

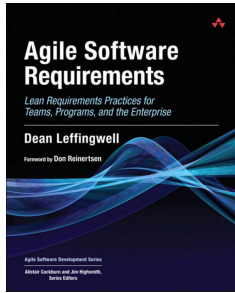


CE Department

# Software Requirements Engineering

40688

These slides are designed to accompany Agile Software Requirements (2011) by Dean Leffingwell and support the university course Software Requirements Engineering, instructed by Mehran Rivadeh. Created and designed by Mahnaz Rasekhi.



## Agile Software Requirements (2011)

Dean Leffingwell

# *Iterating, Backlog, Throughput, And Kanban*

## *Chapter 9*

**Mehran Rivadeh**  
mrivadeh@sharif.edu  
Software Requirements Engineering  
October 2025 - Fall 1404 - SUT

## Contents

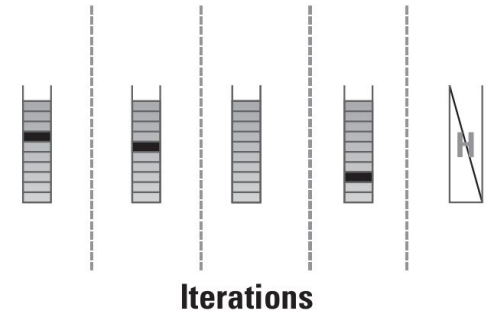
1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

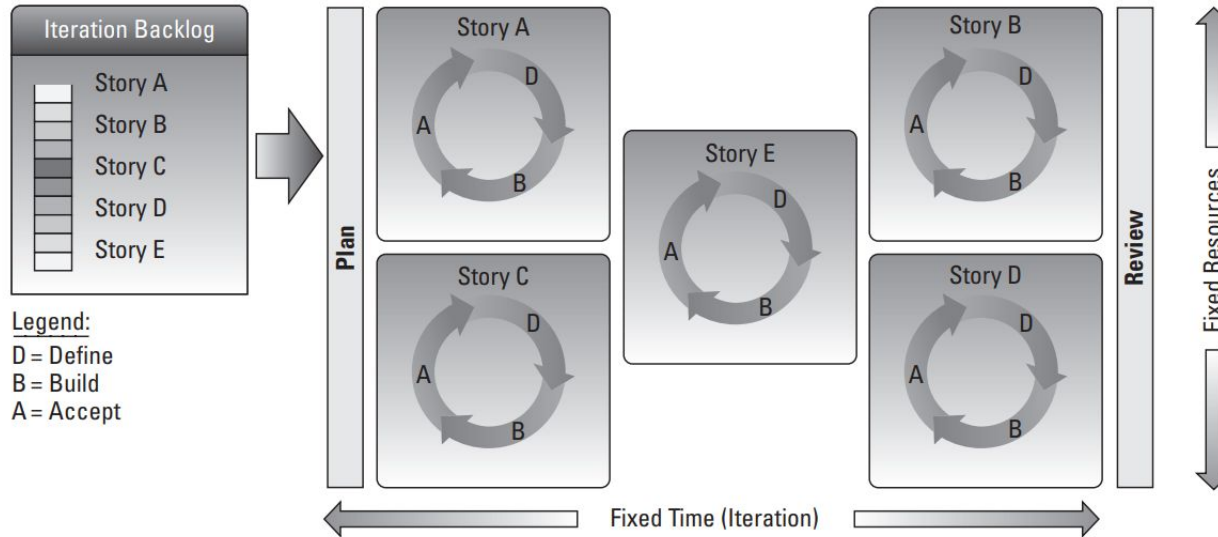
# Iterating: The Heartbeat Of Agility

---

- The **basic unit** of agile development is the **iteration**:
  - ◆ The ability to take a set of user stories from the backlog and refine, code, test, and accept those stories into a new integrated baseline within a fixed timebox.
- The **goal** of each **iteration** is the same:
  - ◆ To **build** an **increment** of potentially shippable code that is of value to the users.



# Iterating: The Heartbeat Of Agility



**Figure 9–1** The basic iteration framework

## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

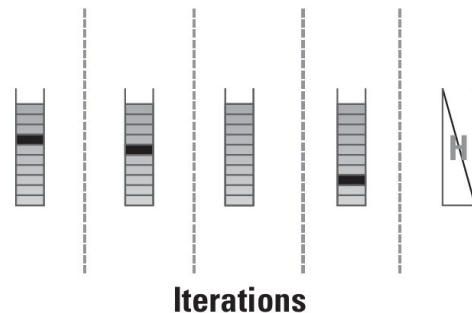
1. **Iterating: The Heartbeat Of Agility**
  - a. **Iteration Length**
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Iteration Length

---

## What Is The Optimal Iteration Length?

- Most agree that iterations are a fixed, constant length, but the length of the iteration in the literature is a variable.
- In practice, most teams have come to a common conclusion:
  - ◆ “A week is too short and 30 days is too long”
  - ◆ So they standardize on **two weeks**.

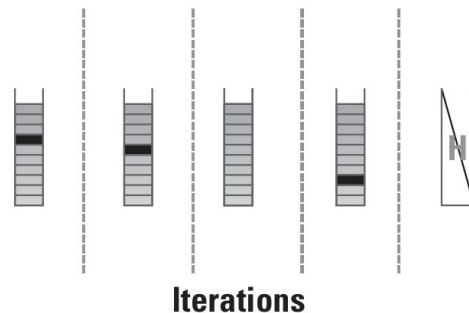


# Iteration Length

---

## Advantages of A Two-Week Iteration

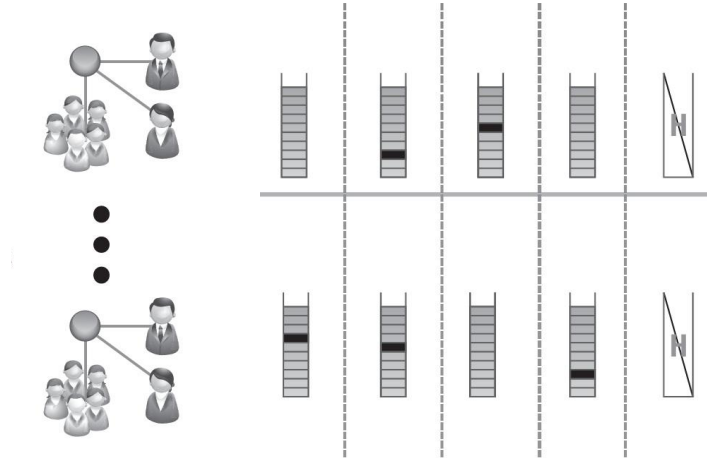
- It is enough time to create **meaningful incremental value**.
- It is the **fastest feedback** they feel they can afford given the transaction costs of planning and other overhead.
- Short iterations force **breaking stories into bite-size chunks**, and define/build/test has to be concurrent, avoiding the tendency to “waterfall” the iteration.



