LeanDog Agile Discussion Guide Notes  
Notes By: Spencer Peterson

Last Updated: April 2018

Based on Book Version: 3.2

Table of Contents

[Forward 8](#_Toc511046409)

[Chapter 1 – Introduction to Agile 9](#_Toc511046410)

[Essentials vs Advanced 9](#_Toc511046411)

[Agile Manifesto 9](#_Toc511046412)

[Individuals and Interactions 9](#_Toc511046413)

[Working Software 9](#_Toc511046414)

[Customer Collaboration 9](#_Toc511046415)

[Responding to Change 10](#_Toc511046416)

[What Does It Mean to “Be Agile” 10](#_Toc511046417)

[Documented Benefits of Being Agile 10](#_Toc511046418)

[Chapter 2 – Agile Concepts 10](#_Toc511046419)

[Agile Concepts 10](#_Toc511046420)

[Essentials 10](#_Toc511046421)

[Whole Team 11](#_Toc511046422)

[Essentials 11](#_Toc511046423)

[Advanced 11](#_Toc511046424)

[Open Workspace 11](#_Toc511046425)

[Essentials 11](#_Toc511046426)

[Advanced 12](#_Toc511046427)

[T-Shaped People 13](#_Toc511046428)

[Essentials 13](#_Toc511046429)

[Advanced 13](#_Toc511046430)

[Sustainable Pace 13](#_Toc511046431)

[Essentials 13](#_Toc511046432)

[Advanced 14](#_Toc511046433)

[Information Radiators 14](#_Toc511046434)

[Essentials 14](#_Toc511046435)

[Advanced 14](#_Toc511046436)

[Clear Communication 14](#_Toc511046437)

[Essentials 15](#_Toc511046438)

[Frequent Releases 15](#_Toc511046439)

[Essentials 15](#_Toc511046440)

[Advanced 15](#_Toc511046441)

[Story Card Wall 15](#_Toc511046442)

[Essential 15](#_Toc511046443)

[Advanced 16](#_Toc511046444)

[Agile Triangle 16](#_Toc511046445)

[Essentials 16](#_Toc511046446)

[The Traditional Triangle 17](#_Toc511046447)

[The Agile Triangle 17](#_Toc511046448)

[Cone of Uncertainty 17](#_Toc511046449)

[Essentials 17](#_Toc511046450)

[Chapter 3 – Agile Process 17](#_Toc511046451)

[Agile Process 17](#_Toc511046452)

[Essentials 17](#_Toc511046453)

[Kanban 18](#_Toc511046454)

[Essentials 18](#_Toc511046455)

[Advanced 18](#_Toc511046456)

[Cycle Time 18](#_Toc511046457)

[Scrum Framework 18](#_Toc511046458)

[Essentials 18](#_Toc511046459)

[Story Card Writing 19](#_Toc511046460)

[Essentials 19](#_Toc511046461)

[Advanced 19](#_Toc511046462)

[Invest Rule 19](#_Toc511046463)

[Values, Roles, and Goals 20](#_Toc511046464)

[The 3 C’s 20](#_Toc511046465)

[Estimation & Sizing 20](#_Toc511046466)

[Essentials 20](#_Toc511046467)

[Advanced 21](#_Toc511046468)

[Sprints/Iterations 21](#_Toc511046469)

[Essentials 21](#_Toc511046470)

[Advanced 21](#_Toc511046471)

[Customer Collaboration 22](#_Toc511046472)

[Essentials 22](#_Toc511046473)

[Advanced 22](#_Toc511046474)

[Chapter 4 – Leadership 22](#_Toc511046475)

[Leadership Role 22](#_Toc511046476)

[Essentials 22](#_Toc511046477)

[Advanced 23](#_Toc511046478)

[Servant Leadership 23](#_Toc511046479)

[Essentials 23](#_Toc511046480)

[Six Thinking Hats 23](#_Toc511046481)

[The Four Major Benefits of the Six Thinking Hats® 24](#_Toc511046482)

[Collaboration 8 24](#_Toc511046483)

[The 7 Levels of Authority (Plus Joker) 24](#_Toc511046484)

[Fist To 5 25](#_Toc511046485)

[Essentials 25](#_Toc511046486)

[Chapter 5 – Scrum Master/Iteration Manager 25](#_Toc511046487)

[Scrum Master/Iteration Manager 25](#_Toc511046488)

[Essentials 25](#_Toc511046489)

[Daily Scrum/Stand Up 26](#_Toc511046490)

[Essentials 26](#_Toc511046491)

[Advanced 26](#_Toc511046492)

[Sprint/Iteration Planning 26](#_Toc511046493)

[Essentials 26](#_Toc511046494)

[Show & Tell 27](#_Toc511046495)

[Essentials 27](#_Toc511046496)

[Retrospectives 27](#_Toc511046497)

[Essentials 27](#_Toc511046498)

[Advanced 27](#_Toc511046499)

[Velocity 28](#_Toc511046500)

[Essentials 28](#_Toc511046501)

[Roadblocks 28](#_Toc511046502)

[Essentials 28](#_Toc511046503)

[Advanced 28](#_Toc511046504)

[Chapter 6 – Quality Assurance 29](#_Toc511046505)

[Quality Assurance Role 29](#_Toc511046506)

[Essentials 29](#_Toc511046507)

[Advanced 29](#_Toc511046508)

[Exploratory Testing 29](#_Toc511046509)

[Essentials 29](#_Toc511046510)

[Advanced 30](#_Toc511046511)

[Automated Regression Testing 30](#_Toc511046512)

[Essentials 30](#_Toc511046513)

[Advanced 30](#_Toc511046514)

[Acceptance Test Driven Development 30](#_Toc511046515)

[Essentials 31](#_Toc511046516)

[Advanced 31](#_Toc511046517)

[Chapter 7 – Developer 31](#_Toc511046518)

[Developer Role 31](#_Toc511046519)

[Essentials 31](#_Toc511046520)

[Advanced 32](#_Toc511046521)

[Collective Code Ownership 32](#_Toc511046522)

[Essentials 32](#_Toc511046523)

[Continuous Integration 32](#_Toc511046524)

[Essentials 32](#_Toc511046525)

[Advanced 32](#_Toc511046526)

[Simple & Evolutionary Design 33](#_Toc511046527)

[Essentials 33](#_Toc511046528)

[Paired Programming (Pairing) 33](#_Toc511046529)

[Essentials 33](#_Toc511046530)

[Advanced 33](#_Toc511046531)

[Test Driven Development 33](#_Toc511046532)

[Essentials 34](#_Toc511046533)

[Technical Debt 34](#_Toc511046534)

[Essentials 34](#_Toc511046535)

[Advanced 34](#_Toc511046536)

[Spikes 34](#_Toc511046537)

[Essentials 34](#_Toc511046538)

[Chapter 8 – Product Owner 35](#_Toc511046539)

[Product Owner Role 35](#_Toc511046540)

[Essentials 35](#_Toc511046541)

[Value Stream Mapping 35](#_Toc511046542)

[Essentials 35](#_Toc511046543)

[Demand Management 35](#_Toc511046544)

[Essentials 35](#_Toc511046545)

[Advanced 36](#_Toc511046546)

[Product Backlog 36](#_Toc511046547)

[Essentials 36](#_Toc511046548)

[Release Planning 36](#_Toc511046549)

[Essentials 36](#_Toc511046550)

[Advanced 37](#_Toc511046551)

[Chapter 9 – User Experience 37](#_Toc511046552)

[User Experience Designer 37](#_Toc511046553)

[Essentials 37](#_Toc511046554)

[Advanced 37](#_Toc511046555)

[Personas 37](#_Toc511046556)

[Essentials 38](#_Toc511046557)

[Story Mapping 38](#_Toc511046558)

[Essentials 38](#_Toc511046559)

[Low-Fidelity Prototyping 38](#_Toc511046560)

[Essentials 39](#_Toc511046561)

[Advanced 39](#_Toc511046562)

# Forward

The goal of these notes was to strip out the important information from the Agile Discussion Guide into a compact way of reading, that allowed for indexing and quick reference moving forward in your agile adventures. What I learned, was the Agile Discussion Guide was *already* very stripped down and to the point. But the idea still worked.

