BITS Pilani

Pilani Campus

BITS Pilani presentation

K.Anantharaman
Faculty CS Department
kanantharaman@wilp.bits-pilani.ac.in

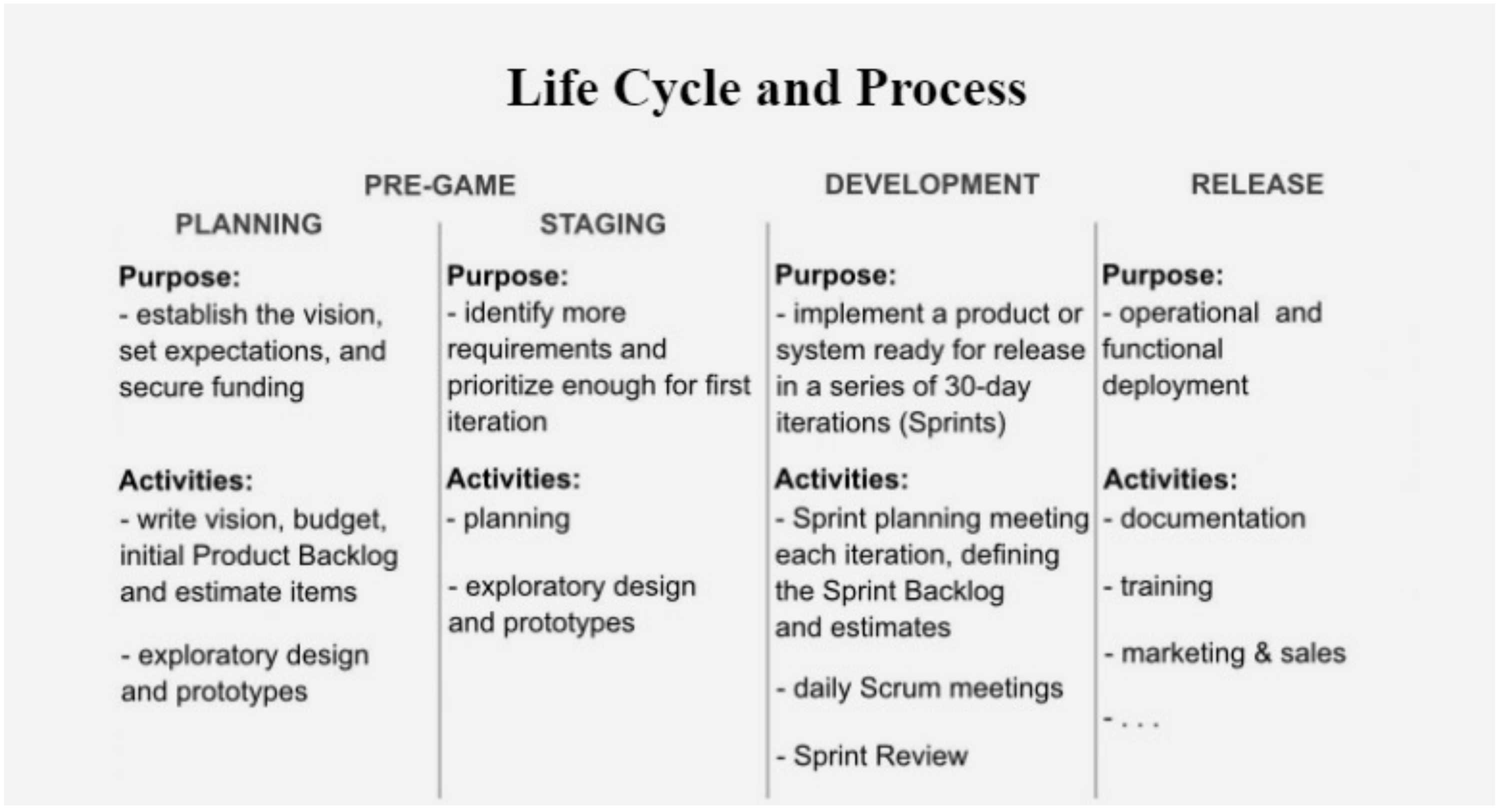
BITS Pilani
Pilani Campus

SE ZG544 , Agile Software Process Lecture No. 4 - Agile Methodologies

Agenda

- Agile Methodologies
- Scrum
- XP
- Lean Software Development
  - Kanban
  - Value Stream Mapping

