to the role: developer tasks, testing tasks, impediment resolution tasks, etc. A task can also be used to suggest that exploratory testing be performed. A task can be used generically to assign work within the project.

Identifying Tasks from Stories

To identify tasks from stories, ask what it will take to do the following:

- Define the functionality of the story.
- Build the functionality.
- Release the functionality.
- Support the functionality.
- Use the functionality.

Template: Tasks

The essential attributes of a task are:

- **Description.** A concise overview of the task to be completed. The title should be descriptive enough to allow the team to understand what area of the product is affected and how it is affected.
- **Estimate.** Tasks must be "right-sized" so that a teamlet or individual team member can complete it in one to several days.
- **Exit Criteria.** Every task must have an exit criteria so that the developer knows when the task is done.
- **Story.** Where did the task come from?

Types of Tasks

Type	Description
Design Task	Work that applies design patterns and coding principles and practices to specify code requirements.
Dev Task	Work that generates code from analysis tasks.
Testing Task	Work that defines validation and verification tests for stories and code.

Estimation and Velocity

Estimation is a Conversation

A primary benefit of estimation is the conversation that happens between appropriate stakeholders about the work to be done. Understanding is more important than accuracy: It validates that everyone understands what is required. Accuracy will come with experience.

Teams face a number of routine challenges to estimation including going into too much detail, making assumptions, doing design while estimating, and being reluctant to commit to an estimate. The Team Agility Master must help the team get through these challenges.

Estimation follows the Plan-Do-Check-Act process from lean: Develop the estimate, do the work, check to see what was done and how much effort it took, and incorporate what was learned for the next iteration.

Estimation is done at every level of planning and involves everyone involved in that planning session. Estimates reflect the team's best guess at the moment. Early on, estimates may be +/- 100%. As learning grows, estimates become more accurate.

Estimating by Points

Agile Teams usually estimate the amount of work required to complete a particular feature or story in terms of story points (or points). A point is not a unit of time. It reflects an arbitrary judgment by the team about how "big" an item of work is. Team capacity describes how many points they can process. The point scheme used follows a sequence such as: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100, 200, 400, 800+.

Other measures can be used for estimation and either converted into points or completely replacing them throughout the life-cycle.

Estimating Work to be Done

Early-on in analysis, estimates are done for features. These are "coarse" estimates. As teams drill-down to stories and tasks, estimates get more accurate: It is easier to estimate smaller things.

Estimating Team Capacity

Team capacity or "velocity" is the number of points that a team can process within the planning horizon (release, iteration). Estimating team capacity involves looking at historical trends: *this* team can handle *NN* points.

Issues and Considerations

Topic	Discussion
What if those doing the estimation only know some of the story?	It is not uncommon to have a team estimate a story where no one member knows the size of all aspects of the story. In other words, some teams may have members that understand how to write the code, others how to test it. In these situations, the team must figure out the best way to do estimation. A good approach is for those who are going to write the code to estimate that aspect of it and those who are going to test the code to estimate that aspect of it and then add the story points of each sub-estimate together to get a total story point for the story. For example, developers may say it will take 8 story points to develop a story and the testers say it will take another 5 to test it. The story estimate is 13 story points.
How do we know we are done with estimation?	Teams will quickly discover that estimates without getting this question answered will usually be lower than what they would have estimated with this question answered. This is a very important question to answer: not just to better understand the story but also to help create better estimates.
What if the story will not be done soon?	Estimates are not really needed for stories that won't be worked on for a while.

Team Estimation Game

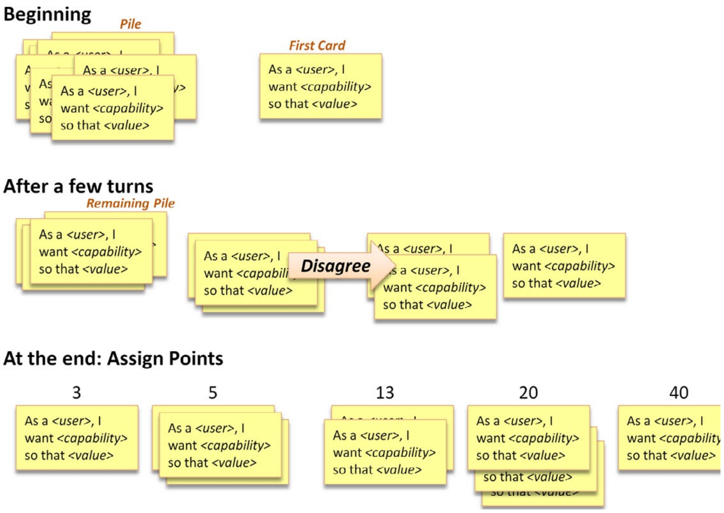
The Team Estimation Game (created by Steve Bockman) helps teams size features and stories based on *relative* complexity. It works because people find it easier to compare the complexity of one feature or story with another even if they do not yet know all aspects of that story. This game is fast, easy, and fun. It helps people not to get bogged down into too many details, which is always a risk during estimation exercises. Relative ranking gives the team the information they need to decide what to work on and how much to commit to in current the planning horizon.

Remember, estimates represent our best guesses about the effort required based on what we currently know. As we gain more experience, we will have more information and can then refine the estimates.

One good way to think of feature or story complexity is the degree of connectedness of the feature or story. This can indicate how interconnected it is with itself or the number of connections to other features or stories.

Traditionally, this game is played at the iteration planning session to size stories for the next iteration.

Another popular estimating technique is "Planning Poker." The Team Estimation Game is typically faster and easier to learn than Planning Poker and results in estimates of similar quality.


Beginning
Pile
As a <user>, I want <capability> so that <value>
First Card
As a <user>, I want <capability> so that <value>
After a few turns
Remaining Pile
As a <user>, I want <capability> so that <value>
Disagree
At the end: Assign Points
3
5
13
20
40

How to conduct the Team Estimation Game

Step	Do this...
Set up	Place story cards in a pile on the table. Select the top card in the pile and place it on the playing surface a foot or so away from the pile.
First play	A player takes the top card off the pile and places it somewhere on the playing surface indicating its size relative to the first card: to the left if it is easier, underneath it is the same size, to the right if it is more complex.
Relative estimation of the rest of the deck	Each person plays in turn doing one of the following: 1. Play the top card from the pile as described above 2. Move a card already on the playing surface, declaring disagreement about its relative size 3. Pass Play ends when there are no more cards in the pile and there are no more adjustments to be made. Note: During play, anyone may talk about why cards are being moved or about what they think about size of the stories. The goal is to get clarification and not to get too hung up on the exact sizes of the estimates. Remember that these are just estimates!
Assign points	The team works together to assign points to each stack to indicate the size (level of effort) of stories that are in that stack. Use the sequence: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100, 200, 400, and 800. When done, write the assigned points on each card.

Part V: Quality

Quality of product and process is the responsibility of *everyone* on the team. It is the result of discipline, communication, and the use of good engineering practices.

Continuous Improvement Before, During, and After

During each iteration, during every activity, the people doing work should help each other think about quality issues. Be explicit about what is required without trying to solve the problem (which is the responsibility of people who will receive stories or tasks to do). The Tec often gets involved in helping to think about issues of technical quality.

Improving quality involves discipline and mutual commitment. Lean techniques – such as retrospections, keeping a clean environment, testing, and a correct use of patterns and code quality – help, but only when the team uses them regularly.

One of the principles of Lean-Agile is to *create knowledge*. Team Agility Masters should be sharing lessons (sanitized if need be) through their community of practice. In fact, this should be part of the assessment of their performance.

The most common point for continuous improvement is the Iteration Retrospection. See "Iteration Retrospective" on page 95.

Tip: STOP! If you do not pause to reflect, you will not learn. And if you do not learn, you cannot improve. And you will cease to serve your customer.

Tip: STAY FRESH! At some point in every team's life, quality improvement becomes rote, stale. It ceases to be useful to the team. Whenever you sense this, it is your responsibility to point it out. The team must consider together how to shake things up, to see how they can make it relevant again.

Impediments to Progress and Quality

An impediment is any technical, personal, or organizational issue that inhibits progress being made on delivering product to the customer. Impediments are inevitable. It is always the responsibility of the team to expose impediments as they arise rather than burying them, hoping they will go away. The team is responsible for resolving impediments that lie within its "span of influence" (within which it can effect change). The Team Agility Master, acting on behalf of the team, elevates all other impediments to management to be addressed. The and other management agree to help.

Addressing Impediments

The team identifies impediments in the process of actually doing work. They should surface at least during the Daily Stand-up and the Iteration Retrospection. Whenever an impediment is identified, the Team Agility Master writes a story for the impediment. The team prioritizes the impediment story and decides whether to resolve as a team or elevate.

In the crush of work, teams are tempted to ignore impediments, or they identify too many impediments and never re-solve anything. Both of these are almost always mistakes. The Team Agility Master must encourage the team to stay disciplined.

Techniques for addressing impediments include:

- An After Action Review, targeting the impediment
- A quick improvement workshop (Kaizen) to address the impediment
- A Root Cause Analysis effort

Issues and Considerations

Topic	Discussion
Motivation	In the crush of work, teams are tempted to ignore impediments, or they identify too many impediments and never resolve anything. Both of these are almost always mistakes. The Team Agility Masters must encourage the team to stay disciplined.

Examples of Impediments

At this level...	Common impediments include...
Individual team member	Related to preparation • Lack of training • Development environment is not available: hardware, software, tools • Team infrastructure is not available: project board, tools Related to communication and customer • Not allowing team members to talk with customers at all • Excessive interruptions • Having to multi-task • Being repeatedly interrupted by too many support issues • Analysts working on too many projects at once, being asked questions from too many different developers Related to project • Business Analysts who do not know the customer, what they need, or what is most important • Not knowing who represents the voice of the customer • Team members making promises inappropriately

At this level...	Common impediments include...
Individual team	Related to the project • Lack of clear project vision • Too much Work-in-Progress • Unclear definitions of Done • Not checking in code frequently Related to time • People do not come to the Daily Stand-up on time • People have different work hours, time zones, shifts Related to the ecosystem • Not being colocated • Not having an open Team Room • Excessive interruptions • Abrupt change of plans through management directives • Management adding work to the backlog in the middle of an iteration
Cross-team	Lack of Value Stream Maps Lack of timely feedback from other teams Lack of appropriate integration and test environments Related to structure • Team structure that makes end-to-end functionality difficult to attain • Having to reform teams each iteration / release, causing re-learning, trust-building, and re-establishing team dynamics • Interfacing with non-Agile teams

At this level...	Common impediments include...
Management	Product backlog not being updated Team Agility Master not being supported by management or team Deficiencies in the code base • Having to deal with old, legacy code that no one knows • Required language skills have atrophied over time • Poor quality code • Lack of documentation (architectural, design) • Having to attend meetings not related to development of the project
Testing and QA	Developers not wanting to talk with a tester, or a customer Developers not talking to testers until late in the process Lack of a testing environment Taking too long to run tests Lack of access to data due to confidentiality or regulations Difficulty in performing manual tests Inability to do automated testing Lack of tools for testing databases programmatically Writing tests only after code has been completed

> ***"Go see for yourself." The Lean philosophy is based on management by fact, and the belief that facts exist where they are created, not far away from it. All improvement, whether it is technical innovation, process method redesign, or policy, must be based on the actual needs of the situation observed for oneself.***

Communicating Information

Do everything possible to ensure clear communication: between team members who are doing work; between customers, stakeholders, and the whole team; between teams within the organization. This lies at the heart of much of Lean-Agile practices.