The main structure that this booked and these notes follow are for each Chapter, there are topics, and for each topic that are essential and advanced information and practices. They are each color coded throughout this document. Red usually refers to ideas or terms not defined in this document, or in the book. I tried to find these and define them in here with a sub bullet, but nobody is perfect.

95% of this I will not take credit for making. It is almost copied word for word from Agile Discussion Guide from LeanDog. Why not read the book? Well, I made this to be accessible and compact, and to be a living document like the book. The table of contents will allow you to quickly navigate to any topic. And with the color coordination (ripped from the book), you can quickly see if there are things you are missing, or if you want to take things to the next step.

Finally, this is a broad document and can sometimes be vague, and this is for good reason. No two agile teams will be the same, no two teams will have the same process. There are ideas in this that need to be applied to each team individually.

Finally, additional links and references can be found in the book online, and I will provide a link to the book as well, if you want to get to the raw material.

**If you are looking for different *Agile Processes*, start at Chapter 3. If you are looking for information on how to be an agile *developer,* start at Chapter 7. If you are looking for the roles like *Scrum Master and Product Owner*, start at Chapters 5 and 8 respectively. *If you are new to Agile*, start at the very beginning.**

**Warning: If you are not using Word, I cannot guarantee that the Table of Contents, Reference tab, or anything else fancy will work. I would highly recommend using Word for this.**

**Also, if you find typos are just poorly documented information, email me at** [**speterson2814@gmail.com**](mailto:speterson2814@gmail.com)**, I would like to fix it for myself!**

***Have fun being agile!***

# Chapter 1 – Introduction to Agile

## Essentials vs Advanced

* Essentials are crucial to adopting Agile, advanced are for improving the essential

## Agile Manifesto

* **Individuals and Interactions** *over* Processes and Tools
* **Working Software** *over* Comprehensive Documentation
* **Customer Collaboration** *over* Contract Negotiation
* **Responding to Change** *over* Following a Plan
  + While there ***is*** value on the right side, we value to left side ***more***

## Individuals and Interactions

* Open Workspace
* Collective Code Ownership
* Retrospectives
* T-Shaped People
* Cross Functional Pairing
* Whole Team
* Daily Scrum/Standup
* Paired Programming
* Sustainable Pace

## Working Software

* Collective Code Ownership
* Exploratory Testing
* Continuous Integration
* Frequent Releases
* Simple & Evolutionary Design
* Test Driven Development
* Automated Regression Testing
* Spikes

## Customer Collaboration

* Show & Tell
* Story Card Writing
* Acceptance Test Driven Development
* Story mapping
* Value Stream Mapping
* Release Planning
* Information Radiators
* Low-Fidelity Prototyping
* Iteration Planning
* Story Card Wall
* Personas

## Responding to Change

* Show & Tell
* Demand Management
* Sprint Iteration
* Estimation Sizing
* Velocity
* Frequent Released
* Product Backlog
* Cone of Uncertainty
* Usability Testing

### What Does It Mean to “Be Agile”

* Not about doing something, like Scrum, but about upholding the values of Agile

### Documented Benefits of Being Agile

* Changing of Priorities
* Productivity
* Project Visibility
* Team Morale
* Time to Market
* Software quality
* IT & Business Objectives Alignment
* Risk
* Development Process Engineering Discipline
* Engineering Discipline

# Chapter 2 – Agile Concepts

## Agile Concepts

### Essentials

* Do we have everyone on the team that we need?
* Is our team located in the same space?
* Are people broadly skilled, knowledgeable, and able to help each other?
* Do we have the workload set so we don’t need heroic efforts?
* Is it easy for the team and non-team members to understand where we stand?
* Do we have inexpensive and simple ways to communicate?

## Whole Team

* The whole team approach states that everyone is accountable for the quality of the product. This approach brings the entire team together to work as a unit and share responsibility for producing high quality software. Whole team is the GLUE to agile practices; it holds all the other practices together

### Essentials

* While team shares workspace
* Team members are 100% dedicated to the team’s work, not allocated to other work outside the team
* Work comes to the team; the team doesn’t form based on the funding of projects
* Avoid shared talent between teams; DBAs and other technical specialists are available on demand, but are preferably T-Shaped and on the team
* T-Shaped people – Broad skills with an area of depth
  + I shaped lacked the horizontal T
  + X Shaped people require mastery first
* Team is led, not managed
* Team communicated and collaborates continuously
* Team is accountable for results

### Advanced

* Business partners are co-located with team
* Team is not named after the project or department, they have their own identity

## Open Workspace

### Essentials

* Workspace
  + General rule is 150 square feet per team member; adequate space for moving, sitting, and collaboration
  + There are no assigned seats
  + Teams do not have other personal space (like cubes)
  + Two small breakout rooms are available for meetings or privacy
  + White boards, cork boards or flip charts
  + Team uses open seating to facilitate paired programming and communication
  + Leadership sits with the team
  + Have teams in a highly visible area, be proud to show the space to customers
  + Team Members have a locker room to store personal belongings
  + Furniture is moveable without permission; team has ability to form the space
  + Allow picture skins
  + Allow team to be create and introduce fun into the workspace
* Team
  + Noise equals collaboration and communication, quiet is “bad”
    - Parent’s Ear Rule: Bad sounds are complete silence or kicking and screaming
  + “Hey Team” cooperation when roadblocks are discovered
  + Teams sit facing each other, not with backs to each other
  + Requires that facilities and leadership understand the goals, and is engaged with the team
  + All rules deserve team discussion
* Equipment
  + Tables have wheels or can easily be moved
  + Power comes from floor or ceiling, not from cubicle walls that don’t move
  + Team room has one team phone, not one per person. Provide cell phones, phone booths, skype IDs to co-workers
  + Easy or permanent access to video conferencing equipment to connect with customers and other teams. Face-to-face communication is valued highly
  + Headphones available
  + Wireless networks