Scrum Life Cycle Process



Life Cycle and Process

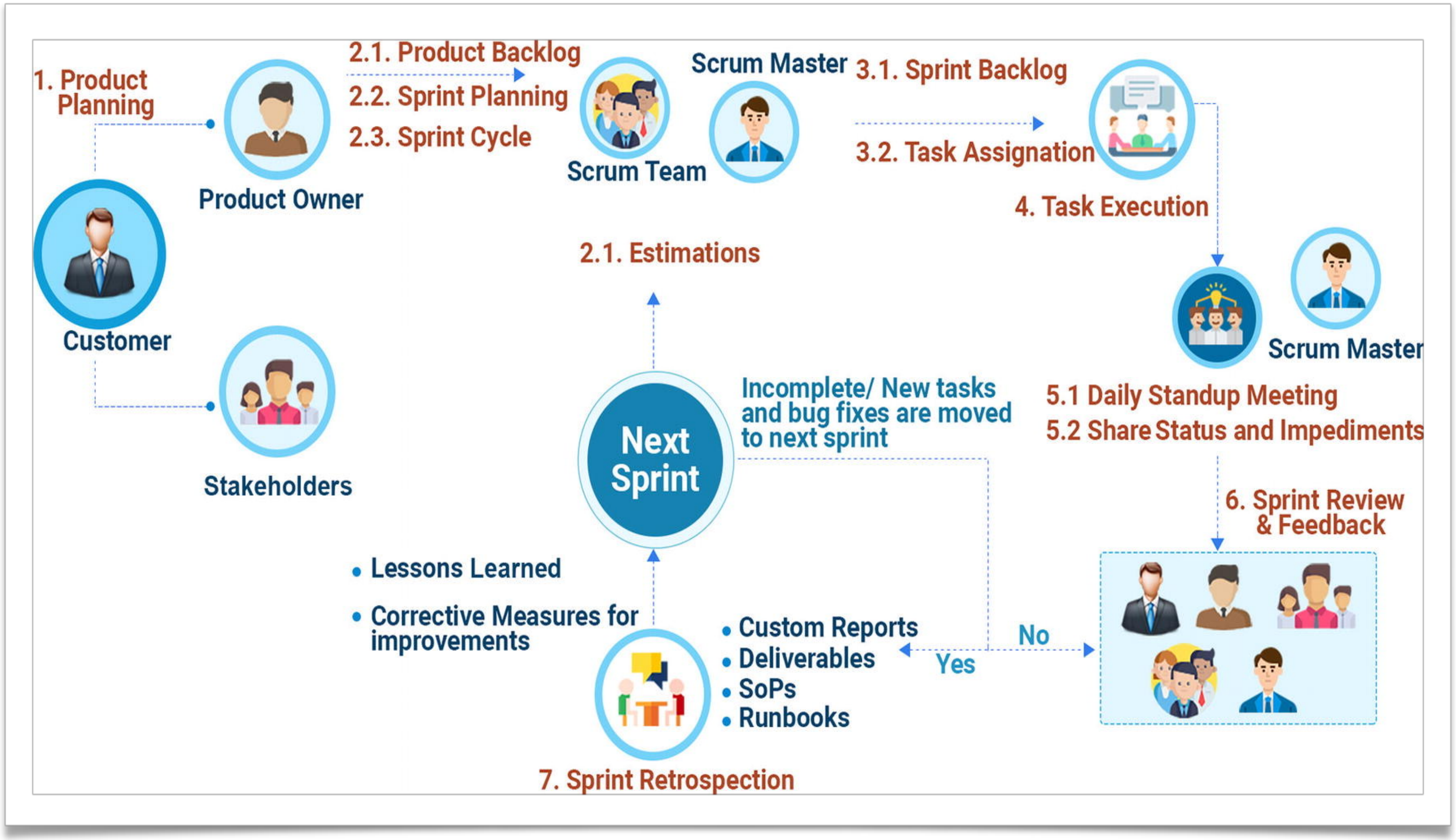
PRE-GAME: PLANNING	PRE-GAME: STAGING	DEVELOPMENT	RELEASE
Purpose: - establish the vision, set expectations, and secure funding	**Purpose:** - identify more requirements and prioritize enough for first iteration	**Purpose:** - implement a product or system ready for release in a series of 30-day iterations (Sprints)	**Purpose:** - operational and functional deployment
Activities: - write vision, budget, initial Product Backlog and estimate items - exploratory design and prototypes	**Activities:** - planning - exploratory design and prototypes	**Activities:** - Sprint planning meeting each iteration, defining the Sprint Backlog and estimates - daily Scrum meetings - Sprint Review	**Activities:** - documentation - training - marketing & sales - . . .