The Lean-Agile Team Room

Whenever possible, the development team should be colocated. Rich communication is crucial for efficiency and speed and not making mistakes. Ideally team members are located within 100 feet of each other.

Colocation is not always possible. For ideas and considerations, see "About facilitating teams that are not colocated" on page 51.

Lean-Agile teams should have a room dedicated to the team. All relevant information, especially the Project Board, is posted and maintained in this room. The Daily Stand-up takes place in this room. Teams use the room for meetings and teamlets may also use it for swarming work. Here are important considerations:

- A sufficient number of developer cubes plus some additional work stations
- Direct, open visibility between all team members
- Gathering points (e.g. small round tables) in the middle of the room allow team members to talk together
- Walls covered with white boards
- A dedicated board for the Information Radiator

Information Radiators and the Team Project Board

The information radiator is a type of visual control that provides visibility about the project to everyone who wants to know about it. Information radiators are always up to date. Whenever management and stakeholders need to know team performance and progress on product development, they visit the workplace and review the team's project board. Visitors should be able to identify projects and who to talk to for more information. At a glance, the team should be able to know what is going on and who is doing what and decide what needs to be done next. This requires making information visible in one, predictable place.

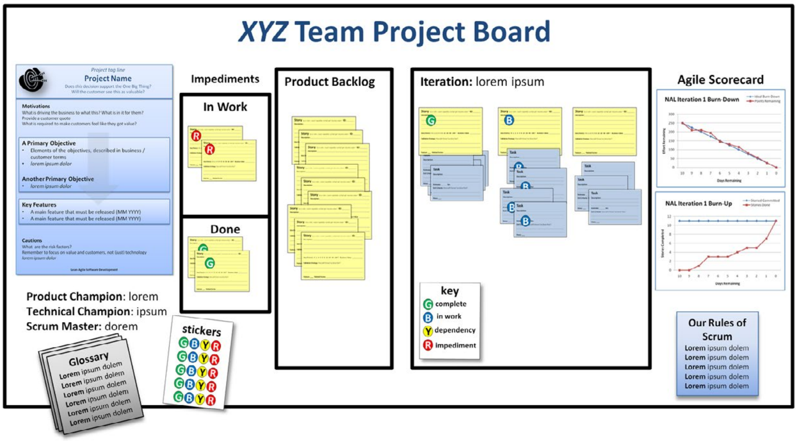
While everyone on the team is responsible for keeping the project board up to date, the Team Agility Master does most of this work.

For teams that are not colocated, they must have one, common project portal for all of the information. Often, teams will maintain two sets of boards: a physical board in the team room(s) and the project portal.

Essential elements of the team's project board include:

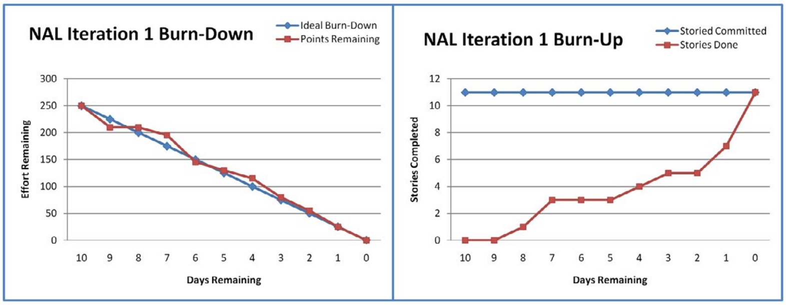
- The product vision / release vision poster
- Contact information, especially the er, and Team Agility Master
- Glossary of terms
- The team's rules
- The Agile scorecard (burn-down and burn-up charts)
- The list of impediments
- The product backlog stories
- The iteration stories and tasks

Here is an example layout of a physical team project board.



The Agile Scorecard shows the story point burn-down and story completion burn-up charts for the iteration. At the end of the iteration, the lines should join.

Here is an example of an Agile Scorecard showing burn-down and burn-up.



Quality: Documentation and Standards

Aim for perfection! Improve constantly! Test Early and Test Often.

Highly performing organizations constantly think about what can go wrong and improve processes to address this. They welcome mistakes as opportunities to learn. They try to stay sensitive to conditions of the team. Quality involves making sure the team builds the right product and builds the product right.

- **Build the Right Product.** The close relationship between the Business and development and short iterations makes it easier to ensure that requirements are well understood. Customers have the opportunity to provide feedback early and often, when it is easiest to change.
- **Build the Product Right.** In addition to discovering defects, testing should aim to prevent defects in the first place.

In Leanban, quality is linked from features to stories to tasks. It is the responsibility of everyone on the team: , Business Analyst, developer, and tester.

In software, quality involves having a language in which to discuss good practices and standards to assess how well code measures up. Testing is fundamental to the pursuit of high quality and is the responsibility of everyone on the team. It must be done as early in the process as possible and continuously.

The primary deliverable is a software product that the Business perceives as offering them value. Stay focused on this deliverable. In the course of this work, other artifacts are created as described in the table. Use these but don't get distracted.

This document	Describes...
Acceptance tests, unit tests	What the features are supposed to do and the criteria for knowing when a work item is done
Team project board	The current status through various visual controls
Feature descriptions	The business value which the feature's functionality will achieve. This includes the Acceptance Tests that determine if that feature has been achieved.
Product vision	What the Business requires of the product, the charter for the project.

The Lean-Agile Approach to Standards

The team must stay "mindful" of what the enterprise has learned: enterprise standards, best practices. As much as possible, begin with organizational coding standards. If it doesn’t work, identify the standards as an impediment which then needs to be resolved by the team.

The Application Development Manager helps teams with understanding, using, modifying, and communicating changes to standards. The latter is important because discoveries by one team could well help other teams.

Documentation should always have a purpose. Never document simply for the sake of documenting. That is *non-value-added work* and that is waste.

Patterns and Code Quality

Patterns (also known as "design patterns") focus on using existing quality solutions to solve recurring problems. As developers use patterns, both the team and the organization begin to experience improved quality:

- They incorporate better solutions than would otherwise have been thought of, because they are working with the cumulative experience of many developers.
- They gain a common vocabulary, which improves the quality of their conversation.
- They gain a deeper and more flexible understanding of object-oriented analysis and design.
- They have architectures that allow for more maintainable code. It is easier to change over time, is appropriately complex, and exhibits good code quality attributes.

To enjoy these benefits, testing should begin as early in the development process as possible.

Attributes of Code Quality

In both object-oriented and procedural code, assessing the quality of software can be difficult. Oftentimes, it involves a conversation among the development team as they think together about the code.

Quality	Description
Testable	Every story and feature must have acceptance tests. Individual code units have unit tests, as needed. Test coverage is adequate.
Non-redundant	Objects/modules handle well-defined responsibilities. Follows once-and-only-once rule.
Encapsulated	Adequate encapsulation, hiding responsibilities at the data, implementation, type, design, and construction levels.
Cohesive	Strong class/module cohesion means that a class/module has a single responsibility and everything is about fulfilling it. Strong method/function cohesion means that each method/function is about fulfilling only one functional aspect of that responsibility. An object method implements a single behavior. Functions are related in a routine (or class). Strong cohesion relates to clarity and understanding.
Proper coupling	Classes/modules have strong internal relationships but small, direct, visible, and flexible relations to other classes/modules. Code is loosely interconnected.
Assertive	It is clear what each class/module is about and follows coding and naming standards as defined by the team. Objects tell each other what to perform.

The testing plan addresses all aspects of testing: Acceptance Testing, Automation, and Continuous Integration

Quality: Testing

Testing involves more than simply discovering bugs. Testing helps discover the causes of errors and eliminating them. Testing helps make explicit the assumptions and requirements that customers have without getting overly technical. Testing ensures code integrity and compatibility with other code modules.

Testing minimizes the risks caused by humans, machines, and environment.

Planning the testing strategy should also begin as early as possible. This requires careful consideration: what will be tested, how much can be automated and what must still be done manually, what the testing environment(s) will include, who will do testing, what they need to know, and what standards they will follow.

A system is accepted when it is "Four Done":

Done: The system runs on the developer's computer as expected

Done: The system is verified by running unit tests, code review, etc.

Done: The system is validated as being of deliverable quality with functional tests

Done: The system is ready for (pre-) production

Acceptance Testing

Acceptance testing helps the team answer the question, "Does the code do the right thing?" Acceptance testing is required by Lean-Agile.

Note: The issues that must be considered are described in *www.netobjectives.com/courses/lean-agile-testing*. For more information, see “The Role of Quality Assurance in Lean-Agile Software Development” in *Lean Software Development: Achieving Enterprise Agility* (Shalloway, Beaver, & Trott, 2009).

Acceptance tests come from the point of view of the user of the system. Factors to consider include:

- Up-front acceptance test specifications help the understanding of the requirement.
- Acceptance testing can be used to integrate development and test, increasing the efficiency of development.
- Acceptance testing can be used to improve the process being used to develop the system by looking at how to avoid errors from occurring in the first place.
- Automating acceptance tests can lower regression costs.

Acceptance tests are as much about the understanding and definition of the requirement as they are about validating that the system properly implements it. Ask the question, "How will I know I've done that?" This is something that opens the door for getting an answer in the form of an example - which can (should) be a test specification. This process of having an abstract statement (the requirement) and a corresponding concrete example (the test specification) improves the conversation between developers and analysts and testers and customers.

Every feature and every story must have an acceptance test. These tests can involve both manual and automated methods. The outcome is to deliver fairly well-perfected code where defects have no serious consequences.

Automated Testing

Automated Testing is the use of software and hardware to verify and validate the behavior and qualities of the software under test. It involves software to control the execution of tests, to compare actual outcomes with predicted outcomes, to set up test preconditions, and to control reporting functions. Usually, test automation involves automating manual testing processes that are already being used.

Planning for test automation happens at the earliest stages and should include all levels of testing: unit, integration, system, system integration, and functional/acceptance.

As a general rule,

> All tests, including related setup, configuration, and evaluation steps, which can be automated, should be automated with priority given to aspects which are the most time consuming, error-prone, and tedious.

Common automated test frameworks include xUnit-based approaches (CPPUnit, JUnit, PyUnit), Boost test, and FIT.