# Iteration Length

## A Tip

- To facilitate coordination with other teams and larger releases, it is recommend that **all teams** on a project **apply same iteration timebox**.



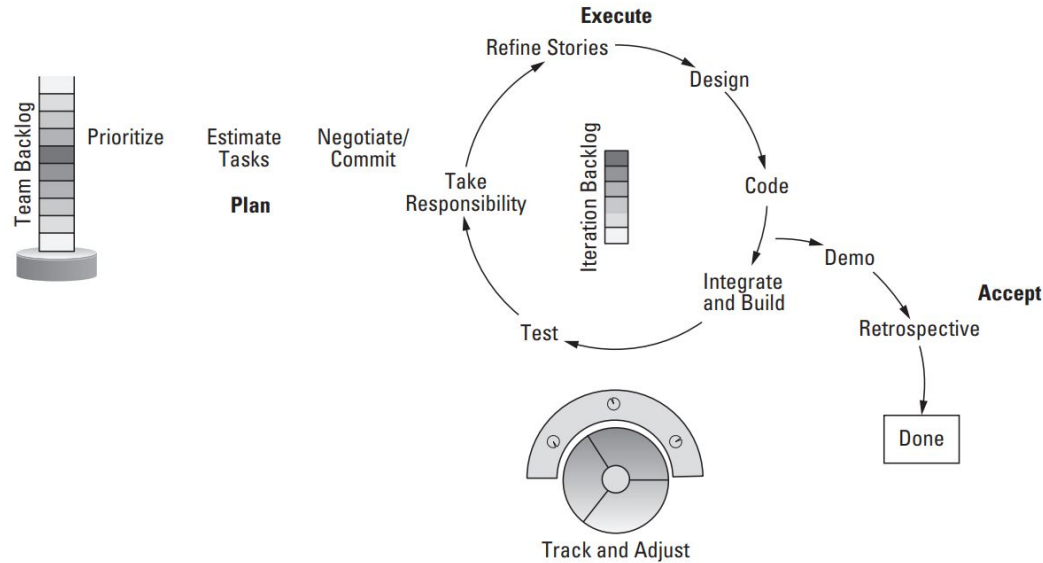
## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern**
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Iteration Pattern

All iterations have the same pattern: plan, execute, review, and retrospective.

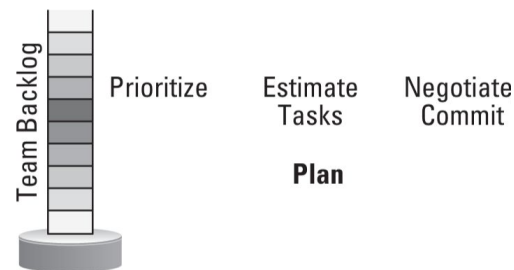


# Iteration Pattern

---

## Plan

- It is a short planning session, **two to four hours** during which
  - ◆ The backlog is reviewed and prioritized.
  - ◆ Estimates are established.
  - ◆ The team commits to an amount of work for the upcoming iteration.

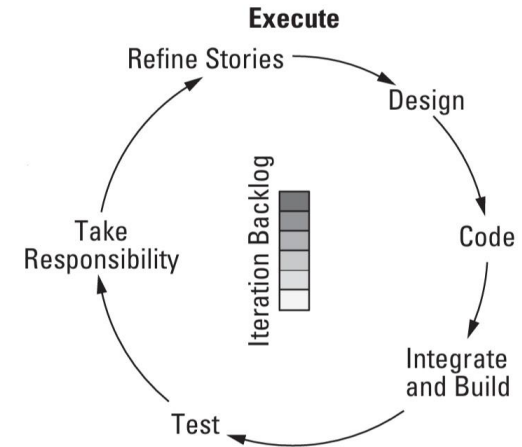


# Iteration Pattern

---

## Execute

- It is when the iteration backlog items are implemented in code and tests.

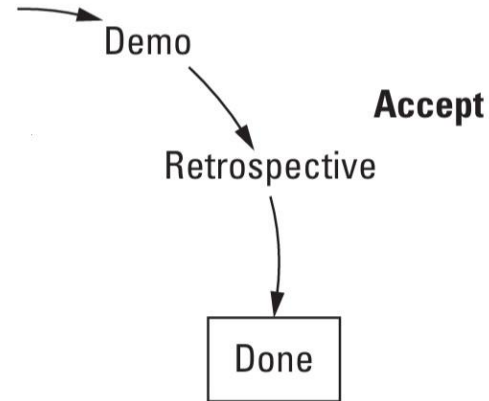


# Iteration Pattern

---

## Review And Retrospective

- The final phase involves review and evaluation of the new system increment followed by a retrospective on the iteration process and results.



## Contents

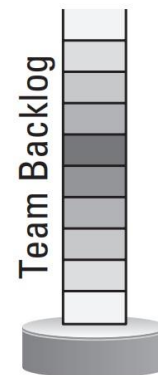
1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog**
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Team Backlog

---

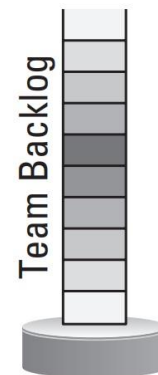
- The team's backlog serves as a **“to-do” list for the team.**
  - ◆ It contains the identified new stories that must be done in order to release the product.
- It contains a **list of reminders of what must be done** in order to **complete the project.**
  - ◆ Unlike a traditional product or software requirements specification, the backlog is not designed to contain all the elaborated requirements for the solution.