### Advanced

* Workspace
  + Provide white noise generators
  + Food station with coffee nearby, but not on center island
  + Rolling tool box for office supplies, clearly labeled and replenished
  + Teams should have access to bright desirable spaces and not be relegated to closed conference rooms and basements
  + One large breakout room available for meetings or privacy
  + Windows are tools, the team can have things taped to them and use dry erase markers on them
  + Two phone booths available for personal conversations
* Team (Working Agreements)
  + Pick one day a week, perhaps after the Friday daily Stand Up, to clean the work area for 15 minutes; ensure that visible info is necessary (look for stale information)
  + Music is OK, team agrees on rules around this
  + If something isn’t working for facilities management, voice concerns at the retrospective meeting
  + If something isn’t working for the team, it will go on the roadblock wall
  + All rules should be prominent in team space (e.g.: leave three feet around perimeter for fire safety, no cords that people can trip on, etc.)
* Equipment
  + Projector(s) or TVs readily available, set up, and ready to use
  + Magnetic whiteboards allow teams to slide cards around easily
  + Each conference room has a high-quality speaker phone/video conference phone
  + Large monitors set up to connect laptops, have two side-by-side
  + Toys should be prevalent: fun fosters innovation
  + Team members choose type of laptop (Apple or PC) OK to have both kinds on the team if team agrees
  + Everyone has laptops to improve mobility, encourage to take home and study craft
  + Team has easy access to a plotter to print information radiators or TVs to radiate information digitally
    - Information radiators defined later
  + Printer is capable of printing on index cards
  + Have hand sanitizer readily available to prevent spreading germs

## T-Shaped People

A T-Shaped person is an individual who has deep knowledge of a specialized skill set but also has acquired tangential, related skills. T-Shaped people are also interested in continuously broadening their knowledge as well as deepening a core skill set.

### Essentials

* People are encouraged to learn and pair in all roles
* Collaboration between team members to build T-Shaped People
* Team members cross-train each other in technical and domain specialized knowledge
* Team creates an environment that facilitates and encourages continual learning
* Activities such as paring, job shadowing, lunch-n-learns, book clubs, and open discussions are used frequently

### Advanced

* Anyone on the team can pick up any story card
* Team members are encouraged to pick story cards in areas with which they are unfamiliar as a learning challenge
* Team will focus learning activities around bottlenecks
* Create cross-team communities of practice around technical specialties, domain knowledge areas, or any other area of interest

## Sustainable Pace

A Sustainable Pace is a constant pace that a development team should be able to maintain indefinitely, to ensure the team has time to plan, think, rest, and deliver effectively

### Essentials

* Team members typically work a 40-hour week
* Included daily work responsibilities, training opportunities, and sustainable fun
* Must give enough time to include all regular team activities such as initiation, planning, development, testing, and deployment
* Avoid the death march or the need for heroic efforts; instead, focus on repetitive delivery of high-quality software

### Advanced

* Team pace should include enough slack, mental downtime, or development time to allow for other activities which foster creativity and innovation
* Team is evaluated on performance, not pressure (Result-Only Focus)
* All regular work week constraints are removed from the core team (i.e. All team members are required to be at their desks for work day by 8:00 am (or the established time)

## Information Radiators

Information Radiators display important project information simply, communicating information even from across the room. Use IRs to map project statuses so the team knows the status at all times: what’s coded, what’s verified, and what’s to come

### Essentials

* Anyone should be able to review the information displayed and easily understand it
* Key charts include:
  + Release Plan and Roadblocks
  + Product Value Map
  + Burn Up and Velocity
  + Story Card Wall
  + Unit Test and Acceptance Test Coverage
  + Used for continuous communication and information radiation
  + Presented in a casual way (no complexity, just charts on walls)
  + Team Blocks and Escalation

### Advanced

* Team decides what is visible
* Flow and information is centered around the workspace. The walls should tell a story to the team and the customer
* Charts track everything from future project requests to fully-tested stories
* Other helpful items to be considered
  + Organizational Value map
  + Program Demand Management
  + Persona Map
  + Ideation Wall (to capture and compare new ideas or potential work

## Clear Communication

Clear communication required a validation of shared understanding. Diagrams, flowcharts, and other visual aids are an excellent way to ensure that ideas discussed are actually understood in the intended manner

### Essentials

* Clear communication is the foundation
  + “I’m glad we all agree” is great, but do we agree more than just in our heads?
* Get mental models our on the table
  + “Ah!”, this means differences in agreement are found
* An explicit model allows convergence through iteration
  + “Ah!”, seeing the idea develop and be described outside the mind helps clear the murky parts
* A genuinely shared understanding is then the result of this process

## Frequent Releases

Frequent releases are intended to shorten feedback cycles and improve responsiveness by deploying code in short cycles. This improves productivity by providing more opportunities to handle new and changing requirements or adjusting priorities of planned work in response to business or user needs.

### Essentials

* At the end of each iteration the software should be ready for production, the business determines if it is released
* Apply fixed capacity against a fixed date, prioritizing cards by business value. The capacity and schedule are fixed, but the scope is flexible
* Fixed capacity and fixed scope allow for flexible dates
* The team (including the product owner) must agree on what a completed state means for individual features and releases

### Advanced

* Working software is frequently delivered to business partners in small, functional chunks
* Releases are scheduled based on the input and needs of business partners
* Teams strive to reduce the risk of large, “big bang” releases by reducing the size of each release, and automation the release process to improve repeatability and minimize the cost of releasing frequently.
* Create a release schedule to help shape the direction of the product or initiative

## Story Card Wall

The Story Card Wall is an information radiator tool used in the team’s workspace. It is an effective way to display the status of each card in its current iteration. The story card wall is broken up into columns reflecting each function necessary to the process. Story cards can be index cards or sticky notes

### Essential

* Must be on a physical wall in the team’s space
* Positioned in a clean, well-lit place
* All team members move cards
* Cards should be written, not printed (easy to change vs edit tool/reprint
* The definition of ‘complete’ should be clear
* Personal WIP limits: no more than one card per person can be in play at a time
  + WIP is Work in Progress
* Everyone on the team can read and explain the story card wall
* Use color coding to differentiate cards of different work types – legend Minimal Marketable Feature Sets (MMF)

### Advanced

* Portable magnetic dry erase card walls are best
* Enough room around wall for team to gather
* Pictures on magnets show card owner
* Retrospective items on wall are not necessarily part of a Story Card Wall. However, some teams do maintain an area of list topics that team members want to discuss at an upcoming retrospective
* Stand Ups should be in front of the wall
* Visual sign explains specific transition criteria for cards to move between statuses on the board (e.g. what has to be done/verified for each card in “Development” before it can move to “Testing”) (This sounded weird)
* Anything in progress has a name attached to it
* Cards are easy to read from a distance
* Team establishes an overall WIP limit (in addition to personal WIP limits) to reflect team’s sustainable capacity
* Team writes “Created”, “Started”, and “Completed/Accepted” dates on cards to facilitate tracking of lead time/cycle time
* On Hold Metrics: Some teams keep track of “Hold” time – i.e. when a card cannot be worked on continuously once it has been stated. Cards are annotated with hash marks to represent any full or half-day increments during which the cards had to be placed “On Hold” for some reason. (Some overlap with Roadblocks, but sometimes reveals/highlights different process problems)

## Agile Triangle

Agile teams are often asked to be “adaptive, flexible, or agile” but also asked to stick with a plan. A traditional plan is based on scope, schedule and cost. An agile triangle turns the triangle upside down and has very different goals of customer delight, building quality products while speeding up the development process. The Agile Triangle is simpler works changed the way we view success.

### Essentials

Question should be: “Can we release this product now?”