When selecting an automated testing framework, consider the following:

- Adequate range of available interfaces: GUI-driven, an interface friendly to automation via scripts and command-lines
- Hardware-software dependencies and system requirements
- Procurement difficulty and cost
- Community support
- Feature mismatch between automatable and manual interfaces
- Performance
- Available API library
- Test data post-processing
- Fit in the overall Agile life-cycle tool suite
- Report generation

Step	Discussion
Check out the software from the repository	Check out from the repository must be handled by scripting or other means in order to avoid error.
Build the software into the deployable or executable form	Compile (and link) the code in order to detect errors in grammar, etc. The automated tool should report errors to compile to the user in an effective manner, such as a scoreboard. Examine errors in the full context which was present when the errors were emitted from the compiler or linker. In the case of an automated build server, use a full-fledged, fully automated build system infrastructure.
Set up of test preconditions	Establish the dependencies required by the system or unit under test. This can involve many activities: • Simple initialization of a variable or instantiation of a class under test • Setup routine or table of a testing framework such as CPPUnit or FIT • Initialization of target nodes • Setting up environment variables • Connections to other systems, such as databases The degree to which developers can properly handle dependencies is the degree to which they can easily and properly test their software in a self-contained and automated fashion. Distributed, real-time, and embedded (DRE) and network systems introduce unique complexities for establishing full testing preconditions, such as setting up COTS network test equipment via third party automation APIs as well as command line scripts, accessing and modifying a database of network topologies, and automated configuration of COTS network connectivity devices.
Deploy the software	This step may be straightforward when deploying on a single workstation or quite complex when deploying on DRE and network systems involving multiple levels of security.

Step	Discussion
Run the tests	If there are only a few tests, a script can automate running the tests. With a larger number of tests, use a testing framework to register and run the suite of tests. Use "Test Runners" or "Test Conductors" to orchestrate the control and actions of many pieces of test equipment, test software, and test result display to fully execute a set of complete functional tests.
Control software and resources that the Site Unit Test requires	Whether testing a module in isolation testing the module with the actual collaborators, the test strategy needs to control the other software entities: their lifetime (creation, initialization, configuration, destruction) and their behavior during the tests. A simple technique is to use a mock object that is part of a runtime polymorphic hierarchy and is hidden behind an interface. This approach allows for the substitution of the mock or the real entity depending on which kind of test is being run. More complex domains will require more sophisticated approaches; for example, controlling the network traffic entering nodes of a large, heterogeneous, mobile *ad hoc* network.
Retrieve and display the results	The testing framework should enable viewing test results, ideally in a simple scoreboard display or other standardized report. As far as possible, they should help developers find which tests failed, which errors occurred, and where.
Notify team members	Rather than requiring the team to monitor the test, the testing framework should actively notify the team about results so that they can isolate and fix problems quickly and effectively.
Information statistical analysis	The test strategy should be informed by statistical correlations of test results and other metrics that show how testing is progressing over time. For example, code coverage over the course of a week or growth rate of the number of test cases.

Continuous Integration (CI)

Continuous integration is the automation of the build and test process, all triggered automatically when code is checked into the Source Control Management (SCM) repository. It offers the team rapid feedback on the integrity and quality of the code.

The basic aspects of CI include:

- Automated, scripted builds and testing
- Using a dedicated CI server / build farm
- Triggering the CI server on check-in
- Building as frequently as possible
- Build the entire system
- Run automated test suites
- Notify team when build breaks or a test fails

Advanced aspects include:

- Static analysis
- Dynamic analysis
- Performance / Load testing
- Database testing
- Automated document generation
- Build deployment packages
- Deploying selectively to live test environments

In order to configure the SCM system to trigger the CI server on a check-in, developers usually require administrative rights to the SCM server. To support this, the SCM server provides "hooks" that allow operations to occur before and after a successful check-in.

For example, if a project is using Subversion, developers configure the post-commit hook to trigger the CI server. The CI server receives information on the SCM repository such as branch, revision, changed files, and comments the developer made regarding the check-in. Based on this information, the build master delegates jobs to a series of predefined build slaves.

Automated, scripted builds and testing

Many systems involve modules that come from a variety of sources: subcontractors, open source, third party closed source libraries, and in-house code. Each of these individual software modules have their own set of build procedures; their own set of dependencies on other software modules. Keeping track of these dependencies is difficult. Not having the ability to track these inter-package dependencies explicitly is a recipe for disaster.

Handling inter-package dependencies should be implemented in an automated way rather than relying on a series of manual steps. Such an approach should support the following:

- Building from clean source
- Ability to clean directories safely, in order to restore the directory to a pristine state, removing executables, etc.
- Safe and fast incremental builds
- Options for compiling with different compilers and operating systems to the extent possible

Using a dedicated CI server / build farm

The build farm is comprised of one or more computers ("build slaves") dedicated to carrying out any instructions that the continuous integration server (the "build master") desires. These instructions involve a set of executable procedures to update from source control, build and test the software, and finally retrieve and return results. Results are then radiated out to team members, for example using an HTTP accessible scoreboard, e-mail, RSS, or SMS text message.

Using multiple build slaves reduces the time it takes to detect defects. For example, if a project has five components and six build slaves are available, dedicate five build slaves to unit testing a single component and dedicate one build slave to running integration and system level tests. This gives fast feedback to developers who need to know about failures in unit tests while running the (lengthy, and potentially CPU intensive) integration tests on another build slave.

Triggering CI server on check-in

The continuous integration infrastructure should be transparent to developers: Developers check code into the Source Control Management (SCM) repository and then the CI system automatically commands the build farm to carry out the appropriate jobs. If any of the tests break, the CI reports which check-in caused the problem. The team can find and fix the problem quickly or at least back out changes from source control.

Build as frequently as possible

In Lean-Agile, build and test is done frequently, ideally on every check-in to detect any errors or any degradation in performance. This minimizes the number of changes to be built and tested at any one time and this reduces the risk of introducing errors.

Build the entire system

Developers spend less time doing rote work (e.g. setting up complex test equipment, configuring XML files for test scenarios) and more time solving complex domain-specific problems.

Is it best to build the entire system from scratch or to automate an incremental build? The trade-off is between speed of build (in order to get rapid feedback) and avoiding introduction of errors and correctness of the build. If the difference in time between a clean build and incremental build is large enough, automate both an incremental build and a clean build: use the former for rapid feedback and the latter to ensure correctness. Doing this task can easily be implemented with multiple build slaves.

> ***Tip. If tests take a long time to run, say over 16 hours, it is better to dedicate hardware to running these tests continuously than run them less frequently. When teams do not have tests running continuously, developers will (inadvertently) break these tests.***

Run automated test suites

Generally, the more tests that are automated, the less time the development team has to spend running manual tests and the more time they can spend implementing new features, automating tests, and solving domain-specific problems.

Running the software and retrieving results may not always be a simple task, especially if the team is targeting an embedded platform. These embedded devices often lack programmable facilities for deploying software and retrieving results. For many projects and teams in the embedded software arena, this prevents them from wanting to use Continuous Integration. The investment in creating the CI server will pay off in the long run.

If the targeted platform cannot be automated, try breaking the dependencies on this target and run the software "off target." This requires an explicit, formal commitment to go back to the objective target eventually.

Getting rapid results back to developers is absolutely priceless and dramatically increases the velocity of the development team.

Notify team(s) when build breaks or a test fails

After running the automated test suite, the results are radiated out to the team, posted to the CI server's scoreboard. The CI server software provides facilities for sending this information out, for example via e-mail, RSS, or even SMS text message. Anyone who has both the authorization and desire to see the information should be able to have access to it.

If a test happens to break, the person who checked-in the failing piece of code is responsible to fix the problem. If that code conflicted with someone else's code, both parties must resolve the conflict.

Beyond Basic CI

Continuous Integration can involve more than merely integrating changes and automating tests. These extensions include new types of tests, performing post-processing on the build and test artifacts, and anything that lends itself to being run in an automated fashion. A few areas for additional efforts appear below in no particular order.

Advanced Topic	Description
Static analysis	Analysis which is performed on the source code of a program in contrast to testing which is performed by actually executing the program. Examples: Checking for uninitialized variables, checking the code against coding style standards, checking for unreachable code, type safety, and checking for potential memory leaks.
Dynamic analysis	Analysis performed during the execution of a program. Examples: Memory usage analysis, code coverage analysis, performance profiling. Examples of tools include gprof (performance profiling), gcov/lcov (code coverage), and valgrind (memory analysis)
Performance/ Load testing	For many software applications, a certain level of system performance is an important non-functional requirement. The customer's requirements may even dictate the use of acceptance testing to verify a certain threshold of system performance has been met. When it comes to CI, if this type of testing involves third-party equipment, special consideration should be given at the earliest stages of test planning to ensure that the third-party solutions can, indeed, be automated. Note: This might not be possible at the individual team level.

Advanced Topic	Description
Database testing	Many systems heavily depend on a database. They need to minimize the impact of change and ensure integrity. In such cases, when it comes to CI, consider the following: • The team's database administrator should be part of a tightly knit cross-functional team so that everyone involved with the database schema is in close contact. • Use a per-developer, local version of the database schema and sample data in addition to a "shared master" version which committed changes are integrated against. • Use an evolutionary database design approach.
Automated documentation generation	The most common form of automatic documentation generation involves documenting the program's API. This works by parsing special symbols in some of the comments of a program's source code.
Build deployment packages	Automatic generation of deployment packages ensures confidence in "release at any time." This process varies from team to team and could include packaging the software binaries resulting from the CI build stage together with the system's documentation. Depending on requirements for deliverables, this may or may not include the program source code. For other projects, this may include creating an executable package to facilitate the installation process.

Advanced Topic	Description
Deploy to live test environment	Recreate the production environment as much as possible. If a live test environment is not readily available, this is difficult. Example. The need for remote management of the power and network connectivity of the system under test could create a requirement for a remotely accessible power distribution unit and remote control of the network. Resource management is another aspect. Examples include: hardware targets, network connections, test equipment, and servers. Represent these resources in a database which tracks their status in terms of configuration state and availability for use in a test. A server uses the information in this database to allocate resources to tests, as well as report status on tests in progress and/or the state of individual devices. Automated asset allocation is important when a team needs concurrent testing of different system-level configurations which use many of the same types of underlying resources. Example. A network of embedded targets can comprise the system under test. Testing this system involves multiple different tests running simultaneously. This requires automatic resource allocation. Identify as many of the requirements for full test-environment automation as possible and cast the entities involved as resources with an eye for how they could be managed in an automated way.

Here are references for more information:

- Static Analysis: *en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis*
- Dynamic Analysis: *en.wikipedia.org/wiki/Dynamic_code_analysis*
- Database Testing:
  The Database Agility course describes a test-driven approach to evolutionary database design; see *www.netobjectives.com/courses/test-driven-development-database-boot-camp* and *martinfowler.com/articles/evodb.html*
- Automated Document Generation: *en.wikipedia.org/wiki/Comparison_of_documentation_generators*
- Build Deployment Packages:
  en.wikipedia.org/wiki/Installer#Installer and *en.wikipedia.org/wiki/Package_management_system#Common_package_management_systems_and_formats*

Unit Testing Frameworks and their Relation to Automation

People often think that xUnit is for unit testing and FIT (or an equivalent) is for acceptance testing. In fact, developers can use both tools equally for these purposes.