# Team Backlog

---

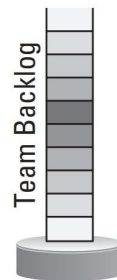
- Prior to implementing a backlog item, more detailed requirements will likely be required.
  - ◆ They may be detailed by **elaborating the item itself**.
  - ◆ They may be detailed by **providing attachments** referenced by the backlog.
  - ◆ They may be developed as **more detailed acceptance criteria** for the backlog item.



# Team Backlog

---

- Backlog items can take on many forms, but most are represented as **user stories**.
- The backlog may contain **items of any size such as features** and/or **epics**.
- Backlog items should be **focused on user functionality**.
  - ◆ More detailed requirements and acceptance criteria are typically not elaborated until just prior to, or within, the iterations in which they are implemented.
- The backlog may also contain other to-do items, including **defects, infrastructure work**, and the like.



## Contents

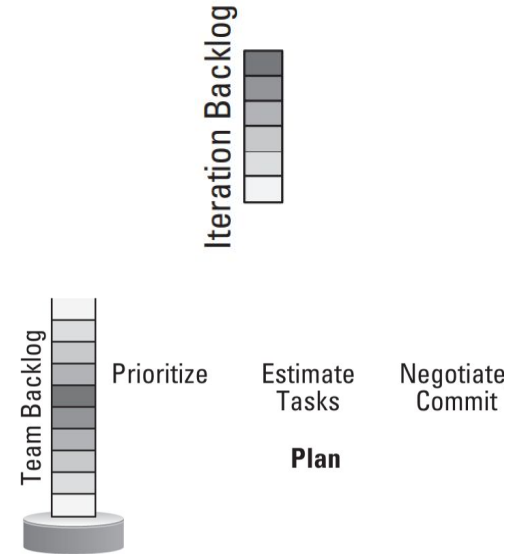
1. Iterating: The Heartbeat Of Agility
2. Backlog, Lean, And Throughput

1. Iterating: The Heartbeat Of Agility
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration**
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Planning The Iteration

The team begins the iteration with a **planning session** during which:

- It reviews the backlog, **selects** and **reviews** the **stories** for the **current iteration**.
- **Defines** and **estimates the tasks** necessary to deliver the stories.
- When complete, the **team makes a commitment** to deliver a certain number of user stories and other backlog items to the baseline; that set of things is the **team's iteration backlog**.



# Planning The Iteration

---

## Refining Backlog Stories

- For **mature teams**, stories will likely have been **elaborated, estimated, and prioritized prior** to the **planning meeting**.
- However, the items may be a few weeks old, or even older, so it is likely that the **stories must be reconsidered in light of the current project state and current project priorities**, so some reestimating and reprioritization may occur.
- In addition, **stories will be split as necessary**, and new stories may be discovered.

# Planning The Iteration

---

## Preparation For The Planning Meeting

- The planning meeting is **short** and **timeboxed**. Everyone who will be involved in the iteration should attend. Required attendees typically include:
- ◆ Agile/Scrum Master
  - ◆ Product owner
  - ◆ Developers
  - ◆ Test, QA
  - ◆ Documentation personnel
  - ◆ Maybe a tech lead or architect
  - ◆ Other project stakeholders may also attend, but the meeting is for the team.



# Planning The Iteration

---

## Preparation For The Planning Meeting → **Product Owner Responsibilities:**

- Review the release plan to make sure the Vision and goals are still appropriate.
- Review and reprioritize items in the backlog, including stories that
  - ◆ were already there
  - ◆ failed acceptance in a prior iteration
  - ◆ are newly generated from defects or other stories.
- Understand how the reprioritization may affect other teams who are dependent on commitments made during release planning.
- Understand the customer needs and the business value that each story is to deliver.
- Be prepared to further elaborate the story.

# Planning The Iteration

---

## Preparation For The Planning Meeting → Development Team Responsibilities:

- Review the top-priority items in the backlog and prepare any questions.
- Consider technical issues, constraints, and dependencies, and be prepared to share these concerns.
- Think about the work involved in delivering the functionality in the stories in order to be better prepared to make estimates in the meeting.
- Understand what the team's capacity should be for the upcoming iteration, based on team discussions at the last review.

# Planning The Iteration

---

- The **objective of planning** is to **define** and **accept** a **reasonable scope** for the **iteration**.
- The **product owner** and the **development team** may **add** or **reduce** **stories**, **defects**, and other infrastructure work on the basis of the **current project context**.
- **Stories** may **be split**, **combined**, **estimated**, and **re estimated** as necessary.
- The **product owner** then **reranks** the **work items**, and the **team** **selects** an **initial scope of work** based on the **estimates** and the **team's velocity**.

# Planning The Iteration

---

- The **product owner** may **adjust certain backlog items** in ways that make them **less costly to develop** or trade out entire backlog items for others.
- **At the end** of the iteration planning meeting, the product owners and development team jointly **commit to the iteration plan**.
- After that, **the scope of the work must remain fixed**; otherwise, the commitment cannot remain meaningful.

## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment**
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Iteration Commitment

---

## Velocity-Based Commitment

- Some teams **simply** pull **stories** into the **iteration backlog** in **priority order**.
- Then **stop** when the **estimate reaches** their **target velocity**.
- Teams know only that “**it is about the right amount of work**”.
- The work is **not further analyzed** with respect to **available resources, task breakouts, and assignments**.

# Iteration Commitment

---

## Velocity-Based Commitment (Cont.)

- In this model, teams are counting on their **experience, constant communication, tag-teaming stories,** and, where necessary, “**covering each other’s backsides,**” for stories that may overload some individuals.
- The **commitment**, then, is just the **set of stories** in the **iteration backlog**.
- Many teams assign a **chief engineer** to **each story**, whose job is to wrangle the story to completion, independent of their particular role in that particular story.
- In this way, every story has an owner whose job is to understand status, eliminate impediments, and, to generally, just help get the job done for that story.

# Iteration Commitment

---

## Objective-Based Commitment

- Some teams who have been working together for some time make their commitment based on the broader objectives negotiated with the product owner.
- In this approach, tasks are just steps toward the goal—the goal itself is what really matters.
- Thereafter, some teams use a pull/kanban style approach, whereby they **pull items from the backlog** and **complete them serially**.
- Few stories, if any, are committed up front.
- The team simply starts work, pulls stories from the backlog, creates new stories where necessary, communicates vociferously, and attempts to meet the objectives of the iteration “on the fly.”