Ref: So, What's The Big Deal About Scrum?,by by André Akinyele,2019

Scrum Model:
Flow, Artifacts, Roles and Responsibilities

Product Backlog User Stories

Sprint	Start	End
1	20	15
2	15	11
3	11	5
4	7	



Source: Hands-On Guide to AgileOps: A Guide to Implementing Agile, DevOps, and SRE for Cloud Operations By Navin Sabharwal, Raminder Rathore, Udita Agrawal

Scrum Project Life cycle

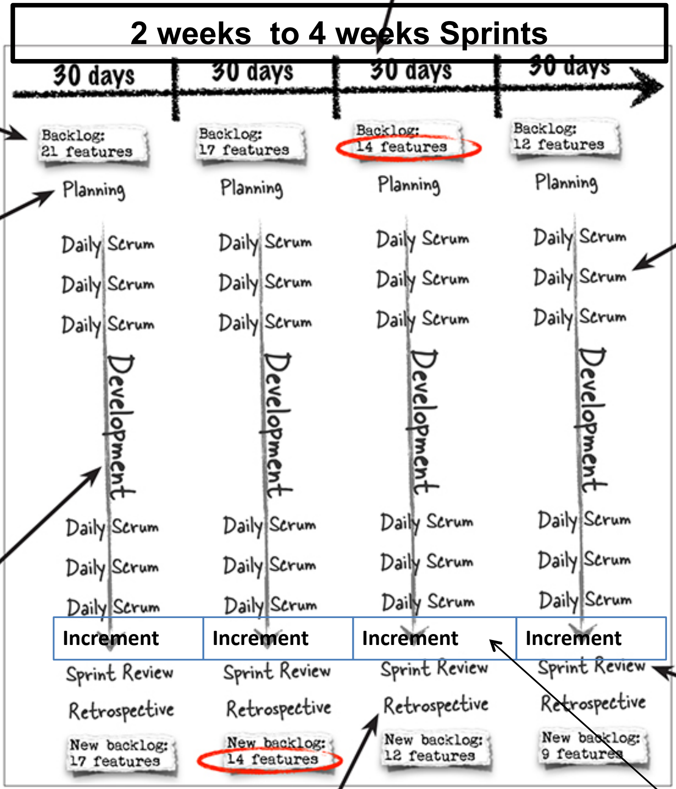
This is the single source for requirements and any changes that will be made to the product throughout the project.

1 The team uses the Product Backlog to keep track of the features that they'll build for the whole project.

2 At the beginning of each Sprint, the Scrum Team gets together for a Sprint Planning session where they choose what items to include in the Sprint.