The xUnit architecture relieves the developer from many of the repetitive tasks associated with unit testing: setting up preconditions, registering tests, executing tests, tearing down tests, defining test suites, presenting test results, reporting state changes, and making assertions.

FIT allows for acceptance tests to be specified by a non-programmer and even run by a non-programmer once the fixtures are in place. FIT provides powerful facilities for creating, setting up, tearing down, running, reporting, and displaying tests. It also provides facilities for full automation via command line scripting.

JUnit: A Cook's Tour (Sourceforge 2009) and xUnit Test Patterns (Meszaros 2007) contain excellent discussions of the patterns and pattern language that underlie the xUnit family of frameworks.

Part VI: Checklists

Checklists are useful reminders for teams to ensure they are considering all of the practices of Leanban.

Starting a Lean-Agile Team Checklist

✓	Activity	Description
	Select the right people	Select the . *This is a crucial decision.* Select the Team Agility Master. Select candidates for the team.
	Select the right project	Select candidate projects that have a good likelihood for visibility and success.
	Train the people	Allow enough time to train the team with the skills they require. Allow enough time to give developers specialized training: • Acceptance Test-Driven Development • Test-Driven Development • Object-oriented concepts and design patterns • Automated acceptance testing
	Build the physical environment	Establish and/or build out the physical environment for the team: • The Lean-Agile team room • Visual controls, team's project board Decide how to handle teams that are not a colocated Lean-Agile team.
	Tools	Acquire, build-out, and configure tools and environments for the team: • Agile life-cycle management tool • Application life-cycle management tool • Source control management • Continuous integration server • Test frameworks

✓	Activity	Description
	Plan for assessment	Develop a plan for assessing maturity of Lean-Agile teams. Develop a plan for assessing Agile project success (metrics).
	Transition plan	Develop a plan for making and managing the transition. Perform gap analysis Gather Team Agility Masters into a community of practice. If possible, gather Product Managers into a community of practice.

Iteration 0 Checklist

✓	Activity	Description
	Roles	Product Owner assigned.
	Team	Team identified with all of the needed roles: funded, dedicated to the release, and colocated as much as possible.
	Team environment	Review and define workflow. Review and define visual board. Review policies for stories. Establish logistics for meetings, locations, times, frequency, participants. Tools for testing, coding, integrating, building are selected and installed Established logistics for Daily Stand-Up (time, location, conference call information, website) Agreed to the ground rules for team life Organized (and clean up) the team work space: physical, communication, collaboration Set up the team's project board Established the test and build environment
	Vision	Release vision statement is prepared. The team understands and agrees to the vision, drivers, and expected outcomes for the release.
	Backlog	MBIs introduced and reviewed with team. Refine description, scope, validation, and done criteria. Identify risks, issues, dependencies, uncertainties, and known impediments. Rank, prioritize, and represent these in stories.
	Architecture	Review and define the high level architecture and design. Architectural goals and approach identified, visible. Dependencies and risks Identified, visible. Conceptual design completed.

✓	Activity	Description
	Technology components	Identify technology components. Input technology features and/or stories into team backlogs.
	Feature sequence	Finalize feature sequence by business value and technical dependencies.
	Iteration backlog	Iteration length is set. Iteration backlog established, populated, and visible. Input items into appropriate tools. Team has committed to Iteration 1 plan.
	Story estimation	Review and define complexity factors that will be used for relative sizing in story points. Stories decomposed and right-sized: • Analysis stories if needed • User stories for first feature • Validation criteria for stories are understood Stories are estimated for first few iterations' work.
	Continuous improvement	Intentionally incorporate lessons learned from previous releases.
	Testing agreements	Testing approach (Unit, Integration, and Acceptance) is committed to and visible.
	Reporting and metrics	Book of Work program backlog. Top-line story points. Feature burn-up.
	Artifacts	Identify product documentation that must be maintained.

Iteration Planning Checklist

✓	Activity	Description
	Vision	The team understands and agrees to the iteration vision. All members of the team have a common interpretation of the various terms used in the vision statement.
	Learning from the past	For the kind of work we are going to do in this iteration, are there lessons learned or good practices that we incorporate from anyone on this team, on other teams in the organization, or in the literature?. Invite a "Second Set of Eyes" from outside to think through the plan.
	Story estimation for iteration	Initial estimate of story points for iteration. Stories decomposed and right-sized. Validation criteria for stories are understood. Stories are estimated for first few iterations' work.
	Dependencies and risks	Dependencies and risks identified as stories. Impediments identified, posted on team board, assigned.
	Iteration backlog	Stories are assigned to the iteration backlog. Tasks are identified for the stories in the iteration backlog. Team has committed to iteration plan.
	Commitments	Commit to demonstrating the product to key stakeholders at the end of the iteration. Commit to conducting a retrospection at the end of the iteration. Commit to conducting After Action Reviews as appropriate *during* the iteration.
	Testing agreements	Definition of Done established and documented (unit, integration, acceptance).

✓	Activity	Description
	Team	The team is staffed with all of the needed roles, dedicated to the release, and colocated as much as possible. Team has received required training in Leanban, process, testing, and engineering practices. Artifacts and deliverables determined (and visible).
	Team environment	Tools for testing, coding, integrating, building selected and installed. Established logistics for Daily Stand-Up (time, location, conference call information, portal, etc.). Agreed to the ground rules for team life. Organized (and clean up) the team work space: physical, communication, collaboration. Set up the team's project board. Established the test and build environment.

Iteration Execution Checklist

✓	Activity	Description
	Input tests / code until tests pass	Unit test has been completed and documented. Defects clearly defined, resolved and tested.
	Run functional tests	Functional tests have been completed, documented. Defects clearly defined, resolved, and tested.
	Run user tests	User tests have been completed, documented. Defects clearly defined, resolved, and tested.
	Update business processes and conduct training	All impacted business processes have been assessed. Appropriate changes have been made. Training has been developed and executed. Participants have been certified.
	Product Manager / Analyst acceptance	Product Manager or analyst has formally validated that the story has been completed and is ready to be promoted to production to support the MBI.

Iteration Implementation Checklist

✓	Activity	Description
	Ready for Production	Code is potentially ready to be released.
	Promote	Release has been successfully executed.
	Value extracted from feature for Business	Feature has been successfully executed and the Business value has been achieved.
	Retrospection	Meeting has been conducted to evaluate success of work effort and ways to improve the process.

Daily Stand-Up Checklist

✓	Activity	Description
	Setup	Status of Stories and tasks have been updated before the Daily Stand-up.
	Conduct	Team members showed up on time. All team members were present. Team members talked with each other. Problem-solving and side conversations were kept to a minimum.
	Team and environment	Team work space is clean: physical, communication, collaboration. Team's project board up to date.
	Impediments	Progress on resolving impediments was reported.

Daily Stand-up Ground Rules

Rule	Description
Be consistent	The Daily Stand-up meets every work day at the same time and in the same place.
Show courtesy	Team members show professional courtesy: show up on time, participate, and listen.
Be brief	The Daily Stand-up typically lasts for 15 minutes. Standing up reinforces brevity. Extra discussions and problem-solving is conducted after the meeting, when there is more time.
Hold a team-wide conversation	The Daily Stand-up is for the team's benefit. Each team member is expected to speak and speaks to the whole team. The team is not reporting to the Team Agility Master. The Team Agility Masteris there to help facilitate this conversation but not to lead the session.
Answer all three questions	Each team member answer three questions: • What have you done since the last meeting? • What will you do before next meeting? • What is blocking or slowing you down? Team members can raise issues and obstacles but not propose solutions.
Review impediments	The Team Agility Masterreviews impediments and status.
Swarm	If you have the skills to help with a task and you do not have something else to do, you should volunteer to join the swarm.
Finish work	The priority is to complete stories that are in-work before starting new stories.

Product Demonstration Ground-Rules

Rule	Description
Demonstrate the product as it is	The demonstration is always done with the product as it is. Avoid using slides or other artificial "demo-ware." This is possible because the product should always be tested (perfected), usable, and completed by the end of the iteration.
It is a conversation	The demonstration is a conversation between the whole team, the s, and the stakeholders. They collaborate on what has been done, what has not been done or could not be done, and what the current needs are. It is not a presentation.
Blame-free environment	The team has done what it could do and is forthright about what it could not do. It is not a time to assess blame but to describe the facts about what was really done.
Everyone is present	Everyone has a viewpoint to share. Many ears help maximize communication. As much as possible, avoid redundancy (waste) in meetings.

Release Checklist

✓	Activity	Description
	Vision	has prepared the product vision and release vision.
	Essential stories	Release planning team has identified the essential stories for the next release.
	Release plan	Release planning team has populated releases and prioritized MBIs.
	Demonstration	Team demonstrated product to key stakeholders.
	Retrospection	Retrospection of the release is planned. After Action Reviews conducted as appropriate *during* the release plan.

Schedule: Various Sync Meetings

Here is a suggested schedule for the various synchronization meetings.

Scope of the meeting	Schedule	Issues
Team Agility Masters: ***Team Progress***	Daily	Issues and impediments Improvements and adjustments Planning and coordination
Product Managers: ***Delivery Progress***	Daily	Impediments Readiness and coordination Skills, resources, and capacity
PO and Analysts: ***Value Progress***	Daily	Feature and story preparation Define Business requirements Validate completed stories
Lead Product Manager, PO, Team Agility Master: ***Daily Value Delivery***	MWF	Progress and scope adjustments Priority, sequence, capacity impacts

Part VII: Resources

Lean and Agile are rich domains. This primer touches on a minimum set of concepts you need to remember. This part contains a number of resources to further your knowledge and skills.

Glossary

Communication is essential to quality improvement. A common glossary of terms is critical to communication across teams. Begin with a common glossary of terms and then update them as new terms are discovered or invented.

The terms in this glossary are in common usage. They have been gathered from a number of sources including books, journals, web sites, and field experts.

5S

A basic Lean concept that helps to create an efficient and effective environment for work: "A place for everything and everything in its place." Derived from five Japanese terms beginning with "s" used to create a workplace suited for visual control and lean production. 1) *Seiri* means to separate needed tools, parts and instructions from unneeded materials and to remove the unneeded ones. 2) *Seiton* means to neatly arrange and identify parts and tools for ease of use. 3) *Seiso* means to conduct a cleanup campaign. 4) *Seiketsu* means to conduct *seiri*, *seiton* and *seiso* daily to maintain a workplace in perfect condition. 5) *Shitsuke* means to form the habit of always following the first four S's. An English equivalent might be, "Sort, Simplify, Sweep, Standardize, Sustain."

Good references for 5S are in most Lean books.

Acceptance Test-Driven Development

Acceptance Test-Driven Development (ATDD) employs a test-first mindset where Business, developers, and testers work closely together to write acceptance tests before implementation begins.