# Iteration Commitment

## Objective-Based Commitment

### Example

#### Objective1 – Update and Support Beta 1.1.41

- Finalize and push last name search
- Finalize and push first name morphology
- Add request an invitation (see Rally)
- GUI/login nits and nats (see defect list)
- Augmented user tracking

This iteration has two objectives: one to support the beta product, the second for new development.

#### Objective 2 – Version 1.2 Index 75% of Web Data

- Get balance of 80% data to Canada
- Start new code development on EC2 Objective?
- Full text indexing (part 3 of 4). Objective:?
- Develop Quofile Storyboard for 0.9

These are not the actual stories, just the PO view of objectives. Stories are either in the backlog or will be created by the teams during planning.

#### Other Stories

- US 51 Establish search replication validation protocol
- RE 102 Refactor artifact dictionary schema
- US 53 Affiliates II (depending on what we discover tomorrow)
- SP 67: Search layer optimization/testing continues

In addition to the two objectives above, these stories are also targeted for completion, in priority order.

# Iteration Commitment

---

## Task-Based Commitment

- Many **XP** and **Scrum** teams are rigorous in their use of task-based commitments.
- In this case, **each story is broken into tasks** that individual team members take responsibility for.
- **Tasks have an owner** (the person who is going to do the task) and are **estimated in hours** (not points).



# Iteration Commitment

---

## Task-Based Commitment

Example:

Story 51: Select photo for upload

Task 51.1: Define acceptance test → Juha, Don, Bill

Task 51.2: Code story → Juha

Task 51.3: Code acceptance test → Bill

Task 51.4: Get it to pass → Juha and Bill

Task 51.5: Document in user help → Cindy

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

- Bill Wake (the inventor of INVEST for user stories) uses the acronym **SMART** to describe tasks.
  - ◆ **Specific**
  - ◆ **Measurable**
  - ◆ **Achievable**
  - ◆ **Relevant**
  - ◆ **Timeboxed**

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

#### → Specific

- ◆ A task needs to be specific enough that everyone can understand what's involved in it.
- ◆ This helps keep other tasks from overlapping and helps people understand whether the tasks add up to the full story.

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

#### → Measurable

- ◆ Can we mark it as done?
- ◆ Does what it is intended to?
- ◆ Tests are included?
- ◆ The code has been refactored?

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

#### → Achievable

- ◆ The task owner should expect to be able to achieve a task.
- ◆ XP teams have a rule that anybody can ask for help whenever they need it; this certainly includes ensuring that task owners are up to the job.

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

#### → Relevant

- ◆ Every task should be relevant, contributing to the story at hand.
- ◆ Stories are broken into tasks for the benefit of developers, but a customer should still be able to expect that every task can be explained and justified.

# Iteration Commitment

---

## Task-Based Commitment

### Tasks

#### → Timeboxed

- ◆ A task should be timeboxed (limited to a specific duration).
- ◆ This doesn't need to be a formal estimate in hours or days, but there should be an expectation so people know when they should seek help.
- ◆ If a task is harder than expected, the team needs to know it must split the task, change players, or do something to help the task get done.

# Iteration Commitment

---

## Task-Based Commitment

- In this task-based approach to commitment, individuals are accountable to complete their tasks, communicate with others when their task is complete or impeded, and then move on to the next task without additional supervision.
- It is highly granular and highly accountable.
- If the tasks are all completed, the stories will be accepted, and the goals of the iteration will be met.

# Iteration Commitment

---

## Result: The Iteration Plan

- No matter the approach to the commitment, the result of the planning meeting is an iteration plan that contains a number of key elements:
  - ◆ An **objective**: a statement of what the iteration is intended to accomplish.
  - ◆ A **prioritized list of stories** to work on for the iteration.
  - ◆ The stories' **estimated tasks** and **owners**.
  - ◆ A **commitment** by the team to the objectives of the iteration.
  - ◆ **Documentation** of the plan in a visible place or in a widely accessible tool.

## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration**
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review

# Executing The Iteration

---

- Having committed to the iteration plan, the team is faced with the question of how to allocate and adjust work for the members of the development team.
- The preferred approach is that developers simply pick the work they would like to do.
- When things are happening quickly, there is not enough time for information to travel up the chain of command and then come back down as directives.
- Taking responsibility for work must be supported by a **visible indicator** showing who is responsible for what work and by the daily status meetings during which status and issues can be discussed.

# Executing The Iteration

---

**Each developer** (or perhaps developer pair or developer/tester pair) will follow the same basic process repeatedly **throughout** the **iteration**:

1. Take responsibility for an assigned backlog item (for example, user story, defect fix, other).
2. Develop (refine, design, code, integrate, and test) the backlog item.
3. Deliver the backlog item by integrating it into a system build.
4. Declare the backlog item as developed, signaling that it is ready for acceptance testing.
5. Get the backlog item accepted by the product owner.

This cycle repeats as someone ultimately takes responsibility for all the backlog items in the queue.

# Executing The Iteration

---

- In most organizations, developers also support management of the process by estimating effort expended so far and remaining effort for the backlog items that they are responsible for.
- This creates a **burndown chart** that the teams use to track the **overall status** of the **iteration**.
- As described earlier, because the story is pliable, these activities happen in parallel, and the objective is to deliver a working story (as it evolves) into the baseline.
  - ◆ Definition affects design, design affects test, test affects design, and so on.

## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment**
  - h. Review And Retrospective
  - i. Feature Review

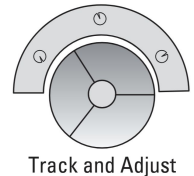
# Tracking And Adjustment

---

- Even within the course of a short iteration, scope must be managed, and deviations from the plan will occur, so tracking status and adjusting course is necessary.

## Tracking Progress with the Big Visible Information Radiator

- Tracking progress requires **having visibility into the status of the stories, defects, and other tasks** that are being worked on during the iteration.
- Most teams use a big visible information radiator (BVIR, or sprint status board) on a wall in the team room for this purpose.