* Look at the value of a functionality over implementing all the requirements
* Quality today – current iteration of release of product
* Quality tomorrow – release that continues to respond to business changes both anticipated and un-anticipated
* Constraints – scope, schedule, cost – not Unimportant, but not the goal of the project

### The Traditional Triangle

The traditional triangle plans out the scope of the project and estimates how much time and cost (both money and effort) the scope will require.

### The Agile Triangle

The agile triangle plans out the tome and cost that a project has and plans out a scope to meet those constraints.

## Cone of Uncertainty

The cone of Uncertainty describes the amount of uncertainty during different time periods of a project. At the beginning of a project, it’s hard to estimate activity with a high level of accuracy and confidence because little is known about the product or the results. As you begin researching and developing the product, the uncertainty level will decrease.

### Essentials

* Estimates at beginning of project are very vague
* Estimates and project plans need to be re-estimated on a regular basis
* Uncertainties should be built in the estimates
* Uncertainties should be visible in project plans

# Chapter 3 – Agile Process

## Agile Process

There are several popular models: eXtreme Programming(XP), Scrum, Crystal, Kanban, and others. All have a common solution for improving the outcomes of software development through concepts like “the minimum responsible amount” of many artifacts. Consider specific circumstances to figure out what might be the best fir for each situation. The LeanDog Agile Discussion Guide (and these notes) gives you an idea of the many tools and approaches that are available for addressing specific challenges that your team may encounter.

### Essentials

* Recognize that current processes, organizational structures, culture and approaches are usually the key causes of throughput/quality problems – not your people
* Coach your people on finding problems and challenges: finding, identifying and talking about problems is EXPECTED
* Based on the problems and challenges that are being highlighted, we can pick processes and approaches to help resolve them

## Kanban

In the 1950’s, Taiichi Ohno began using Kanban in Toyota’s primary machine shop. Translating to “signboard” or “billboard”, Kanban is a visual scheduling system that helps a team determine what to produce, when to produce it, and how much to produce

### Essentials

* Visualize workflow and picture the product in each state – from concept to deployment
* Limit WIP to prevent your team from becoming overwhelmed and keep progress continuous and steady
* Manage flow to allow and prepare for changes that will occur within the iteration
* Make team, queue and practice policies explicit so your team knows how to participate properly, ensuring a smoother iteration
* When a TEMPORARY surplus in capacity occurs, give precedence to helping other team members complete any work already in progress, before taking the new tasks/cards

### Advanced

* Implement feedback mechanisms like operations review, mentorship and daily production meetings to ensure demands are being met while identifying where improvements can be made
* Improve collaboratively with scientific approach: continuous reflective analysis paired with small adjustments, encouraging timely evolution at a sustainable pace
* Periodically review/revisit team WIP limits with team, customers and stakeholders, especially when changes to teams’ composition occur
* WIP limits are intended to facilitate the flow to work through a team by making it easy to decide whether or not the team is ready to take on new work. As such, WIP limits should never be used as negotiation tool – only as a collaboration tool

### Cycle Time

* Decrease WIP to decrease cycle time (delivery rate)
* Smaller stories in process will come out faster

## Scrum Framework

Scrum is an Agile development model that consists of small teams working interdependently. The teams work together but focus on their own tasks and must be capable of self-management and decision-making. Scrum is based around a “sprint”, which is generally a 1-4-week period for delivering a working part of the system. At the end of a sprint the results are delivered, then reviewed, and the next sprint is started.

### Essentials

* A product backlog is created and prioritized by the product owner
* The product owner selects story cards from the product backlog and adds them to the sprint backlog, deciding what cards get implemented
* A sprint (or iteration) is a short amount of time to get a significant amount of work done. Team selects the duration, LeanDog recommends 1-2 weeks
* The team meets each day to assess their progress; a “daily scrum”
* The Scrum Master keeps the team focused on their goals
* At the end of each sprint, the work that was selected from the product backlog should be released to the customer, put on a store shelf, or shown to a stake holder
* Every sprint ends with a sprint review and retrospective
* When a new spring begins, the team selects another group of troy cards from the product backlog
* The cycle repeats until there are no more story cards to be played in the product backlog, budget is unavailable, or the deadline has passed. Using the scrum proves ensures that the most valuable parts of the product are built first.

## Story Card Writing

A Story Card is the unit of each deliverable for an Agile team. Story cards include a sentence or two describing a needed function. Rather than representing detailed requirements, story cards are a “placeholder for a conversation”. Story cards are testable and include acceptance criteria. The details are elaborated upon via conversation between the customer, and the delivery team.

### Essentials

* Cards are hand written
* Cards should follow the format: “In order to (X), as a (Y), I want (Z);” these are the Values, Roles, and Goals of the card requested feature
* Spike cards exist (see Spike practice for definition)
* Follow INVEST rule (defined later)
* Story cards should represent a vertical slice of the application and deliver business value
* Story cards are sized to fit an iteration
* Use the 3 C’s: The Card, the Conversation AND the Confirmation

### Advanced

* Developer pairs with Product owner to write cards
* Utilize Story Mapping (later defined) as a tool to identify stories and priorities

### Invest Rule

* **Independent**
  + Story Cards should be independent of each other
* **Negotiable**
  + The card should be a short description of the deliverable without too many details. The details should be worked out during the “conversation” phase
* **Valuable**
  + Each story must be of value to the customer and have a value given to it
* **Estimable**
  + To plan and prioritize, developers need to be able to ballpark or estimate the effort to complete the story
* **Small**
  + Ideally, the story should be small, typically taking no more then 2-3 days
* **Testable**
  + If you cannot test it then you will never know when you are done, so a story needs to be testable for confirmation to take place

### Values, Roles, and Goals

As a \_\_\_(who)\_\_\_

I want \_\_(what)\_\_

So that \_\_(why)\_\_

Business Value: In order to \_\_\_\_\_\_\_\_\_\_

User Role/Persona: As a \_\_\_\_\_\_\_\_\_\_\_

Goals/Preform Something: I want to \_\_\_\_\_\_\_\_\_

### The 3 C’s

User stories have three essential aspects:

* **Card**
  + Stories are written on cards. Each card does not contain all of the information, but just enough to show the requirements of the feature. Cards are used during planning.
* **Conversation**
  + The requirements on each card are communicated from the customer to the delivery team members through conversation: exchange of thoughts, opinions, and feelings. These conversations happen many times before/during release planning, during iteration planning, and again when the story is ready for implementation
* **Confirmation**
  + After running acceptance tests and confirmation with the customer that the tests were successful, a card has been completed

## Estimation & Sizing

Estimation and Sizing are techniques for evaluating the size/complexity of delivering features to facilitate planning and guide investments decisions. A good planning process based on a reliable estimation and sizing approach reduces risk and uncertainty, supports better decision making, establishes trust and convey information.