Acceptance Testing

Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

Adaptive Management Adaptive Management (AM), also known as Adaptive Resource Management (ARM), is a structured, iterative process of robust decision-making in the face of uncertainty, with an aim to reducing uncertainty over time via system monitoring. In this way, decision-making simultaneously meets one or more resource management objectives and, either passively or actively, accrues information needed to improve future management. Adaptive Management is a tool which should be used not only to change a system but also to learn about the system (Holling 1978). Because Adaptive Management is based on a learning process, it improves long-run management outcomes. The challenge in using the Adaptive Management approach lies in finding the correct balance between gaining knowledge to improve management in the future and achieving the best short-term outcome based on current knowledge (Allan & Stankey 2009).

Agile Agile software development is a conceptual framework for undertaking software engineering projects that embraces and promotes evolutionary change throughout the entire life-cycle of the project. Scrum and XP are two software development methods based on Agile tenets. See also: *Scrum, XP*.

Backlog A list of work to be completed that has various degrees of commitment associated with it depending upon where it is being used. There are three main backlogs in Leanban:

Portfolio backlog. This is the current list of work that is projected for completion at the portfolio level. It is not committed to, but should be ordered in the expected order of completion based on value delivery and cost of completion

Product backlog. This is the current list of work that is projected for the product being built. Typically, funding will be committed for the product backlog. This is organized around MBIs and the work on the product backlog may stop when the business sponsor determines that enough value has been delivered.

Iteration backlog. This is the list of work committed to be done in the current iteration.

Benchmarking A technique in which an organization measures its performance against the performance of best-in-class organizations, determines how those organizations achieved their performance levels, and then uses the information to improve its own performance. Organizations may be internal or external to the company and may be in a different type of industry. Subjects that can be bench-marked include strategies, operations and processes. Sometimes, third-party organizations, such as the American Productivity and Quality Center, are used to conduct the benchmarks to protect confidentiality.

Best Practice A superior method or innovative practice that contributes to the improved performance of an organization, usually recognized as "best" by other peers for given contexts. It is often worthwhile to focus on “good” practices rather than striving for “best” practices.

Brainstorming A technique teams use to generate ideas on a particular subject. Each person on the team is asked to think creatively and write down as many ideas as possible. The ideas are reviewed after the brainstorming session. Most books on facilitation discuss the many approaches to brainstorming.

Bring Me a Rock The common experience of product developers in which a customer or customer's representative (the "Business Analyst") tells the developer to make a product ("bring me a rock"). The developer makes a list of features about the product and proceeds to build it. When the product is finally delivered to the customer, the customer says, "That is not the rock I asked for! Go and bring me another." Iterative methods address this problem by delivering small chunks of finished work to show customers for their approval. It reduces delay in customer feedback and allows the team to adjust to meet the true requirements.

Build Verification Test A group of tests to determine the health of a build at a high level. Typically, these tests exercise the core functionality of code and help determine whether further testing is warranted. These are also known as "smoke tests."

Burn-down Chart A chart showing the rate (story points/day) at which stories or tasks are being completed.

Burn-up Chart A chart showing the rate (story points/day) at which business value is growing.

Business The "Business" is a short-hand for someone from another part of the organization who is not part of the technical development team but has expertise and/or responsibility for profitably addressing customers and markets.

Business Value Business value is what drives product development. It is the measurable or intangible return the Business anticipates it can realize from the product. The Business is responsible for identifying Business value. Business value is what the Business is willing to pay for. Development work should always be traceable to Business value.

Cadence The flow or rhythm of events, especially the pattern in which something is experienced. Cadence in Leanban is the rhythm of when backlogs are refreshed, work is readied for synchronization, retrospections are done, demos are presented, and teams reflect on their work.

Capability The combination of people and people skills, processes, and technologies required in order to equip the Business / customer to use a product. All three are required since until the Business can begin to use a product, the product merely represents *potential*.

Code Analysis The process of checking that code conforms to design guidelines, looking for common coding and design errors per coding standards. The Team makes an agreement to conform to these coding standards; thus, code analysis serves an integrity check with how well the Team is working according to their standards.

Code Coverage A metric used to describe the degree to which source code has been tested. Code coverage is expressed as a percentage of lines of code tested over the total lines of code.

Collaboration Collaboration benefits organizations, teams, and individuals by enabling col-leagues to work together to create new ideas from a starting point that is: uncertain, poorly understood, or innovative; adaptive or intangible; or requires change to accomplish.

Collaboration assumes that no one has the best, correct, or full answer and that the solution can only be found by exploring and synthesizing diverse points of view and using thought experiments and what-if scenarios. Collaboration is creative and the work of humans not machines.

Commonality / Variability Analysis (CVA) A technique that enables developers to write code that can be easily modified later. It complements iterative development. It is described in Shalloway and Trott's *Design Patterns Explained: A New Perspective of Object-Oriented Design.*

Constraint Anything that limits a system from achieving higher performance or throughput; also, the bottleneck that most severely limits the organization's ability to achieve higher performance relative to its purpose or goal.

Consultant An individual who has experience and expertise in applying tools and techniques to resolve process problems and who can advise and facilitate an organization's improvement efforts.

Continuous Process Improvement A philosophy and attitude for analyzing capabilities and processes and improving them repeatedly to achieve customer satisfaction. Also known as Continuous Quality Improvement.

Cooperation Cooperation benefits organizations, teams, and individuals by spreading the work across numerous functions, abilities, and methods (e.g., human and machine can cooperate to produce work).

Cooperation assumes that the work to be done is well understood, generally predictable, and has an accepted action plan. It is about helping to execute an already determined plan. Cooperation improves efficiency.

Cost-Benefit Analysis A method of project evaluation that compares the potential benefits with the anticipated costs.

Cross-Functional Team Teams that have all of the capabilities to deliver the work they've been assigned. Team members can specialize in certain skills but as a whole, the team is capable of delivering what they've been called on to build.

Customer The recipient of the output (product, service, information) of a process. Customers may be internal or external to the organization. The customer may be one person, a department, or a large group. Internal customers (outside of IT) are sometimes called the "Business."

Customer Satisfaction The result of delivering a product, service, or information that meets customer requirements.

Cycle Time The total elapsed time to move a unit of work from the beginning to the end of a process. Cycle time includes process time, during which a unit is acted upon to bring it closer to an output, and delay time, during which a unit of work is spent waiting to take the next action.

Daily Stand-up A stand-up meeting of the team where status is exchanged, progress is observed, and impediments are noted and removed. Typically, these meetings last 15 minutes. Each member answers three questions:

- What did you do since last meeting?
- What do you plan to do before next meeting?
- What is blocking or slowing you down?

Dashboard A type of information radiator that provides management and teams with graphs and reports indicating progress and trends and to identify potential problems.

Demonstrable Product A version of a product that can be demonstrated to customers. It may not be quite ready for release or delivery, since that usually requires additional work (art work, production plans, etc). It is the primary deliverable of an iteration.

Design Pattern A collection of best practices for solving problems in a recurring context. The more general term is "pattern" because patterns are involved with analysis, design, implementation, and testing.

Done The criteria for accepting an MBI, feature, or story as finished. Specifying these criteria is the responsibility of the entire team, including the business.

Elevation A planning chunk that helps the team plan work iterations that minimize risk, increase learning, and coordinate effectively with other teams. Given all of the work required to realize an MBI, the team defines an objective for a particular iteration or set of iterations. The elevation is what is required to achieve that objective and ensure that the resulting code integrates well and maintains technical integrity.

Emergent Design Allowing a design to emerge over time, as part of the natural evolution of a system. Requires good practices and testing to ensure that the system is not inadvertently allowed to decay or become overly complex over time. See *Emergent Design: The Evolutionary Nature of Professional Software Development* by Scott L. Bain.

Error Proofing The ability to catch errors immediately, and to take corrective action to prevent problems down the line. Four strategies for error proofing: 1) eliminate possibility of error, 2) reduce occurrence if elimination is not possible, 3) reduce consequences of errors, 4) capture and address error early if they cannot be eliminated.

eXtreme Programming See: *XP*.

Facilitator An individual specifically trained to enable groups to work more effectively, collaborate, and achieve synergy. The facilitator is a 'content neutral' party; does not take sides or express or advocate a point of view during the meeting, advocates for fair, open, and inclusive procedures to accomplish the group's work.

Feature A feature is a business function that the product carries out. Features are large and chunky and could be implemented by using many stories. Features may be functional or non-functional. Features form a basis for organizing stories.

FIT Framework for Integrated Test

Functional Test The activity that validates a feature against customer requirements. Functional tests are usually done by a tester as part of the customer team.

Gemba The "Gemba" is the place where value-added work is actually being done: a work cell, the developer team room, the help desk, the customer's office. Management must go to these locations to observe, evaluate, coach, and engage with the team. This is in contrast to management practices that rely on management hierarchies, team leads or formal status meetings in conference rooms or management offices.

Happy Path The basic course of action through a single use case.

Harness Tests A group of test scenarios with expected outcomes related to a specific system module to confirm that defects have not been introduced as a result of current iteration programming activity. For example, a pricing test harness would confirm no defects from current pricing module programming.

Impediment A technical, personal, or organizational issue that is preventing or delay in progress on delivering product.

Information Radiator A type of visual control that displays information in a place where passersby can see it and get information about the project without having to ask questions. To be effective, the information must be current and must be easy to see and understand, with sufficient detail to explain status.

Intangible Metrics Intangible metrics and deliverables are indirect drivers of change. For example, culture, leadership, behaviors, mental models, assumptions, teaming dynamics, social physics, and the ability to work on the organization. Intangible drivers of change are used to positively impact technical delivery and, when ignored, often have a negative impact on technical delivery and results.

Integration System Test The activity that verifies that software code does not harm other parts of the software product. Preferably, Integration system testing is done by a tester with automated tools.

Inventory Work items that have been started but not completed.

Iteration A time-boxed event (generally two-weeks) that serves as the time frame for planning, work, demo, delivery and retrospection. In Scrum, an iteration is called a "sprint."

Iteration Planning The activity to prioritize and identify the stories and concrete tasks for the next iteration. Also known as "loading the front burner."

Just-in-Time Design The process of waiting until you know what the design needs to be and then refactoring code to meet these new needs before adding the functionality that is forcing the design change.

Kaizen A Japanese term that means gradual unending improvement by doing little things better and setting and achieving increasingly higher standards. Masaaki Imai made the term famous in his book, *Kaizen: The Key to Japan's Competitive Success.*

Lean The approach that produces value for customers quickly through a focus on reducing delays and eliminating waste which results in increased quality and lower cost.

Lean-Agile An approach to software development that incorporates principles, practices, and methods from lean product development, Agile software development, design patterns, test-driven development, and Agile analysis. Net Objectives advocates by this approach for those who want to be effective in creating products at scale that add value to customers and to the business.