# Tracking And Adjustment

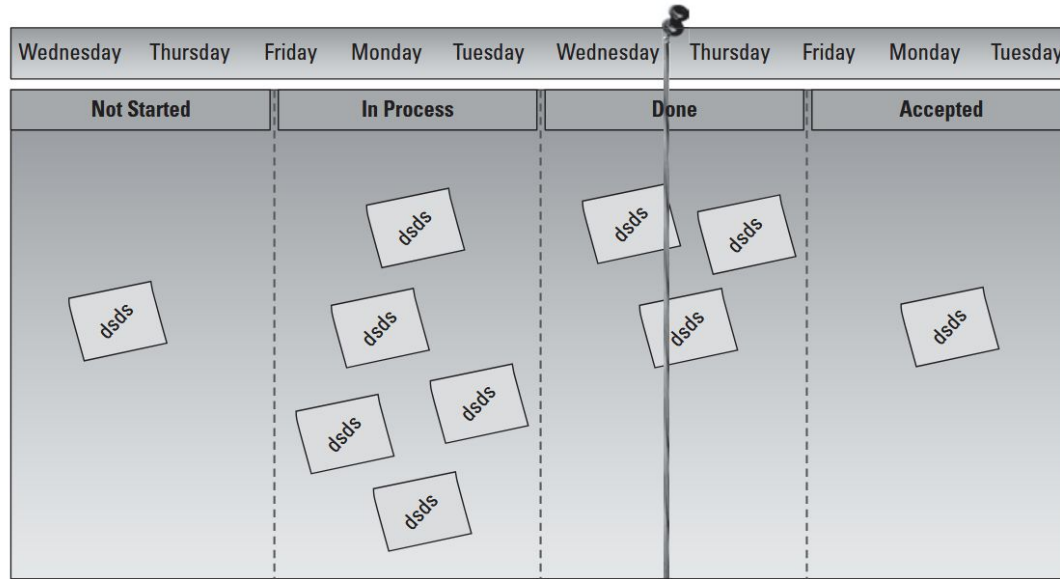
---

## Tracking Progress with the Big Visible Information Radiator (Cont.)

- In the deceptively simple radiator shown in Figure 9–4, the team simply moves the ribbon to the current day before each daily stand-up.
- This gives the entire team an instant assessment of **where they are in the iteration** (note how you can tell that the iteration in Figure 9–4 is at risk) and, more importantly, **what they need to do to complete it successfully**.
- If a manager or remote participant needs status information, all the team needs to do is to snap a picture and send it off.

# Tracking And Adjustment

## Tracking Progress with the Big Visible Information Radiator (Cont.)



**Figure 9-4** Example of an iteration BVIR

# Tracking And Adjustment

---

## Tracking in Daily Stand-Ups

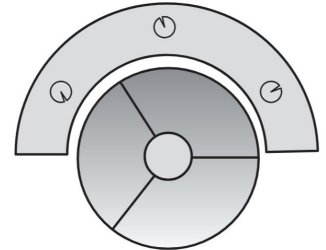
- One of the key rhythms of agile development is the practice of **daily**,
  - ◆ **15-minute stand-up** meetings, an event that **all team members** attend.
- The stand-up's purpose is to share information about the progress, to communicate, and to coordinate activities daily.
- Ideally, the team performs the daily stand-up in front of the BVIR.
- A typical round-robin format is as follows:
  - ◆ What **stories I worked** on **yesterday** and their status.
  - ◆ What **stories I will** be able to **complete today**.
  - ◆ What is getting in my way (**am I blocked?**).

# Tracking And Adjustment

---

## Tracking with Agile Project Management Tooling

- For larger and distributed teams, status is usually tracked using an automated agile project management.
- With a tool like this, any stakeholder can see the current status of the iteration including the state of each story.



Track and Adjust

# Tracking And Adjustment

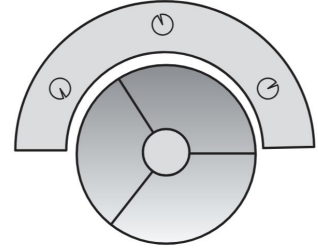
---

## Tracking with Agile Project Management Tooling

→ An example of such a state model is as follows:

- ◆ Backlog
- ◆ Defined
- ◆ In-Progress
- ◆ Completed
- ◆ Accepted
- ◆ Blocked

→ The remaining task-based work estimates (to-do hours remaining), and the overall burndown for the iteration.



Track and Adjust

# Tracking And Adjustment

---

## Tracking with Agile Project Management Tooling

- Many teams use both.
- The agile project management tool is the information master for stories and story state and also contains attachments and acceptance criteria that further define the detailed requirements for the story.
- This creates the “one central version of the truth” that is so valuable when multiple teams must cooperate on a project.
- Even then, however, many teams use the BVIR for visual status, but the story card on the wall is then just a token representing the real object in the tool or repository.

# Tracking And Adjustment

---

## Completing The Iteration

- This part is simple.
- When the timebox is over, the iteration is done, no matter the actual status of the stories in the iteration.
- From there, the team moves into the review and retrospective.
- These are conducted on time, and as scheduled, no matter the actual accomplishments of the team in the timebox.

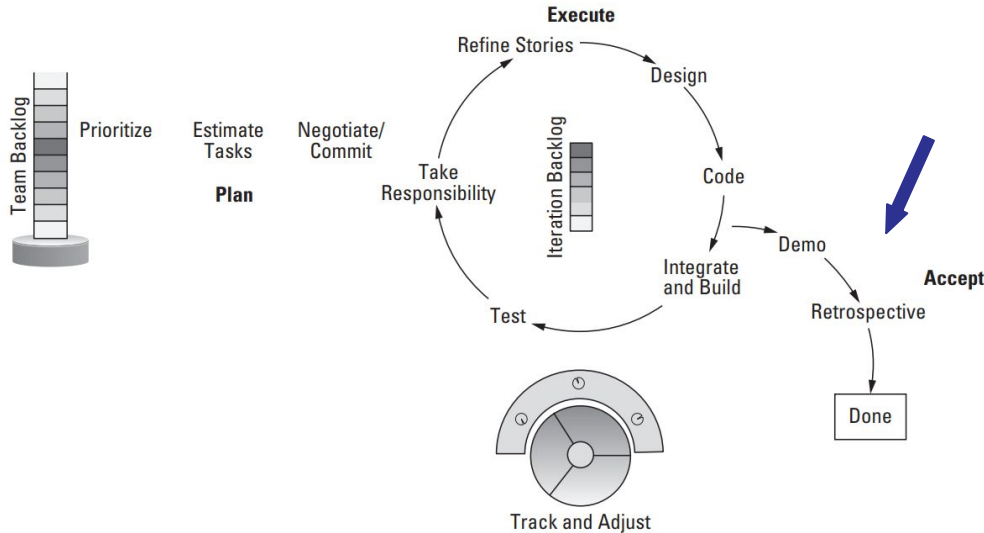
## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective**
  - i. Feature Review

# Review And Retrospective

The final activity has two parts, the **product demonstration** and **review** and the **iteration retrospective**.