3 The items for the Sprint are pulled off of the Product Backlog and added to the Sprint Backlog. During the Sprint, all of the development work is focused on building the items in the Sprint Backlog.

Every Scrum project is organized into timeboxed iterations called Sprints. Many teams use 30-day Sprints, but it's pretty common to see two-week Sprints too.



Every day, the team holds a Daily Scrum, a short meeting where each person gives an update on their progress, what they'll work on next, and any roadblocks they've hit.

4 When Sprint is done, the team holds a meeting called the Sprint Review, where they meet with the users and give a demo of the working software that they built.

5 The very last thing the team does in the Sprint is meet and hold a Sprint Retrospective, where they talk about what happened during the Sprint so they can reproduce the things that went well and learn from any problems.

PSI-Potentially shippable increment

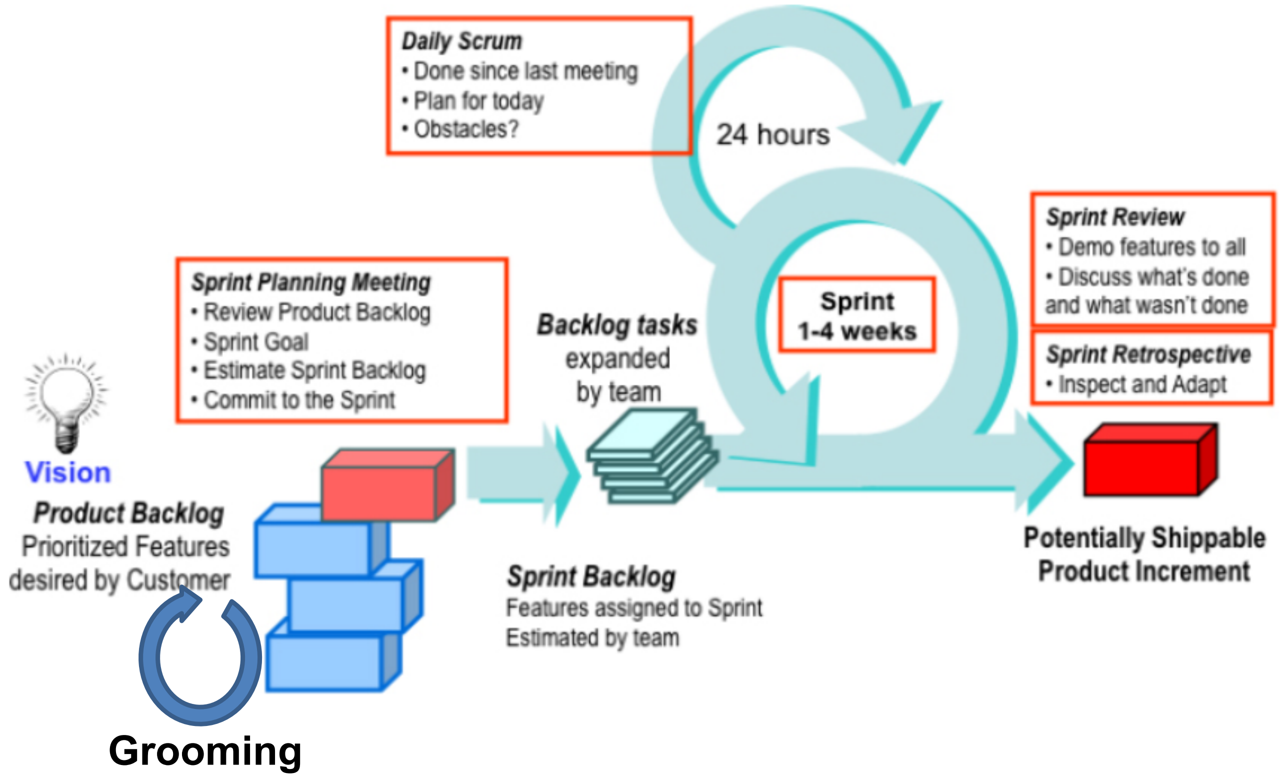
The Scrum events:
1. The Sprint
2. The Sprint Planning
3. The Daily Scrum
4. The Sprint Review
5. The Sprint Retrospective