### Essentials

* Use relative sizes, NOT HOURS
* Planning Poker – Story Points, T-Shirt sizes, etc.
* Team understands it’s not just time, it’s complexity
* Very few large cards
* Cards which the team feels are too large/complex within an iteration should be broken down into smaller/less complex cards
* Re-estimate when significant new information is uncovered, as it changed complexity
* Management understands differences of size/estimating vs hours
* Re-estimate in the event of team changes
* Make all cards as small as possible
* Consider all cards small “By Definition”
* Rewrite/breakdown/decompose card when smaller slice is found

### Advanced

* Measure and account for “Card Growth” between planning and delivery (e.g. 1 card in the backlog may turn into, on average, 1.25 cards delivered, due to some fraction of the cards being able to decompose into smaller slices when we discuss them in detail with the product owner

## Sprints/Iterations

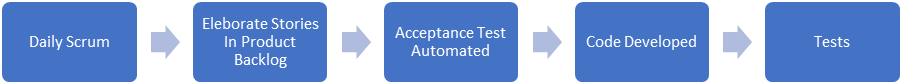
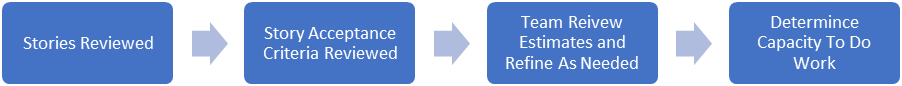
A Sprint/Iteration is a set time frame for measuring a development team’s throughput. Sprints/Iterations are one to two weeks in length with shorter iterations being the goal.

### Essentials

* Work is organized to be completed in short time frames (1 – 2 Weeks)
* Story cards worked on during an iteration must meet the team’s definition of done and get Product Owner acceptance to be considered complete
* Team gets credit for functional, testing and approved code

### Advanced

* Unplanned work is noted at retrospectives



**Iteration Start**

**Iteration Work**

**Iteration Close**

## Customer Collaboration

Customer Collaboration is the practice of including your customer throughout development. Only the customer can tell you what they want, and collaboration allows you to listen to their needs instead of guessing at what they need. The customer drives the requirements, prioritization, and review of work. While it is not always possible to have customers on site, you must make interaction with them a priority.

### Essentials

* Customer must be clearly identified, and roles and responsibilities clearly defined
* Customer may take the form of customer proxy, product owner, business stakeholder, end-user or any combination of these roles. Customer owns the responsibility of driving work in the team that meets overall product goals and objectives for software being built and delivered by focusing the team’s efforts on the highest priorities, highest business value features, and stories for release
* Customer works closely with team to break features down to stories
* Customer determines minimal marketable features and signs off on completed stories supported those features for release
* Grooming of product backlog is a collaborative effort between customer and team to adapt to changing product goals and objectives
* Customer listens to the team, the team listens to the customer
* The definition of “done” must be clear

### Advanced

* Create personas as a tool to better understand customers and their expectations
* Co-locate with the team to increase collaboration and communication
* Attend all team meetings (daily scrum, retrospectives, and sprint planning)

# Chapter 4 – Leadership

## Leadership Role

To support the successful adaptation of Agile and uphold its values, principles, and practices, an organization’s leadership needs to create a culture of continuous improvement. This section covers servant leadership, and important leadership style along with other collaboration techniques that allow teams to self-organize and make decisions. It is important that the entire organization, from the top down, understands the notion of servant leadership and agrees on a team-based decision-making process.

### Essentials

* Practice servant leadership
* Establish team-based decision processes
* Enjoy seeing team learn
* Remove roadblocks

### Advanced

* Leads by example by practicing Agile practices such as:
  + Daily Scrum
  + Information Radiators
  + Retrospectives
  + Making Work Visible
  + Liming WIP
  + Open Workspace

## Servant Leadership

A servant leader is an individual who puts others first. They are good communicators, collaborators, systemic thinkers, and lead with moral authority. A leader with moral authority is one who is worthy of respect, inspires trust and confidence, and establishes a quality standard for performance. They accept and delegate responsibility, share power and control, and create a culture of accountability.

### Essentials

Seven Pillars of Servant Leadership

1. **Person of Character**
   1. Maintain integrity, demonstrate humility and serve a higher purpose
2. **Puts People First**
   1. Displays a servant’s heart, is mentor-minded, and shows care and concern
3. **Skilled Communicator**
   1. Demonstrates empathy, invites feedback, communicates persuasively
4. **Compassionate Collaborator**
   1. Expresses appreciation, builds teams, and negotiates conflict
5. **Foresight**
   1. Visionary, displays creativity, and exercise sound judgement
6. **System Thinker**
   1. Comfortable with complexity, demonstrates adaptability, and considers the “Greater Good”
7. **Leads with Moral Authority**
   1. Grated by others

## Six Thinking Hats

Four people try to describe a house: one stands in front, one in back, one on each side. They can’t agree on the correct view. We’ve all been in that situation. The Six Thinking Hats® creates an environment that encourages “parallel thinking” (versus argument) where everyone in the discussion looks in the same direction at the same time.

1. **White Hat**
   1. Calls or information know or needed. “The facts, just the facts”
2. **Yellow Hat**
   1. Symbolizes optimism. Under this hat you explore the positives and probe for value and benefit
3. **Black Hat**
   1. Judgement. The devil’s advocate, or why something may not work. Spot the difficulties and dangers; where things might go wrong. Probably the most powerful and useful of the hats, but a problem if overused
4. **Red Hat**
   1. Signifies feelings, and intuition. When using this hat, you can express emotions and feelings, and share feats, likes, dislikes, loves and hates
5. **Green Hat**
   1. Focuses on creativity: the possibilities, alternatives, and new ideas. It’s an opportunity the express new concepts and new perceptions
6. **Blue Hat**
   1. Used to manage the thinking process. It’s the control mechanics that ensures the Six Thinking Hats® guidelines are observed

### The Four Major Benefits of the Six Thinking Hats®

1. **Power**
   1. The group is looking and working in the same direction
2. **Time Saving**
   1. Thinking in parallel prevents argument and saves time
3. **Removal of Ego**
   1. Minimizes emotion, creates focus
4. **One Thing at a Time**
   1. Minimizes confusion created by argument

## Collaboration 8

Collaboration 8 is a slight twist on Jurgen Appelo’s “7 Levels of Authority”. We use it as a fast and simple means of identifying who should be involved in the decisions, creating operating agreements and making personnel involvement publicly visible. The Joker card is a LeanDog addition. This card is played when someone sets expectations one way but does not continue to play by that agreement.

### The 7 Levels of Authority (Plus Joker)

* **Tell:** I know exactly what I want, and we need to do it this way
* **Sell:** I consider this to be my responsibility, but I will get buy-in from others
* **Consult:** I want your insight, but I will decide how we do the work
* **Agree:** I want to work together
* **Advise:** I want to give you insight and let you do the work
* **Inquire:** I trust you to do the work, but I want to understand so I can support you
* **Delegate:** I will support whatever decision you make
* **Joker:** You are violating the agreement that we made earlier

## Fist To 5

The Fist to Five approach combines the speed of thumbs up/down, with the degrees of agreement. Using this approach, people vote using their hands and fingers to represent their degree of support.