# Review And Retrospective

---

## Review And Demonstration

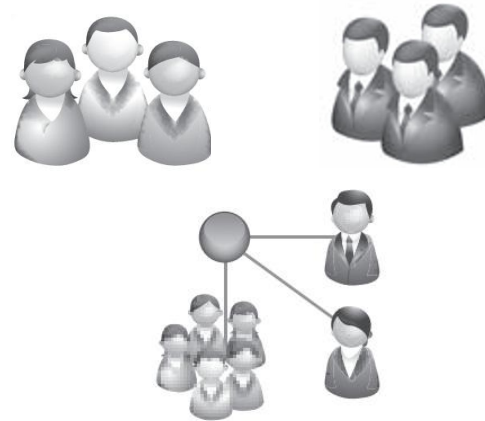
- The reality of software development is that the **customer's understanding of the requirements** for a system **evolves** as they see and use the software.
- Every **iteration** is an **opportunity** for the team to **get feedback** and **guidance** from the **customer** or **product owner** (customer proxy) about how to **make** the **system more valuable**.
- This **feedback** is typically structured as a somewhat **formal, one-hour demonstration** of **new functionality**.

# Review And Retrospective

---

## Review And Demonstration

- Attendees include the key stakeholders, such as
- ◆ Product managers
  - ◆ Business owners
  - ◆ Executive sponsors
  - ◆ Other teams
  - ◆ Customers
  - ◆ and, of course, the product owner and team.

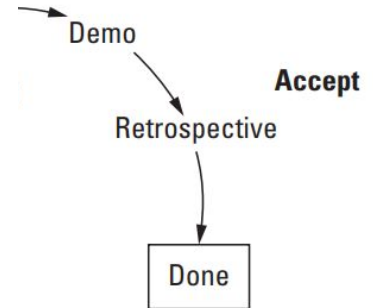


# Review And Retrospective

---

The format for this meeting is as follows:

- Demonstration of each story that was committed to be delivered at the iteration planning by the responsible party.
- Discussion and feedback of each with stakeholders.
- Afterward, the teams review the results of the iteration from two perspectives:
  - ◆ Did we accomplish the objectives for the iteration?
  - ◆ If so, or if not, how might that impact the upcoming release?



# Review And Retrospective

---

## Iteration Retrospective

- The goal of the retrospective assessment is to **mine the lessons learned during the iteration** and then **adapt the development process accordingly**.
- The assessment allows the team to **continually improve** the throughput of the **development process** and the **quality** of the **resulting system**.
- Typically, the team conducts the retrospective in two parts.

# Review And Retrospective

---

## Iteration Retrospective

### → Quantitative (metrics) review:

- ◆ The team assesses whether the team met the objectives of the sprint (yes or no).
- ◆ They also collect any metrics they have agreed to analyze, which must include velocity—both the portion that is available for new development and the portion devoted to maintenance.

# Review And Retrospective

---

## Iteration Retrospective

### → Subjective (Process) Review:

- ◆ In the subjective review, the team analyzes its own process, with a focus on finding one or two things they can do better in the next iteration.

| What Went Well? | What Didn't? | What (One Thing) Can We Do Better Next Time? |
|-----------------|--------------|--|
|                 |              |  |

# Review And Retrospective

---

## Iteration Calendar

- As part of establishing a rhythm of agile development timeboxes, meetings, and checkpoints, teams typically establish a standard “**cadence calendar**” at the start of each program.
- The cadence calendar helps the team set its schedules so that members can set aside time for planning meetings, daily stand-ups, demos, reviews, and retrospectives.
- Thereafter, no meeting notices are required, because the schedule for each day is fixed.

# Review And Retrospective

## Iteration Calendar

Sample

|       | Day 1   |       | Day 2–9                                 |       | Day 10                  |
|-------|---|-------|---|-------|-------------------------|
| 8:00  |   | 8:00  |   | 8:00  |                         |
| 9:00  | Iteration Planning (PO Presents Stories)                        | 9:00  | Daily Stand-up (15 min)                 | 9:00  | Daily Stand-up (15 min) |
| 10:00 | Planning: Team Breakouts, Elaborate, Task, and Estimate Stories | 10:00 |   | 10:00 |                         |
|       |   | 11:00 |   | 11:00 |                         |
|       |   | 12:00 |   | 12:00 |                         |
|       |   | 1:00  |   | 1:00  |                         |
|       |   | 2:00  |   | 2:00  | Demo                    |
|       |   | 3:00  |   | 3:00  | Retrospective           |
| 4:00  | Reconvene, Negotiate Scope, Commit                              | 4:00  | (Mid-Iteration Only)<br>Feature Preview | 4:00  |                         |

## Contents

1. **Iterating: The Heartbeat Of Agility**
2. Backlog, Lean, And Throughput

1. **Iterating: The Heartbeat Of Agility**
  - a. Iteration Length
  - b. Iteration Pattern
  - c. Team Backlog
  - d. Planning The Iteration
  - e. Iteration Commitment
  - f. Executing The Iteration
  - g. Tracking And Adjustment
  - h. Review And Retrospective
  - i. Feature Review**

# Feature Preview

---

- One challenging aspect of agile is what we call the “tyranny of the urgent iteration.”
- The team’s commitment to the current iteration causes an intensity of focus that is unparalleled in prior development models.
- That is mostly good, because focus increases productivity and quality, in part by eliminating task switching and overhead.

# Feature Preview

---

- Moreover, working in a highly productive team, coupled with peer pressure, drives the team to new levels of performance.
- During this process, the teams tend to operate in a “heads-down” mode, and there is a danger that they could make tactical, near-term decisions in the code that could complicate future stories.
- So, how to address this problem?