The Scrum Artifacts:
1. The Product Backlog
2. The Sprint Backlog
3. The Product Increment

The Scrum Roles:
1. The Product Owner
2. The Scrum Master
3. The Team

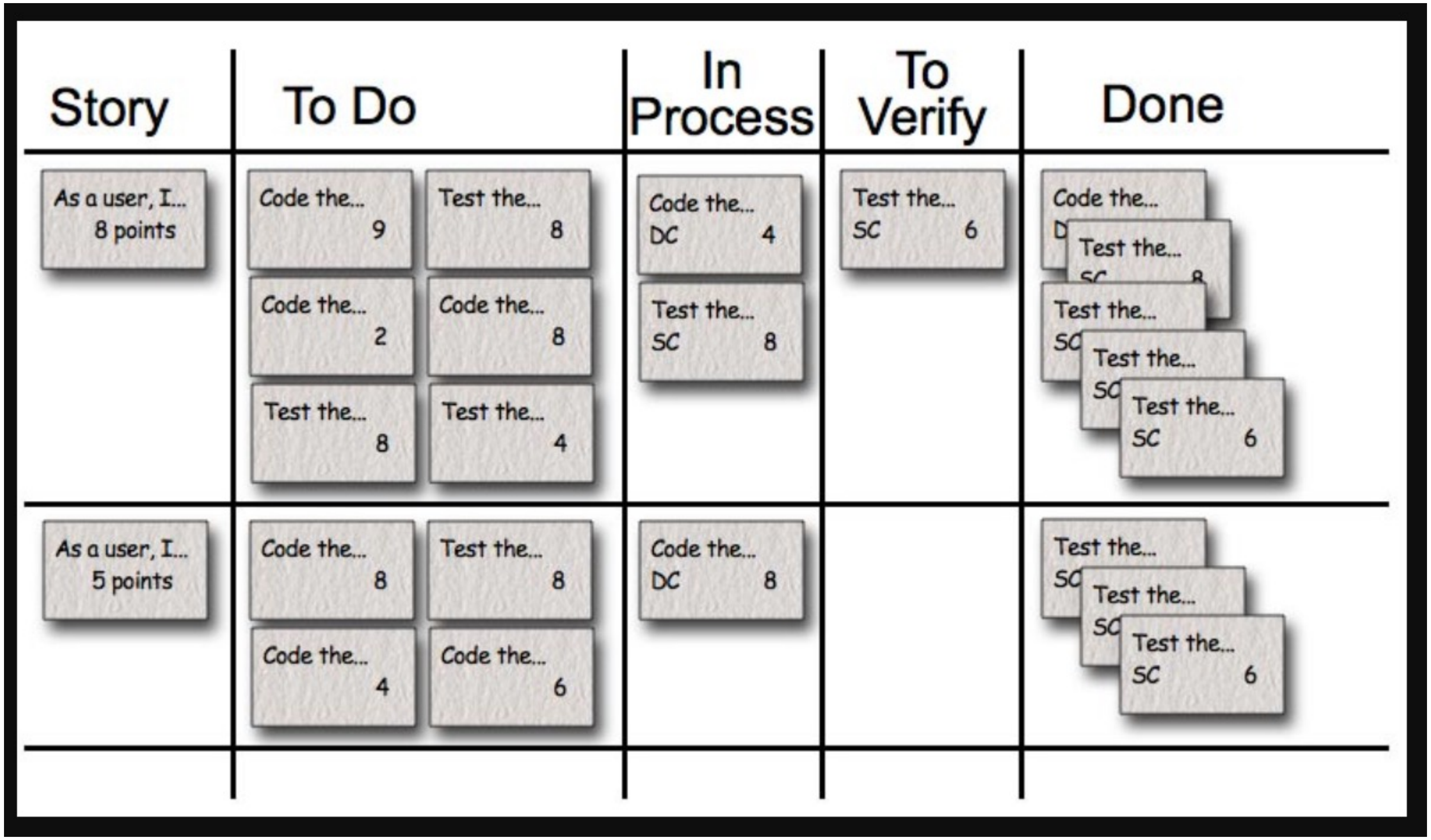
Ref:Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