### Essentials

* **Fist** – A “no” vote, a way to block consensus. “I need to talk more on the proposal and require changes for it to pass”
* **1 Finger** – “I need to discuss certain issues and have some suggestions”
* **2 Fingers** - “I am more comfortable with the proposal but would like to discuss some minor issues.”
* **3 Fingers** – “I’m not in total agreement, but I feel comfortable to let this decision or proposal pass without further discussion.”
* **4 Fingers** – “I think it’s a good idea and will work for it”
* **5 Fingers** – “It’s a great idea and I will be one of the leaders implementing it”
* If anyone holds up fewer than three fingers, they should be given the opportunity to share their concerns with the team
* Teams continue the Fist-to-Five process until they achieve consensus (a minimum of three or higher) or determine they must move on to the next issue
* There must be a reasonable level of trust and cooperation amongst the group
* Ensure that everyone understands the voting system

# Chapter 5 – Scrum Master/Iteration Manager

## Scrum Master/Iteration Manager

This person is the servant leader of the team sometimes known as Scrum Master or Iteration Manager. A key part of their role is helping the team adhere to the Agile values and principles. Once the team has agreed to certain approaches to their work, it’s up to the Iteration Manager to remind the team of those commitments.

### Essentials

* Remove barriers between the development and the product owner, ensuring the development is directly driven by the product owner
* Assist the product owner how to maximize return on investment (ROI), and meet their objectives through Agile approaches
* Improve the lives and productivity of the development team by facilitating creativity and empowerment
* Team champion of the process
* Influences practices and Agile values and principles without command and control activity
* Internal team coach and protector of the team
* Does not promise or schedule work on behalf of the team
* Keep information about the team’s progress up to date and visible to all
* Identify, raise up and remove roadblocks

## Daily Scrum/Stand Up

The Daily Stand Up is an Agile ceremony where the team meets for no more than 15 minutes every morning. Each person on the team reports briefly on what they did yesterday, what they are going to do today, and any concerns or roadblocks they are facing. Daily Stand Ups reduce the need for team meetings. They encourage accountability, as team members are aware of all work going on. Stand Ups also allow for mid-course corrections, encourage the team to solve problems on their own and help notify managers early if there are any roadblocks.

### Essentials

* Collaborative activity with core team to share what they’re working on and any roadblocks that are preventing forward progress on a story
* Occur daily at the same time for 15 minutes or less
* Address what was done yesterday/ what will be done today/roadblocks/how you are doing
* Everybody attends and shows up on time
* Includes the customer so they can be informed and make adjustments
* There is no Stand-Up leader – team facilitates

### Advanced

* No stale roadblocks exist
* Debts help reinforce values at Stand Ups (i.e.: late to Stand Up/playing with toys/pass to someone who went)
* Happens daily, regardless of priorities
* Sound signal to Stand Up (Alarm/cow bell/gong)
* Team selects Stand Up time
* Team members records the roadblocks on the whiteboard, so they are visible and cannot be ignored
* Team members can “Pass” if they have no new information to share and no roadblocks

## Sprint/Iteration Planning

The Sprint/Iteration Planning meeting is held at the beginning of each new iteration to break down each of the features schedules into individual stories. At the start of a new iteration, available work hour and projected velocity are set, the team reviews any card not completed in the prior iteration, business intent for new cards is established, the team re-estimates cards if necessary and the team determines what cards fit the new iteration assigns them.

### Essentials

* Product Owner pulls stories from the product backlog to he played during the iteration
* Meeting includes the Iteration Close
  + Review of work completed during iteration, including roadblocks
  + Review of cards not completed during iteration
  + Velocity
* Meeting includes the Iteration Open
  + New story cards are reviewed for business intent/completeness
  + Team signs up for available work hours (Capacity Planning)
  + Target velocity is established
* Business partners understand velocity and work within the budget to select stories that will be played in the new iteration

## Show & Tell

Show and Tell is a demonstration of features completed during a sprint/iteration. This provides a forum for team members, customers and other stakeholders to review progress, provide feedback on new features, and prioritize any changes needed to delivered features. By using Show and tell, teams maintain accountability to the customer.

### Essentials

* Core team, business owner and supporting roles demonstrate working code and acceptance tests completed during the iteration
* Customers attend
* Customer approves that the implementation met expectations
* Whole team attends
* Customer prioritizes changes immediately

## Retrospectives

Retrospectives are meetings that take place after each iteration and are designed to help the team find ways to improve their process and output. This meeting should be a safe environment in which to talk about what worked and what didn’t work. The team votes on whether the items discussed during the respective should become cards to be played in the coming iteration.

### Essentials

* Core team applies continuous improvement principles to team activities; team discusses what went well and what didn’t go well during the iteration to determine what needs to be improved
* Goal is to stress continuous improvement. Experimental changes, even those that fail, are encouraged
* All team members attend retro and assess how things went during the previous iteration
* Format includes a version of “what worked” and “what didn’t work”
* Team decides which items to focus on’ output lists actions
* Progress on action items is tracked and published
* Conducted after every iteration

### Advanced

* Use safety checks to indicate that team members feel safe to bring up difficult topics
* Team holds spontaneous retrospectives for significant issues
* Team experiments with alternative formats (e.g. Lean Coffee, Six Thinking Hats, etc.)

## Velocity

Velocity is a simple, yet powerful method to consistently measure and track the rate a team delivers business value in each iteration. It is measured by counting the competed cards from the iteration that is ending. Then the velocity is used to determine and plan upcoming sprints. Iteration and estimate time of completion.

When a team and their environment are healthy, velocity should stabilize after 3-6 iterations to a predictable number range. Highly volatile velocity, 5 or 6 iterations, is an indication that the dynamics around or within the team has not stabilized and warrants further inspection.

### Essentials

* Must be measured
* If a card is not finished within a sprint/iteration, the cards and points are carried to the next sprint/iteration
* Never take partial credit, don’t split points across sprints/iterations; “not done” means zero points in the iteration even if only one hour of work is left
* The team’s previous velocity is used as the first guess of a future velocity
* Well understood by the whole team: “What was your velocity for the last sprint/iteration?”
* Managers look at burn up charts, not velocity
* Point method must be consistent within the team; does not have to be consistent across the teams
* Velocity adjusted based on holidays and large team absences

## Roadblocks

A roadblock is an obstacle that is standing in the way of the team. Management should be aware of roadblocks early, so they are quickly resolved, and the rest of the card can be completed in a timely manner.

### Essentials

* Displayed on a wall in each team area
* Roadblocks are prioritizing and completed
* Reviewed at end of each iteration/sprint
* When in progress, there should be an owner on the card
* Are declared during stand up

### Advanced

* Date on the card denotes roadblock’s inception
* Break roadblocks into columns of progress
* Create an escalation process for roadblocks in the event someone needs help
* Purpose of roadblock wall is clearly defined
* Senior management understand that roadblocks exists and where to find them

# Chapter 6 – Quality Assurance

## Quality Assurance Role

A Quality Assurance role is one where team members are responsible for inspecting and/or validating the product delivered both within the context of a story and as a whole

### Essentials

* Continually improve the tests with refactoring (including data)
* Work with the product owner and developer to establish a plan for providing business value
* Understand the business intent of the story card which help define tests to measure the effectiveness of the story implementation
* Collect and organize story tests into an automated test suite that becomes the regression test
  + Regression Tests are to help the team focus on the new functionality to the software while also maintaining stability
* Run and define exploratory tests and communicates results

### Advanced

* Undertake testing craftsmanship to help maintain development speed, quality, and efficiency
* Commit to be one of the “Three Amigos” / “Fantastic Four”., with excellent collaboration between product owner and developers
* Assist with defining features and specifications with Acceptance Test Driven Development (ATDD)
  + Testing that is molded by “Three Amigos” which involve, “What problem are we trying to solve?”, “how might we solve this problem?”, and “What about...”. These are the customer, developer, and tester.
* Automate ATDD Tests
* Pair with fellow team members to ensure someone else knows that section of the application, and hold each other to high levels of test quality