# Feature Preview

---

- One way to address this is with a weekly (or every other week) meeting where the product owner discusses upcoming features or user stories.
  - ◆ This meeting is sometimes called **story time** or **backlog grooming**.
- The meeting can include brainstorming new ideas, having short implementation discussions, and refining and estimating backlog items.
- This scheduled meeting gives the team a “timeout” from the current iteration and time to think a bit about the future.
- It also gives the product owner a way to meet with the team and discuss upcoming stories and perhaps estimate some features that a new customer is requesting, without interfering with the team’s cadence or velocity.

## Contents

1. Iterating: The Heartbeat Of Agility
2. **Backlog, Lean, And Throughput**

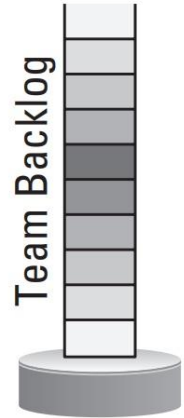
## 2. **Backlog, Lean, And Throughput**

- a. Is That Well-formed Product Backlog Decreasing Your Team's Agility?
- b. Little's Law And An Agile Team's Backlog
- c. Applying Little's law to Increase Agility And Decrease Time To Market
- d. Managing Throughput By Controlling Backlog Queue Length

# Backlog, Lean, And Throughput

---

- The invention of the simple backlog construct in agile can radically improve the performance of a software team.
- Indeed, it's hard to overstate the importance of this construct in organizing and unifying the mission of the team.



# Backlog, Lean, And Throughput

---

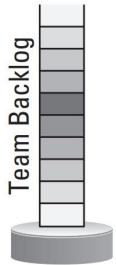
## What Are The Backlog Benefits?

- It prevents the team from getting instructions from a variety of sources.
- Since the backlog has a single owner, teams always know what the current priorities are and who to ask for clarification.
- The fact that “if it isn’t in there, it isn’t going to happen,” communicates to all the stakeholders how the team does their work.
  - ◆ It also informs stakeholders how to influence that work. That is, by working with the product owner to get their item in the backlog.
- The fact that the backlog is prioritized at iteration boundaries means that priorities are current and local.

# Backlog, Lean, And Throughput

---

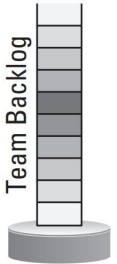
- Therefore, agile software teams spend a significant portion of their time identifying, estimating, and elaborating items in their backlog.
- For some teams, a nice, well-formed, and deep backlog gives them a sense of control of their destiny.
  - ◆ They can see the work ahead.
  - ◆ They can plan for current and future work.
  - ◆ They have a sense of comfort in knowing that they are always working on the next higher-prioritized thing.



# Backlog, Lean, And Throughput

---

- A deep, well-considered, estimated, and elaborated backlog is our silver bullet for managing development.
- But we must take a look at the backlog from another perspective.
  - ◆ The perspective of lean
  - ◆ How backlogs affect time to market for new ideas and initiatives.
- After all, a backlog is nothing less than a queue of work, and long queues of work are not so good.



## Contents

1. Iterating: The Heartbeat Of Agility
2. **Backlog, Lean, And Throughput**

## 2. Backlog, Lean, And Throughput

- a. **Is Well-formed Product Backlog Decreasing Team's Agility?**
- b. Little's Law And An Agile Team's Backlog
- c. Applying Little's law to Increase Agility And Decrease Time To Market
- d. Managing Throughput By Controlling Backlog Queue Length

# Is Well-formed Product Backlog Decreasing Team's Agility?

---

- How big product backlog needs to be?
- How well formed or how well elaborated product backlog needs to be?
- The size of the team's backlog, the rate of backlog processing, and the ultimate rate of value delivery is really a problem of queuing theory.
- Little's law, the general-purpose theory for queuing and processing problems, is one of the fundamental laws of lean.

$$W_q = \frac{L_q}{\lambda}$$

# Is Well-formed Product Backlog Decreasing Team's Agility?

---

- **W<sub>q</sub>** is the average waiting time in the queue for a standard job.
- **L<sub>q</sub>** is the average number of things in the queue to be processed.
- **Lambda** is the average processing rate for jobs in the queue.

$$W_q = \frac{L_q}{\lambda}$$

## Contents

1. Iterating: The Heartbeat Of Agility
2. **Backlog, Lean, And Throughput**

## 2. Backlog, Lean, And Throughput

- a. Is Well-formed Product Backlog Decreasing Team's Agility?
- b. Little's Law And An Agile Team's Backlog**
- c. Applying Little's law to Increase Agility And Decrease Time To Market
- d. Managing Throughput By Controlling Backlog Queue Length

# Little's Law And An Agile Team's Backlog

---

Example: Let's assume the following.

- A single **agile/Scrum team**, working in **two-week iterations**.
- The **team averages** about **25 to 30 story points per iteration**.
- A **story completion rate** of about **8 stories per iteration**.
- The team is justifiably proud of how well they are maintaining their backlog, and the backlog averages about **100 stories**, most of which are **committed to near-term releases**.

# Little's Law And An Agile Team's Backlog

---

Example: Let's assume the following.

$$W_q = \frac{100 \text{ Stories}}{\left( \frac{8 \text{ Stories}}{1 \text{ Iteration}} \right)} = 12.5 \text{ Iterations}$$

- The answer is **12.5 iterations** to **get into the sprint**, **plus 2 weeks** to get out, or **27 weeks on the average**. More than half of a year!
- And, if your item is in the back of the backlog, it could take even longer.
- If it takes a team on average half a year to deliver a new requirement to the customer, **that doesn't seem very responsive**.

# Little's Law And An Agile Team's Backlog

---

- Plus, in the enterprise, there are multiple teams with **interdependencies**, and the **individual results** of the **teams** have to be **aggregated**, **packaged**, and **validated** in some kind of release envelope before distribution, so it can take longer still.
- Therefore, it's understandable when we see an **enterprise** with **20**, **50**, or even **100** reasonably **agile teams** that it still takes **300** to **500 days** to move a **new requirement** from customer request to delivery.
- So although this may be an exaggerated case (as we'll see in the comments shortly), it is not at all unusual to see these types of delays in the enterprise.
- So yes, **it may be understandable**, but no, **it's not acceptable**. Let's see **what we can do about it**.