Lean-Agile Team (Team) A cross-functional group comprised of persons with skills who can perform three roles – customer (requirements & validation), developers, and test. The team selects the iteration goal and specifies work results, has the right to do everything within the boundaries of the project guidelines to reach the iteration goal, and organizes itself and its work. Also known as the Team, Scrum Team, Agile team, or a work cell.

Leanban A team-level process based on Lean-Thinking that incorporates Scrum and Kanban practices as appropriate. Leanban was created by Net Objectives to provide a consistent approach for all teams across an organization while still enabling it to be tailored as needed for each team using it.

Manual Test A document that lists the steps that a person follows to complete a test pass. Not automated.

Minimum Business Increment The smallest set of functionality that must be realized in order for the customer to perceive value. Among other things, this means it:

- Adds value for the customers of the business
- Provides valuable feedback that the right functionality is being built
- Provides valuable feedback that the functionality is being built the right way
- Provides functionality that can be verified as an increment that can be delivered
- Enhances the ability of the organization to deliver value in the future

Open-Closed Principle Suggested by Ivar Jacobsen, refined by Bertrand Meyer, and promoted by patterns, the Open-Closed Principle suggests that it is better to create designs that can accommodate change by adding to them, rather than by changing them. "Open-Closed" means we are “open to extension but closed to modification." This is a principle, and it is impossible to follow it literally at all times, but it can guide us in refactoring as well.

Operational Metrics Operational metrics are direct drivers of change and governance. Operational metrics show status across the entire program or project. There are two kinds of operational metrics:

Program- and project-level operational metrics. These are owned by the Product Owner and are calculated weekly. Examples include Release Burn-up, Total Top-Line, satisfaction, and process improvement.

Daily operational metrics. These are owned by the Team Agility Master and are calculated daily. They are based on velocity. Examples include unestimated stories, open stories, open defects, tests passed, open issues and impediments.

Pattern A collection of best practices for solving problems in a recurring context, represented as collections of forces and provide a professional language for high-fidelity communication among developers. The subject of many books including *Design Patterns Explained* (Shalloway and Trott 2004).

PDCA Plan-Do-Check-Act (PDCA) cycle is an iterative four-step problem-solving process for quality improvement. Also known as the Deming Cycle, Shewhart Cycle and Deming Wheel.

- *Plan*: Development of plan to effect improvement.
- *Do*: Plan is carried out.
- *Check*: Effects of plan are observed.
- *Act*: Results studied and learning identified for future usage.

Performance Test A test that ensures the required level of performance of a product is met. This test checks both that the functionality works and that the time required to do the work is acceptable.

Persona A description of the typical skills, abilities, needs, tasks, behaviors, and backgrounds of a particular set of users. As an aggregation, the persona is a fiction but is used to ensure groups of users are accounted for.

Planning The activity that seeks to prioritize and define the stories and tasks for the next iteration. Also known as "loading the product backlog."

Portfolio Level The Portfolio Level includes a diverse group of stakeholders working across organizational boundaries. The Portfolio Level is responsible for priority, the order of work, and deciding what is active. Roles involved in the portfolio include the Value Stream Owners, the Business and Technology Sponsors, and stakeholders. Business strategy flows into the technology organization at the Portfolio Level.

Process A series of actions, changes, or functions performed in the making or treatment of a product.

Product A collection of tangible and intangible features that are integrated and packaged into releases that offer value to a customer or to a market.

Product Vision A short statement of the vision that is driving the project, expressed in business and customer terms while taking technological enablers into consideration: Who it is for, the opportunity, its name, the key benefit and differentiators. Typically, the Product Manager provides the product vision. Sometimes called the "Project Charter."

Program Increment A larger development time-box that uses cadence and synchronization to facilitate planning, coordinate higher-level retrospectives, and aggregate work for feedback. The Program Increment is usually composed of multiple team iterations, often 4-6 team iterations.

Program Level The Program Level is the center of responsibility that is focused on an application area. Work is comprised of releases, enhancements, production support, and maintenance requests. The Program Level includes cross-functional representatives including the Product Manager, Business PM, and Technology Delivery Manager who prioritize work across the value stream. This level also is responsible for program ecosystem and operational metrics.

Project A collection of releases, iterations, team members, and stories that creates a product. May have defined end dates or be on-going. May be:

- All of the software a team or related teams are working on
- A specific product or product group
- A version of a company's product suite
- A specific client implementation

Quality Assurance A role and activity that ensures integrity of the code to be released. The job of QA is to prevent defects from happening in the first place. It is not ‘merely’ to find bugs.

Refactor The disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Refactoring encourages and supports incremental design and implementation, the conversion of proofs of concept and early code into more suitable, stable code

Refactoring to the Open-Closed The same tools and techniques that are used to clean up poor design and code (aka Legacy Refactoring) can be used to change a design just enough to allow for a clean addition or change to it. We take a design that was adequate initially, but which cannot now be changed cleanly to accommodate a new or changed requirement. Refactor until the code follows the open-closed principle, and then we integrate the new code.

Release A version of a product that is released externally to customers. Releases represent the rhythm of the business and should align with defined business cycles. A release contains a combination of Minimum Business Increment that form a releasable product. A release may be internal and may be used for further testing.

Two types of refactoring are fixing bad code and injecting better design into good code as requirements change.

Release Testing

The activity that validates that the product is good enough to release to customers. Release testing is typically performed by a tester and is sometimes called "Quality Assurance" (QA).

Release Testing explores the statement, “We have built something that users can use.” Often, release testing exposes requirements that fail to satisfy actual users’ needs. See also: *Quality Assurance, Test*.

Retrospection

Retrospection is the structured reflective practice to learn and improve based on what has already been done. The purpose of retrospection is to build team commitment and to transfer knowledge to the next iteration and to other teams.

Retrospections must be done at the end of every iteration. A brief version, the “After Action Review” can be done at any time, whenever there is value for the team to stop and learn based on what has been done. In fact, the AAR can be more valuable because it allows the team to change while it can still help their current work.

Role

The role assumed by an individual on a given project. People may serve different roles on different projects or even different iterations. Lean-Agile defines three roles:

- The *Product Manager* represents all customers and manages the Product Backlog.
- The *Team Agility Master* is a facilitator for the team, responsible for coaching and ensuring that the Lean-Agile process is used correctly, for helping to remove impediments.
- The *Development Team* includes Business Analysts, programmers, and testers.
  The team is self-organizing within the guidelines, standards, and conventions of the organization. The team is responsible for completing the work in iterations.

Root Cause

A factor that caused nonconformance to plan and should be permanently eliminated through process improvement. QA helps lead Root Cause Analysis.

Scrum An iterative and incremental Agile software development methodology or framework for managing product development.

Scrum Master See *Team Agility Master*. Also spelled “ScrumMaster.”

SIPOC A diagramming tool used by Six Sigma process improvement teams to identify all relevant elements (suppliers, inputs, process, outputs, customers) of a process improvement project before work begins.

Sprint Scrum’s term for iteration. See *iteration*.

Story An invitation for a conversation about requirements, features, and/or units of business value that can be estimated and tested. Stories describe work that must be done to create and deliver a feature for a product. Stories are the basic unit of communication, planning, and negotiation between the Development Team, Business Owners, and the Product Manager.

Story Point The sizing metric used in the Team Estimation game. They express relative complexity for purposes of estimating how much to place into an iteration. A common approach is to use Fibonacci numbers (1,2,3,5,8, ...) where 1 is low complexity, 8 is more complex) and then use Team Estimation or Planning Poker to get the team to agree on estimates.

Story Point Burn-up The rate of progress the team is achieving in completing the project. It is measured as the number of story points completed per iteration and tracked toward the full project scope. Along with *velocity*, it visibly shows the duration and length of the current project.

Subject Matter Expert (SME) A person who can speak with authority on an aspect of a project or who knows to whom to talk in order to get answers. Subject Matter Experts may represent a technology, the business, the customer, process, or any other topic of importance to the project.

Swarm

The activity of a teamlet that forms to complete a work item. The rules for swarming are:

Focus on one story at a time. Within an iteration, teams should have only a few stories open at a time because the focus is on burning down stories. Do not dissipate energy by focusing on too much at once.

The swarm is the priority. While individuals may work on other tasks, their priority should be the swarmed story.

Swarms are skill-based. If you have the skills to contribute to burning down a story, and you have capacity, you are expected to join in the teamlet, even if it is not exactly in your job description to do so.

Task

Tasks are descriptions of the actual work that an individual (or sub-team) does in order to complete a story. Typically, there are several tasks per story. Tasks have the following attributes:

A *description* of the work to be performed, in either technical or business terms

An *estimate* of how much time the work will take (hours, days)

Exit criteria and *verification* method (test or inspection), and an indication of who will be responsible for the verification. All tasks must be verified, not just “done”

Team Agility Master

The Team Agility Master champions the needs of the team to the organization and for championing the needs of the organization to the team. The Team Agility Master is the team's facilitator, shepherding the team as a servant-leader by creating a positive and safe environment, improving team cohesion, being creative and focused, asking difficult questions about process aimed at challenging the team to improve, bringing issues and impediments to the team, Product Owner, or management, and helping to develop the product backlog.

In Scrum, this is called the "Scrum Master;" however, Team Agility Master is a richer role than is commonly understood in Scrum.

Team Level The Team Level consists of individual teams responsible for producing and implementing a Business value increment, for quality assurance, and for continuous incremental improvement. Work is composed of the stories and tasks for a specific release, enhancements, production support, and maintenance requests.

The Team Level also includes shared services.

Teamlet A group or sub-group of a Lean-Agile team who agree to swarm on a work item (or story or task) in order to complete the work. The teamlet must have all of the knowledge and skills required to perform the work. Teamlets are generally formed at the Daily Stand-up, when the team reviews progress that has been made, and decide what needs to be worked on next. Teamlets are fluid, forming and disbanding based on the requirements of the work items at hand. The teamlet may assign a "Story Captain" to help the teamlet stay on track.

Test Automatic and manual inspections of code and process to ensure correctness and completeness. Types of tests include Unit, Integration, System, Regression, Performance, and User Acceptance (Customer Acceptance)

Test-Driven Development An evolutionary approach to development. In TDD, each test is written before the functional code that makes the test pass. The goal of TDD is specification and not validation, to think through a design before code is written, to create clean code that always works.

Time-box An allocated period of time allowed for getting work done. This means that we commit to completing at a certain time, typically with a certain amount of effort (people and resources). We commit to getting as much work done as possible within this time frame.

Unit Test The activity that verifies that software code matches the design specifications. Typically, unit testing is performed by a developer – by the code creator or by a "buddy" – however, testers often advise developers in testing approaches. Each unit test confirms that the code accurately reflects one intention of the system.

Use Case In Lean-Agile, use cases express the details for a requirement story. If the use case becomes so large that it cannot be implemented in a single iteration, then the requirement story associated with the use case can be broken down into iteration requirement stories first. Alternatively, if a use case is more abstract, it can represent several requirement stories.