## Exploratory Testing

Exploratory Testing is a method of questioning and learning about the product as you design and execute tests, as opposed to following a predefined script. This is considered a mode of simultaneous learning, test design, and test execution that values the tester as a critical part of the test process

### Essentials

* Manual tests that occur during and after the story is developed
* Used to catch the edge cases that aren’t caught in automation
* Integral part of the quality strategy for a team or a project
* Involved the tester before the story card is implemented and throughout delivery of the story card

### Advanced

* Is never “done”
* Is usually executed against a “charter”
  + [How to Write A Charter](https://www.qualitestgroup.com/resources/knowledge-center/how-to-guide/write-exploratory-test-charter/)
* May be constrained to a time-boxed session between 30 and 120 minutes

## Automated Regression Testing

Automated Regression Testing is a testing process focused on detecting defects introduced into the software at the earliest possible opportunity while eliminating repetitive manual labor. Automated Regression Testing uses a comprehensive suite of automated tests to confirm the proper operation of each feature in the application and notifies the team if one or more features of the software is not operating correctly.

### Essentials

* An iterative activity to build up a regression test suite by creating tests prior to writing code
* Tests are executed every time code is checked into the code repository, resulting in immediate feedback on the health of overall code base
* Scripts are created for regression testing
* Automated tools run the scripts for regression testing
* Automated testing with duplicable test data generation
* Test entire application, both new and existing functionality, at least once per sprint/iteration
* To keep up with development efforts, regression testing must be automated.
* Most development efforts are on existing application. If you do not have great test coverage, where do you start?
  + With buggy or trouble areas
  + If no trouble areas, then start with critical application areas
  + Any new functionality

### Advanced

* See the Acceptance Test Driven Development Process for more guidance on improving the effectiveness and efficiency of regression testing

## Acceptance Test Driven Development

Acceptance Test Driven Development (ATDD) is a practice in which the team discusses the acceptance criteria, then creates a set of clear acceptance tests before development begins. It ensures that the team has a shared understanding of what is being developed. It requires a significant shift in how the team members interact with each other. The transition to ATDD can be a challenge, but the payback is significant

### Essentials

* Requirements are captured as combinations of free-form specification and Gherkin scenarios that describe how the system should behave when completed. These scenarios become the Acceptance Tests
  + [Gherkin Language](http://docs.behat.org/en/v2.5/guides/1.gherkin.html)
* Just prior to the development of a story, the product owner, tester and developer meet to refine and add any missing Acceptance Tests. We call this a “Three Amigos” meeting
* Automation of the Acceptance Tests must be completed prior to the completion of development
* Development focuses on making the Acceptance Tests pass. Together with the tester, their focus throughout this effort is to prevent defects
* Story is functionally complete once all Acceptance Tests pass
* Acceptance Tests run continuously to act as a regression tests
* The definition of “done” must be clear

### Advanced

* Tester automated Acceptance Tests at the same time as the developer writes code. They are in constant collaboration during this effort.

# Chapter 7 – Developer

## Developer Role

A developer is responsible for creating high-quality working code that meets the expectations of the product owner.

### Essentials

* Understands the business intent of the story card
* Works with the product owner and tester to come up with an approach on how to provide the value
* Save both tests and code in a repository
* Build code to satisfy the tests and conditions of the story
* Write and code with the purpose of clearly communicating the intent to the next developer that will look at the code, and be satisfied that you have coded is ready for production use
* Build tests that measure whether the implementation is working
* Continually improve the test and code base with refactoring
* Pair with a fellow developer to ensure someone else knows this section of the application, and hold each other to high levels of design and code quality

### Advanced

* Undertake software craftsmanship to help maintain development speed, quality, and efficiency
* Commit to be one of the “Three Amigos” / “Fantastic Four” and encourage collaboration between product owner and QA
* Build code to get acceptance tests to pass (ATDD – Acceptance Test Driven Development)

## Collective Code Ownership

Collective Code Ownership is a practice where all team members share responsibility for code quality. It also allows each developer to change any piece of code at any time. Collective Code Ownership also helps eliminate “specialization”, as everyone is expected to fix problems when they are discovered.

### Essentials

* All developers are responsible for the code; everyone can change the code
* Developers commit to writing outstanding new code from the start, instead of doing a half-hearted job and expecting someone else to fix the mistakes
* Check egos at the door
* Works best when coupled with Paired Programming and Test-Driven Development
* Eliminate knowledge silos by encouraging developers to work in unfamiliar areas of code

## Continuous Integration

Continuous Integration is the practice of ensuring all code changes are compatible with the existing shared codebase by assimilating each change into the code base as soon as possible after a developer submits the change. The entire system is tested in order to confirm the health of the code and very that the changed made are compatible with the shared codebase. Testing is preformed using an automated, scripted process known as “the build”

### Essentials

* The system gets built and all unit tests run at least once a day
* The entire build and test run takes no more than ten minutes
* Team is notified whenever a build fails
* Fixing a broken build is the team’s highest priority

### Advanced

* A system is build and all tests run every time a developer checks in code
* Visual indicators in the room show the build status

## Simple & Evolutionary Design

Simple & Evolutionary design is a system which minimized the mount of up-front design done before coding begins. Instead, the design of the system grows as it develops, emphasizing simple architectures and designs that provide adaptability to change. Simple designs also allow teams to have increased velocity as they spend more time providing customer value.

### Essentials

* “Simple is best” approach to software design where refactoring is an integral part of the code simplification process
* Believes in doing “the simplest thing that works’
* Avoid building unnecessary architecture
* Design is done just in time end evolved with the app
* Spikes are used to mitigate risk in design or investigate a new domain

## Paired Programming (Pairing)

Paired Programming is the technique of joining two team members at one computer. With two minds in constant collaboration, paired programming encourages sharing knowledge and catching bugs early in continual code review. The knowledge moves across the team more quickly and ensures that the best practices are followed.

### Essentials

* All code is produced by a pair of developers working together on one story at one workstation
* It’s encouraged for all roles to pair with anyone, anytime – switching every few hours is optimal
* Pairs change frequently, and very few knowledge silos exists
* No coding without a pair partner
* Don’t underestimate good hygiene
* Environment support pairing: desks are easy to access, office doesn’t have obstacles or obstructions
* Developers do not have their backs to the team, they sit beside or across from each other

### Advanced

* Frequent pair switching – switch pairs every 2-4 hours
* Remote Pairing
* Pairs use focusing techniques (e.g. Pomodoro) to stay on task

## Test Driven Development

Test Driven Development is a software development technique where developers write a test prior to any new application code and use short development cycles to design the product. There are three stages:

1. Writing a failing test
2. Write code to make it pass
3. Refactor and continue cycle until the developers cannot think of any more tests

Using TDD results in fewer defects, and provides a fast feedback look, allowing for fearless refactoring of the system.

### Essentials

* Tests are written prior to any code
* TDD workflow
  + Write a failing test
  + Write code to make the test pass
  + Refactor and continue cycle until the system fulfills all required functionality and satisfies all known use cases
  + Forces testable designs/architectures
  + Produces clean code
  + Tests to provide a safety net as application design evolves

## Technical Debt

Technical Debt is the gap between the right solution and the solution that currently exists. As technical debt accrues, the more difficult it becomes to reconcile the differences and deliver what is need. Technical debt can be uncured as a strategic initiative but needs to be repaid within only a few iterations.