## Contents

1. Iterating: The Heartbeat Of Agility
2. **Backlog, Lean, And Throughput**

## 2. Backlog, Lean, And Throughput

- a. Is Well-formed Product Backlog  
Decreasing Team's Agility?
- b. Little's Law And An Agile Team's Backlog
- c. **Applying Little's law to Increase  
Agility And Decrease Time To Market**
- d. Managing Throughput By Controlling  
Backlog Queue Length

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

- The formula is not complicated.
- If we are going to improve (decrease) time to market, we have to either increase the denominator or decrease the numerator (or both).
- And of course, if we can do both, we will achieve even better results. Let's look at each opportunity.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion

- Increasing the rate of story completion, and thereby the overall rate of value delivery, is the legitimate goal of every agile team.
- Of course, if we could simply add resources, we could probably increase Lambda, but for the purpose of this discussion, let's assume that is impractical.
- Besides, although it's the easy way out of the argument, it increases the cost of the value created and decreases return on investment (ROI).

$$W_q = \frac{L_q}{\lambda} \leftarrow$$

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

- So although you might get there faster, you may not make any money when you do.
- Let's work within the fixed resource constraints of our archetypical team and see what we can do.
- The primary mechanism for increasing the rate of story completion is the team's inspect and adapt process, whereby the teams review the results of each iteration and pick one or two things they can do to improve the velocity of the next.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

- This is the long-term mission; it is a journey measured in small steps, and there is no easy mathematical substitute for such improvements.
- These improvements include better coding practices, unit testing and unit testing coverage, functional test automation, continuous integration, and other enhanced agile project management and software engineering practices.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

- In experience, however, two primary areas stand out as the place where teams can get the fastest increase in Lambda:
  - ◆ First, gaining a better understanding of the story itself before coding begins.

Acceptance Test-Driven Development
  - ◆ Second, decreasing the size of the user stories contained in the backlog.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

### → Acceptance Test-Driven Development

- ◆ The fact is that the overall velocity of the team is not typically limited by the team's ability to write, or even integrate, code.
- ◆ Instead, it is gated by the team's ability to understand what specific code they need to write, as well as to avoid code they do not need to write.
- ◆ Doing so involves having a better understanding of the requirements of the story, before coding it.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

### → Acceptance Test-Driven Development

- ◆ However, this must be done on a just-in-time basis, just prior to the iteration boundary, or else the team's backlog will get wider, and the team will have too much requirements inventory.
- ◆ Some of it will likely decay before they get to it. However, a wider backlog (small numbers of well-elaborated stories) is not nearly as bad as a deeper backlog (larger numbers).
- ◆ The worst case is that a few team members have gone too far, too early, in elaborating a few backlog items, but it won't slow value delivery nearly so badly as would a deeper backlog. It's a bit of waste, but it doesn't really drive Little's law.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda, The Rate of Story Completion (Cont.)

### → Acceptance Test-Driven Development

- ◆ Therefore, once a story has reached a priority whereby it will be implemented in the next iteration or two, time spent in elaborating the story will pay dividends.
- ◆ Often, this is described as Acceptance Test-Driven Development (ATDD), and, fortunately, it's a little easier for teams to intellectualize and adopt than code-level TDD.
- ◆ ATDD involves two things: writing better stories and establishing the acceptance tests for the story before coding begins.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda with Smaller Stories

- Small jobs just go through a system faster than large ones.
- Decreasing the size of user stories has both a linear and exponential effect on Lambda, both of which are positive.
- **Linear Effect**
  - ◆ Smaller user stories are just that, smaller.
  - ◆ They go through the iteration faster so teams can implement and test more small user stories in an iteration than large ones.
  - ◆ And, although the total value of a small story can't be as big as a large story, the incremental delivery hastens the feedback loop, improving quality and fitness for use.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda with Smaller Stories (Cont.)

- Small jobs just go through a system faster than large ones.
- Decreasing the size of user stories has both a **linear** and **exponential effect** on Lambda, both of which are positive.
- **Exponential Effect**
  - ◆ Because they are smaller and less complex, small user stories decrease the coding and testing implementation effort.
  - ◆ The coded functions are smaller and less complex, and the number of new paths that must be tested also decreases exponentially with story size.
  - ◆ (However, some of this is offset by the additional overhead of managing more stories.)

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Increasing Lambda with Smaller Stories (Cont.)

- So, even if the length of the backlog remains the same, a combination of better defined and smaller user stories has a very positive effect on value delivery time.
- Clearly, increasing Lambda, the denominator of Little's law, is a prime opportunity for the team to increase agility and time to market. Every truly agile team continuously commits to doing so.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Decreasing $L_q$ , The Length of The Queue

- let's see what opportunities we find here.
- Decrease time to market while the team is working on continuously improving development practices.
- That is by forcing a limit on the length of the queue.
- Decreasing queue size causes a directly proportional decrease in the wait time.
- Therefore, if we cut our queue size in half, we can halve our time to market.

# Applying Little's Law To Increase Agility And Decrease Time To Market

---

## Decreasing $L_q$ , The Length of The Queue

- Why long queues are fundamentally bad in the product development process?
  - ◆ Increased risk
  - ◆ Increased variability
  - ◆ Increased costs
  - ◆ Reduced quality
  - ◆ Reduced motivation and initiative

## Contents

1. Iterating: The Heartbeat Of Agility
2. **Backlog, Lean, And Throughput**

## 2. Backlog, Lean, And Throughput

- a. Is Well-formed Product Backlog  
Decreasing Team's Agility?
- b. Little's Law And An Agile Team's Backlog
- c. Applying Little's law to Increase Agility  
And Decrease Time To Market
- d. **Managing Throughput By Controlling  
Backlog Queue Length**

# Managing Throughput By Controlling Backlog Queue Length

---

Point #1: Little's Law Doesn't Lie

Point #2: Don't Be Hostage to Your Own Backlog

Point #3: The Enterprise Can't Be Held Hostage to the Team's Backlog Either

End of Chapter 9

# Contributions

---

- Author of Reference Book: **Dean Leffingwell**
- Course Instructor: **Mehran Rivadeh**
- Slide Creator: **Mahnaz Rasekhi**
  - ◆ These slides are primarily based on Agile Software Requirements by Dean Leffingwell, with occasional adaptations to enhance clarity and engagement.