Use cases should be expressed in the style of Cockburn detailed use cases. They may be white box or black box and include a main scenario, exceptions, and alternatives. Use cases can refer to business rules and data definitions.

User Acceptance Test (UAT) The activity that verifies that software code matches the business intent. UAT belongs to the customer. They decide what constitutes an acceptable product. Unit tests help ensure that the acceptance tests are not about functional failures, but about the actual acceptability of the approach. Thus, UATs should not result in "it crashed" but «I would like the menus to be more descriptive."

Validation The testing activity that confirms, "you built what I asked for."

Value-Added A term used to describe activities that transform input into a customer (internal or external) usable output. An activity on a project is value-added if it *transforms the deliverables* of the project in such a way that the customer *recognizes* the transformation and is *willing to pay for it*.

Value Stream The set of actions that take place to add value to a customer from the initial request to delivery. The value stream begins with the initial concept, moves through various stages for one or more development teams (where Agile methods begin), and on through final delivery and support.

Value Stream Owner A person responsible for systematic management approach with immediate impact on the critical elements of a company's value streams. Makes change happen across departmental and functional boundaries.

Velocity Velocity measures how many points the team spends during an iteration. The following velocities are used in the planning game to determine how many stories the customer should give to the team to estimate for the iteration:

Story velocity. How many story points the team completes in an iteration.

Task velocity. How many "engineering hours" the team actually can perform in an iteration.

Verification The testing activity that confirms, "I built what I intended to."

Voice of Customer The "voice of the customer" is the term used to describe the stated and unstated needs or requirements of the customer. The voice of the customer can be captured in a variety of ways: Direct discussion or interviews, surveys, focus groups, customer specifications, observation, warranty data, field reports, complaint logs, etc.

Wavelength An approach to transitioning to Lean-Agile based on Lean and Agile principles and practices, technical practices, organizational alignment and transition principles. The goal is to help the organization instantiate a Lean-Agile practice that addresses its own particular organizational dynamics.

Waste Any activity that consumes resources, produces no added value to the product or service a customer receives, or delays development of value. Types of waste include 1) anything that does not add customer value, 2) anything that has been started but is not in production, 3) anything that delays development, 4) extra features, 5) making the wrong thing.

Work Breakdown Structure (WBS) The Work Breakdown Structure is “a deliverable-oriented grouping of project elements which organizes and defines the total scope of the project.” The WBS organizes work according to three primary groups:

- Product work or activities
- Environment and team work or activities
- Business work or activities

Work Cell A cross-functional set of resources and people required to perform work in a value stream. In Lean-Agile, the preferred term for a work cell is "teamlet" because they tend to be temporary, forming to complete a particular work item.

XP (eXtreme Programming) A software development methodology adhering to a very iterative and incremental approach. XP consists of a number of integrated practices for developers and management; the original twelve practices of XP include Small Releases, Acceptance Tests, On-site Customer, Sustainable Pace, Simple Design, Continuous Integration, Unit Tests, Coding Conventions, Refactoring Mercilessly, Test-Driven Development, System Metaphor, Collective Code Ownership, and Pair Programming.

Yesterday’s Weather The expectation that a team will complete as many story points worth of work in the next iteration as they did in the last. Of course, this will only be true after they have done a few iterations and have hit a somewhat steady level. Typically this is after 3-4 iterations. The term comes from the fact that before weather satellites, the most accurate way to predict weather was to say it would be the same as the day before. Hence, “yesterday’s weather” means we expect what happened before.

Recommended Resources

Net Objectives Resources Page

www.netobjectives.com/resources

The main resources page for Net Objectives with sections for Lean, Agile, Scrum, Design, Testing, and Programming.

Net Objectives Agile Design and Patterns Resources

www.netobjectives.com/resources/agile-design-and-patterns

Resources focused on the design, testing, and programming needs of developers. It has a wealth of materials on design patterns, agile design, object-oriented analysis and design.

Acceptance Test-Driven Development Resources

www.netobjectives.com/atdd

Acceptance Test-Driven Development (ATDD) is about communication, collaboration and clarity. This site offers resources help product owners and developers to use ATDD to gain clarity on requirements

Agile Developer Resources

www.netobjectives.com/resources/books/essential-skills-agile-developers

Resources for the book, *Essential Skills for the Agile Developer*.

Team Agility Master Resources

Podcast: Scrum Master Overview, Part 1: *www.netobjectives.com/blogs/scrummaster-overview-part-1*

Bens, Ingrid M, and Janet MacCausland. *Facilitation at a Glance! A Pocket Guide of Tools and Techniques for Effective Meeting Facilitation*. Boston, MA: Goalqpc, 2012

Coaching Resources

Here are resources for learning about coaching, an essential skill for Team Agility Masters. This list is not comprehensive but it should give you a start. An annotated bibliography is on the Net Objectives Coaching page: *www.netobjectives.com/resources/coaching#CoachingBook*

Coaching

Caruso, Joe. *The Power of Losing Control*. 2004.

Rackham, Neil. *SPIN Selling*. McGraw-Hill Education, 1988.

Weinberg, Gerald M. *The Secrets of Consulting: A Guide to Giving and Getting Advice Successfully*. Dorset House Publishing, 1986.

Agile Practices Coaches Should Know

Cohn, Mike. *Agile Estimating and Planning*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.

Denne, Mark, and Jane Cleland-Huang. *Software by Numbers: Low-Risk, High-Return Development*. Upper Saddle River, NJ: Prentice Hall PTR, 2003.

Highsmith, James. *Agile Software Development Ecosystems*. Addison-Wesley Professional, 2002.

Larman, Craig. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2003.

Shalloway, Alan. "Lean Anti-Patterns and What to Do About Them." *Agile Journal*. 2008. *www.agilejournal.com/content/view/553/39*.

Shore, James, and Shane Warden. *The Art of Agile Development*. O'Reilly Media, Inc, 2008.

Issues of Transition, Knowledge Management and Organization

Bridges, William. *Managing Transitions: Making the Most of Change*. De Capo Lifelong Books, 2009.

Bungay, Stephen. *The Art of Action: How Leaders Close the Gaps between Plans, Actions, and Results*. Boston, MA: Nicholas Brealey Publishing, 2010.

Collison, Chris, and Geoff Parcell. *Learning to Fly: Practical knowledge management from some of the world's leading learning organizations*. Chichester, West Sussex: Capstone, 2004.

Heath, Chip, and Dan Heath. S*witch: How to Change Things When Change Is Hard*. Crown Business, 2010.

Kotter, John P. *The Heart of Change: Real-Life Stories of How People Change Their Organizations*. Harvard Business Review Press, 2012.

Mann, David. *Creating a Lean Culture: Tools to Sustain Lean Conversions*. New York: Productivity Press, 2005.

Maurer, Robert. *One Small Step Can Change Your Life: The Kaizen Way*. Workman Publishing Company, 2004.

Rother, Mike. *Toyota Kata: Managing People for Improvement, Adaptiveness and Superior Results*. McGraw-Hill Education, 2009.

Scholtes, Peter R. *The Leader's Handbook: Making Things Happen, Getting Things Done*. New York: McGraw-Hill, 1998.

Townsend, Patrick L, and Joan E. Gebhardt. *How Organizations Learn: Investigate, Identify, Institutionalize*. Milwaukee, WI: ASQ Quality Press, 2007.

Facilitation Resources

Here are resources for learning about facilitation, an essential skill for Team Agility Masters. This list is not comprehensive but it should give you a start.

General Facilitation

Bens, Ingrid M, and Janet MacCausland. *Facilitation at a Glance! A Pocket Guide of Tools and Techniques for Effective Meeting Facilitation*. Boston, MA: Goalqpc, 2012.

Cameron, Esther. *Facilitation Made Easy: Practical Tips to Improve Meetings & Workshops*. 2nd. London: Kogan Page, 2001.

Farrell, John D, and Richard G Weaver. *Practical Guide to Facilitation: A Self-Study Resource*. Amherst, MA: HRD Press, 2000.

Gottlieb, Marvin R. *Managing Group Process*. Santa Barbara, CA: Praeger Publishers, 2003.

Hogan, Christine. *Understanding Facilitation: Theory & Principles*. London, UK: Kogan Page, 2002.

Straus, David. *How to Make Collaboration Work: Powerful Ways to Build Consensus, Solve Problems, and Make Decisions*. San Francisco, CO: Berrett-Koehler Publishers, 2002.

Icebreakers

Rostron, Sunny Stout. *Accelerating Performance: Powerful Net Techniques to Develop People*. London, UK: Kogan Page, 2002.

Lucas, Robert W. *The Creative Training Idea Book: Inspired Tips and Techniques for Engaging and Effective Learning*. New York, NY: AMACOM, 2003.

Process Mapping

Straus, David. *How to Make Collaboration Work: Powerful Ways to Build Consensus, Solve Problems, and Make Decision*s. San Francisco, CO: Berrett-Koehler Publishers, 2002.

Maurya, Ash. *Running Lean: Iterate from Plan A to a Plan That Works*. O'Reilly Media, Inc., 2012.

Warner, Jon and Paul Wildman. T*he Problem-Solving & Decision-Making Toolkit: 32 Fully Reproducible, Ready-to-Use Tools to Help You Build Your Know-How to Solve Problems, Be More Creative, and Make Better Decisions*. HRD Press, 2003.

Problem Solving

Gottlieb, Marvin R. *Managing Group Process*. Santa Barbara, CA: Praeger Publishers, 2003.

Straus, David. *How to Make Collaboration Work: Powerful Ways to Build Consensus, Solve Problems, and Make Decisions*. San Francisco, CO: Berrett-Koehler Publishers, 2002.

Hohmann, Luke. *Innovation Games: Creating Breakthrough Products Through Collaborative Play*. Addison-Wesley Professional, 2006.

Facilitation with Virtual and Global Teams

Marquardt, Michael J., and Lisa Horvath Davies. *Global Teams: How Top Multinationals Span Boundaries and Cultures with High-Speed Teamwork*. Boston, MA: Nicholas Brealey America, 2012.

Paulenn, David. *Virtual Teams: Projects, Protocols, and Processes.* Hershey, PA: Idea Group Publishing, 2003.

Recommended Web Sites

Here are a number of very helpful web sites.

- Agile Alliance: *www.agilealliance.org*
- Agile Data: *www.agiledata.org*
- Agile Modeling: *www.agilemodeling.com*
- Code Qualities and the Open-Closed principle: *www.objectmentor.com/resources/articles/ocp.pdf*
- eXtreme Programming: *www.extremeprogramming.com*
- Gemba Panta Rei: *www.gembapantarei.com*
- Jim Highsmith: *www.jimhighsmith.com*

Recommended Books

A complete annotated bibliography may be found at *www.netobjectives.com/resources/bibliography*.

Bain, Scott L. *Emergent Design: The Evolutionary Nature of Professional Software Development*. Upper Saddle River, NJ: Addison-Wesley Professional, 2008.

Beck, Kent, and Cynthia Andres. *Extreme Programming Explained: Embrace Change* (2nd Edition). Boston, MA: Addison-Wesley Professional, 2004.