### Essentials

* Clean code
* Good test coverage
* A learning objective
* Payback plan
* A truly informed product owner

### Advanced

* Technical debt is tracked / acknowledged / disclosed
* Technical debt is continuously minimized

## Spikes

A Spike is a time-boxed experimental story card that is included in the iteration cycle to determine an estimate but stops short of completing the story. Spikes are excellent way to try out something quickly, and are usually a few hours to a compel days in length

### Essentials

* Time-boxed
* Few in number
* A learning/discovery objective
* Code produced should be thrown away
* Results should be communicated back to the team and product owner along with demos
* Pulled forward in the release plan to reduce risk
* Played with enough leeway to adapt a plan
* Incorporate spikes into planning
* Estimate the spike like anything else

# Chapter 8 – Product Owner

## Product Owner Role

The product owner represents the stakeholders and acts as the voice of the customer. They are accountable for ensuring that the team delivers value to the business. The product owner writes customer-centric items (typically user stories), prioritizes them and adds them to the product backlog. Scrum teams should have one product owner, while they may also be a member of the development team, it is recommended that this role not to be combined with that of a scrum master.

### Essentials

* Decided what will be built and in which order
* Defines the features of the product or desired outcomes of the project
* Orders the product backlog to best achieve goals and missions
* Prioritizes features and outcomes according the market value
* Facilitates release planning ceremony with iteration manager/scrum master
* Adjusts features, outcomes, and priority as needed
* Accepts and rejects work results

## Value Stream Mapping

Value steam mapping is a type of flow chart used to design and analyze the flow of information needed to bring a product to the customer. Mapping allows you to discover waste, then construct a plan to estimate it.

### Essentials

* Flow of activities begins with the customer’s need and ends when the need is satisfied
* Identify the target product, product family or service
* Draw a current state value stream map illustrating current steps, delays and information flows required to deliver the target product
* Assess the current state value stream map in terms of creating flow and eliminating waste
* Draw a future state value steam map

## Demand Management

Demand Management helps organizations deliver the most value from their capacity by adopting continuous portfolio alignment, incorporating pull-based thinking, and limiting work in progress (WIP)

### Essentials

* Regular User Summits bring all users together in order by value to discuss their goals
* Project demand funnel is visible to team
* Cards are ordered by highest value
* Everyone understands concept of Minimal Marketable Feature (MMF) or Minimal Usable Feature (MUF)

### Advanced

* Customer personas are present and considered
* Story Mapping technique is used and understood by everyone

## Product Backlog

The Product Backlog is a high-level list of all potential features prioritized by business value for the customer.

### Essentials

* Just enough product backlog to create product unnecessary features
* Must be visible
* Uses concept of Minimal Marketable Feature Sets (MMFS)
* Must prioritize and organize backlog by business value
* Backlog is regularly reviewed and maintained
* Not a requirement list
* Team and Customer collaborate on prioritization, to ensure that technical complexity, duration, and cost of features are understood to the extend that they impact business value
* Customer has final decision authority around prioritization

## Release Planning

A Release Plan is an evolving roadmap that sets delivery goals for high-level feature sets. It will provide the team with an overview of the release and what is required to make the release a success. The plan should include key features, goals, responsibilities, and risks.

### Essentials

* Core team, business owner, and supporting roles drive out product features, details, and stories for release; anything not deemed part of the release will either be out of scope or moved to a future release
* Planning the next release is a joint effort between business partners and team
* Release plans map out several iterations to package releases and maximize business value
* Spikes are used to identify and mitigate risk, risk is pulled forward
* Produce burn up/burn down charts
* Layout cards in order of feature importance
* Team agrees to planning velocity which is used to create the Release Plan
* The team should review the plan and ask:
  + Is there enough work for all pairs?
  + Are any pairs stepping on others?
  + Are highest risk cards being played first?
  + Are the most valuable features coming out first?
  + Are there any dependencies which are missing?

### Advanced

* Release planning is a continuous activity for regularly schedules releases
* Suggest a rolling release planning window
* Suggest a sperate Release Planning Meeting with product owners to prioritize next set of features to work on for upcoming releases

# Chapter 9 – User Experience

## User Experience Designer

The User Experience (UX) Designer(s) are responsible for defining, designing, and testing the experiences and interactions that users of the system engage with. UX is a broad spectrum of skills that include information architecture, user research, data analysis, content strategy, visual design and usability. All of thee elements come together to craft a measurable user experience.

### Essentials

* Start with the problem and business objective
* Follow the 80/20 rule by focusing on the core functionality, or “happy path:
  + [80/20 Rule](https://medium.com/design-ibm/the-80-20-rule-in-user-experience-1695de32aaae)
* Stay one step ahead of the team, before development work begins
* Do just enough design
* Iterate your designs (refactoring is to be expected)
* Test and validate with users
* Communicate with your team
* Define success metrics

### Advanced

* Stay 1-2 sprints ahead of development
* Design spikes
* Cross-functional pairing
* Facilitate ideation sessions and encourage the team to participate
* Co-located to be with the team
* Define styles and patterns for common elements early in the process
* Visualize the workflow or interactions
* UX design reviews

## Personas

Personas are profiles of the customers who will be using your product. They are an important tool that helps reduce waste by ensuring all features are necessary. Personas typically include: behavior and usage patterns, goals and motives, knowledge or skills, and sometimes the person’s role and demographic. Personas are not made-up but researched thoroughly. The goal is to define the target audience you are designing the system for.

### Essentials

* Customer base is depicted visually
* Personas are based on a subset of the customer profile
* Understand your customer clearly
* Provide an instance of user types
* Take scenarios from personas
* Avoid mirror personals or elastic users
* Design each part of the interface for just one persona
* Should “wiggle” under the pressure of development
* No more than three primary personals for the entire system
* Describe current situation of customer profile, describe goals, needs of the profile, and how will application be created add value to person

Home Buyer

Goals

* Move into a bigger house
* Better life for her kids than she had

Situation

* 34 years old
* Current home not large enough
* Drives kids to school each morning

Value

* Our web app will show here the availability of homes in the area
* App allows user to access info about available home

## Story Mapping

Story mapping is the highest level in the ideation process and should be completed first. It includes every feature you want in your product. A story is categorized by size, then grouped together by features that are linked or related.

### Essentials

* Early stage of progressive elaboration
* Team uses note cards to depict the features and their flow in the system
* Start at the Epic level and continue down into user stories, then into features
* Features are organized by value order
* Discovery and Prioritization
* Used early on, not an ongoing practice
* Technique can be applied to reverse engineer an application
* A good tool for getting people to think through what they are asking for

## Low-Fidelity Prototyping

Low-Fidelity Prototyping is a tool used to create a mock-up that is quick and incomplete but has the characteristics of the target product. It’s simple and required minimal effort to quickly produce the prototype and test broad concepts.

### Essentials

* Initial version of user interface is created manually
* Rapid iterations and prototypes with team members
* Build the software and encourage interaction

### Advanced

* Creation of story boards that represent a larger portion of the project
* Lightweight, no tools, fast