Highsmith, James. *Agile Software Development Ecosystems*. Boston, MA: Addison-Wesley Professional, 2002.

Larman, Craig. *Agile and Iterative Development: A Manager's Guide*. Boston, MA: Addison-Wesley Professional, 2003.

Osterwalder, Alexander and Yves Pigneur. *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. John Wiley & Sons, 2010.

—. *Value Proposition Design: How to create Products and Services Customers Want*. John Wiley & Sons, 2015.

Mann, David. *Creating a Lean Culture: Tools to Sustain Lean Conversions*. New York: Productivity Press, 2005.

Meszaros, Gerard. *xUnit Test Patterns: Refactoring Test Code*. Upper Saddle River, NJ: Addison-Wesley Signature Series, 2007.

Pugh, Ken. *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration*. Boston, MA: Addison-Wesley Professional, 2011.

Reinertsen, Donald G. *The Principles of Product Development Flow: Second Generation Lean Product Development*. Redondo Beach, CA: Celeritas Publishing, 2009.

Saaty, Thomas L. *Decision Making for Leaders: The Analytic Hierarchy Process for Decisions in a Complex World*, RWS Publications, 3rd revised edition, September 2012.

Scholtes, Peter R. *The Leader's Handbook: Making Things Happen, Getting Things Done.* New York: McGraw-Hill, 1998.

Shalloway, Alan. "Lean Anti-Patterns and What to Do About Them." *Agile Journal*. 2008. *www.agilejournal.com/content/view/553/39/* (accessed 02 02, 2009).

Shalloway, Alan, and James R Trott. *Design Patterns Explained: A New Perspective on Object-Oriented Design,* Second Edition. Boston, MA: Addison-Wesley Professional, 2004.

Shalloway, Alan, Guy Beaver, and James R Trott. *Lean-Agile Software Development: Achieving Enterprise Agility*. Boston, MA: Addison-Wesley, 2009.

Shalloway, Alan, Scott Bain, Ken Pugh, and Amir Kolsky. *Essential Skills for the Agile Developer: A Guide to Better Programming and Design*. Boston, MA: Addison-Wesley Professional, 2011.

Sheridan, Richard. *Joy, Inc: How We Built a Workplace People Love*. New York: Portfolio Hardcover. 2013.

Sutton, James, and Peter Middleton. *Lean Software Strategies: Proven Techniques for Managers and Developers*. New York: Productivity Press, 2005.

Townsend, Patrick L, and Joan E Gebhardt. *How Organizations Learn: Investigate, Identify, Institutionalize*. Milwaukee, WI: ASQ Quality Press, 2007.

Weinberg, Gerald M. *An Introduction to General Systems Thinking*. Weinberg & Weinberg, 2011.

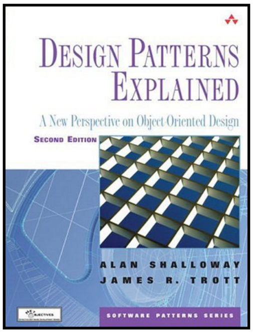
Womack, James P, and Daniel T Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. New York: Simon & Schuster, 1996.

Books from Net Objectives

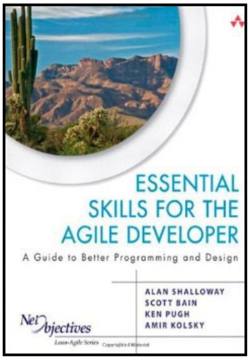
Net Objectives consultants have written a variety of books and resources for business-driven software development. You can find them and other helpful resources at *www.netobjectives.com/marketplace*.

In the approximately ten years since the publication of the *Design Patterns: Elements of Reusable Object-Oriented Software,* this practice has moved from being an esoteric part of computer science research to the mainstream of software engineering. Yet despite their widespread acceptance, design patterns are frequently misunderstood.

Design Patterns Explained, Second Edition provides the reader with a gentle yet thorough introduction to design patterns and recent trends and developments in software design.

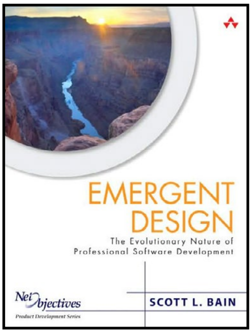

DESIGN PATTERNS EXPLAINED
A New Perspective on Object Oriented Design
SECOND EDITION
ALAN SHALLOWAY
JAMES R. TROTT
SOFTWARE PATTERNS SERIES

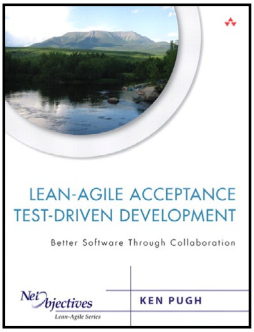
Scott L. Bain integrates the best of today's most important development disciplines into a unified, streamlined, realistic, and fully actionable approach to developing software. Drawing on patterns, refactoring, and test-driven development, Bain offers a blueprint for moving efficiently through the entire software life cycle, smoothly managing change, and consistently delivering systems that are robust, reliable, and cost-effective.


ESSENTIAL SKILLS FOR THE AGILE DEVELOPER
A Guide to Better Programming and Design
Net Objectives
ALAN SHALLOWAY
SCOTT BAIN
KEN PUGH
AMIR KOLSKY

Essential Skills for the Agile Developer: A Guide to Better Programming and Design answers the question many developers have after taking some initial Agile/Scrum training – "OK, how do I write code now that we are building our software in iterations?" This book provides over a dozen proven practices that help developers improve their coding practices and make their code more easily changeable and maintainable in Agile projects.

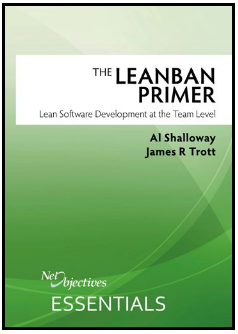
For software to consistently deliver promised results, software development must mature into a true profession. *Emergent Design* points the way. As software continues to evolve and mature, software development processes become more complicated, relying on a variety of methodologies and approaches.


EMERGENT DESIGN
The Evolutionary Nature of Professional Software Development
Net Objectives
Product Development Series
SCOTT L. BAIN

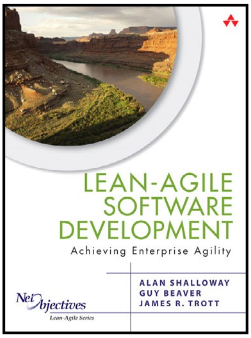


Software development projects have been adopting agility at a rapid pace. Although agility provides quicker delivery of business value, lean principles suggest reducing waste, delays, and hand offs can provide even faster delivery. With *Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration* to help, the business customer, the tester, and the developer collaborate to produce testable requirements.

These acceptance tests form the detailed specification of how the software should work from an external point of view. They help the customer to clarify their needs, the developer to have an objective to code towards, and the tester to plan for more than just functional testing.



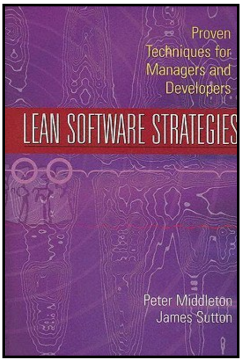
The *Leanban Primer* is a useful reference for teams who have had some basic training and want to use Leanban practices including Scrum and kanban in the context of Lean. Topics include what is Leanban, essential competencies, how to get started, relationship with Lean, roles, estimation, iterations, quality and testing, and communication. This primer offers a complete set of checklists for teams and resources teams need to succeed.



Lean-Agile Software Development: Achieving Enterprise Agility addresses how to drive maximum value from Lean development and avoid or fix the mistakes that prevent software teams from succeeding with Lean, the crucial make-or-break details that team leaders and developers need to succeed with Lean processes, and why many teams fall back on ineffective processes that compromise their commitment to Lean software development.

More and more software organizations are recognizing the potential value of Lean techniques for improving productivity and driving more business value from software. It requires clarity, knowledge and skills that many organizations haven't developed.

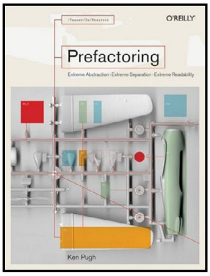
This book brings together practical insights every organization needs to succeed with Lean. The authors answer the most important questions about Lean development: What tools can I use to successfully implement Lean in my company? How do I transition to Lean Software Development? How do I correct specific counterproductive practices that stand in my way? How do I identify waste within my company?"



Lean Software Strategies: Proven Techniques for Managers and Developers shows how the most advanced concepts of lean production can be applied to software development and how current software development practices are inadequate.

Lean production, which has radically benefited traditional manufacturing, can greatly improve the software industry with similar methods and results. This transformation is possible because the same over-arching principles that apply in other industries work equally well in software development. The software industry follows the same industrial concepts of production as those applied in manufacturing; however, the software industry perceives itself as being fundamentally different and has largely ignored what other industries have gained through the application of lean techniques.

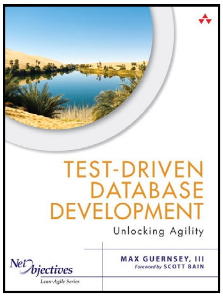
Written for software engineers, developers, and leaders who need help creating lean software processes and executing genuinely lean projects, this book draws on the personal experiences of the two authors as well as research on various software companies applying lean production to software development programs.


O'REILLY
Prefactoring
Extreme Abstraction · Extreme Separation · Extreme Readability
Ken Pugh

Prefactoring approaches software development of new systems using lessons learned from many developers over the years. It is a compendium of ideas gained from retrospectives on what went right and what went wrong in development. Some of these ideas came from experience in refactoring. Refactoring is improving the design of existing code to make it simpler and easier to maintain.

This practical, thought-provoking guide details prefactoring guidelines in design, code, and testing. These guidelines can help you create more readable and maintainable code in your next project. To help communicate the many facets of this approach, Prefactoring follows the development of a software system for a fictitious client, named Sam, from vision through implementation.

In *Test-Driven Database Development*, Max Guernsey, III shows how to adapt Test-Driven Development (TDD) to achieve the same powerful benefits in database design and development.



Guernsey explains why TDD offers so much potential to database practitioners and how to overcome obstacles such as the lack of conventional "testable classes." He shows how to use "classes of databases" to manage change more effectively; how to define testable database behaviors; how to maximize long-term maintainability by limiting a database's current scope; and how to use "emergent design" to simplify future expansion.

This book is a guide to applying the proven practice of TDD to database needs: organizing and optimizing the organization's data for a significant competitive advantage.

Ordering and Licensing Information

This primer is available in a volume discount from the Net Objectives Marketplace. For pricing, visit *www.netobjectives.com/marketplace.*

License (Private Label) this Primer for Your Company

Many companies have found it useful to adapt this primer to support their own Agile methodology. We can work with you to create a print version or to embed it in your web-based tool.

Please contact *info@netobjectives.com*.

Index