Scrum Project Lifecycle


Daily Scrum
• Done since last meeting
• Plan for today
• Obstacles?
24 hours
Sprint Planning Meeting
• Review Product Backlog
• Sprint Goal
• Estimate Sprint Backlog
• Commit to the Sprint
Backlog tasks expanded by team
Sprint 1-4 weeks
Sprint Review
• Demo features to all
• Discuss what's done and what wasn't done
Sprint Retrospective
• Inspect and Adapt
Vision
Product Backlog Prioritized Features desired by Customer
Sprint Backlog Features assigned to Sprint Estimated by team
Potentially Shippable Product Increment
Grooming

Agile project management with Scrum

Progress Tracking - Scrum Task Board

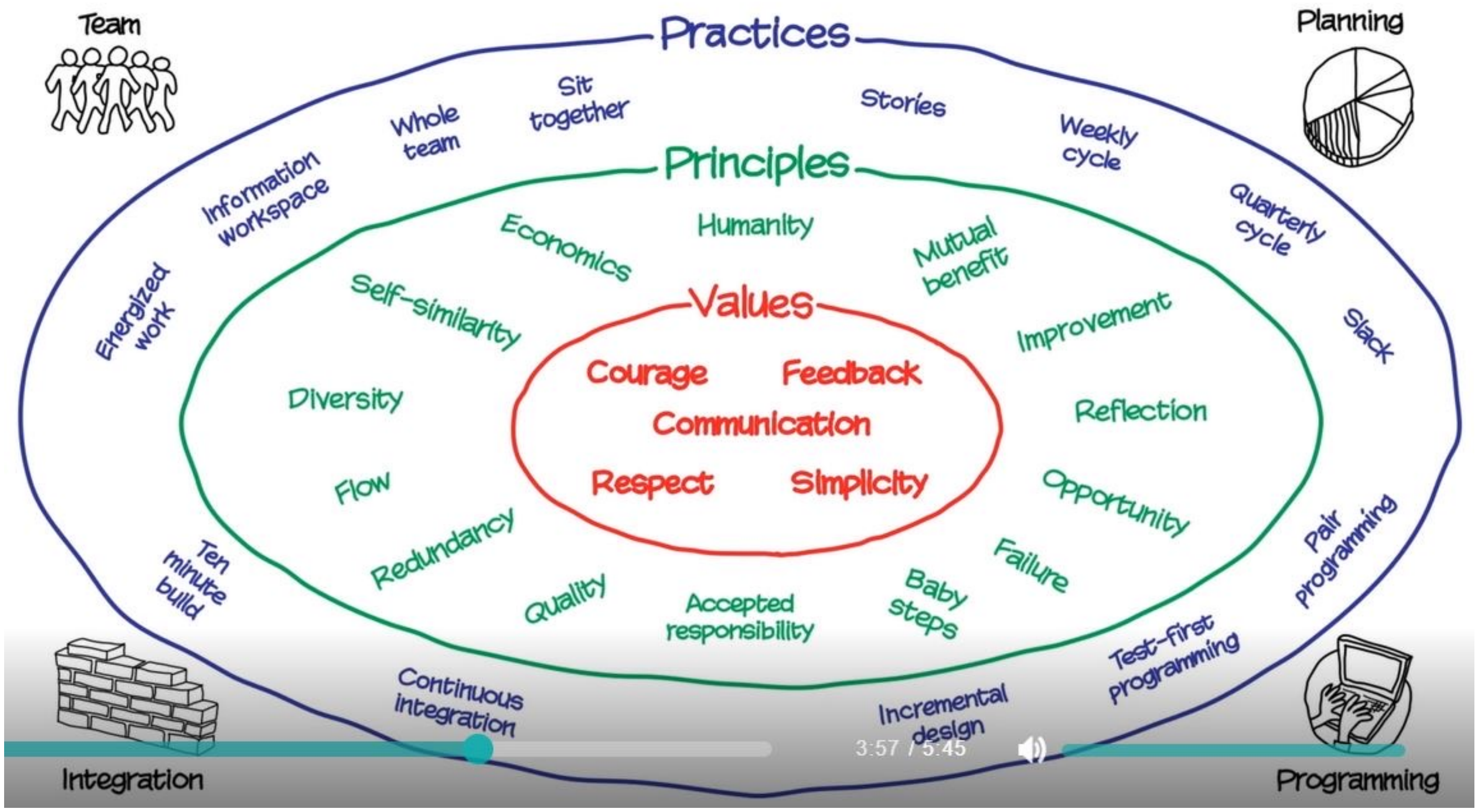


Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 2005 innovate achieve lead

Q&A

eXtreme Programming (XP)

(Similar to Scrum Model with some differences)



Ref :Agile Sketchpad By Dawn Griffiths and David Griffiths

XP Practices

- Test-Driven Development
- Refactoring
- Pair Programming
- 10-Minutes Build
- Continuous Integration

Scrum & XP Differences

Focus:
Scrum: Project Management Framework
XP: Engineering Practices & Techniques

Roles and Responsibilities:
Scrum: Scrum defines specific roles such as Product Owner, Scrum Master, and Development Team.
XP: Encourages collective ownership

Iterations and Releases:
SCRUM: 1 to 4 weeks iteration
XP: 1 week iteration, prefer shorter time scales.

Engineering Practices:
Scrum: No prescribed Engineering Practices.
XP: Extreme Programming (XP) promotes a set of engineering practices designed to ensure high-quality software, including Test-Driven Development (TDD), Pair Programming, Continuous Integration, Refactoring, and Simple Design.

Q&A

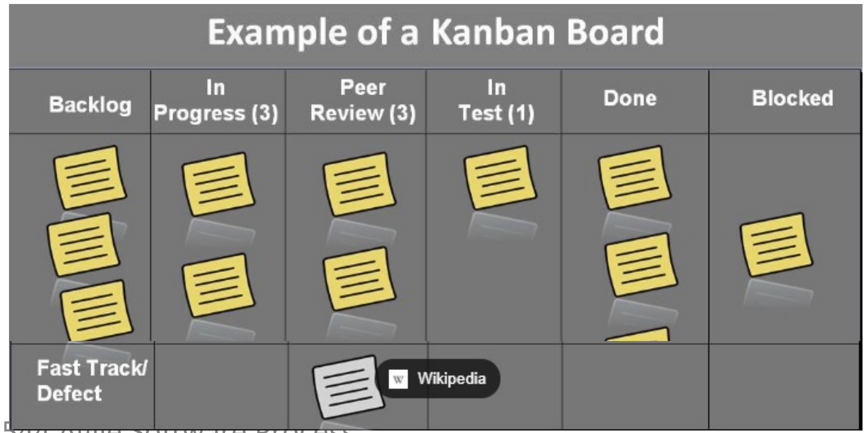
What is Lean Software Development? & Seven Principles of Lean

- Lean is a systematic method to eliminate waste and maximize the flow of value through a system. Value is defined as something your customer will pay money for.

1. Eliminate Waste
2. Build Quality In
3. Create Knowledge
4. Defer Commitment
5. Deliver Fast
6. Respect People
7. Optimize the Whole

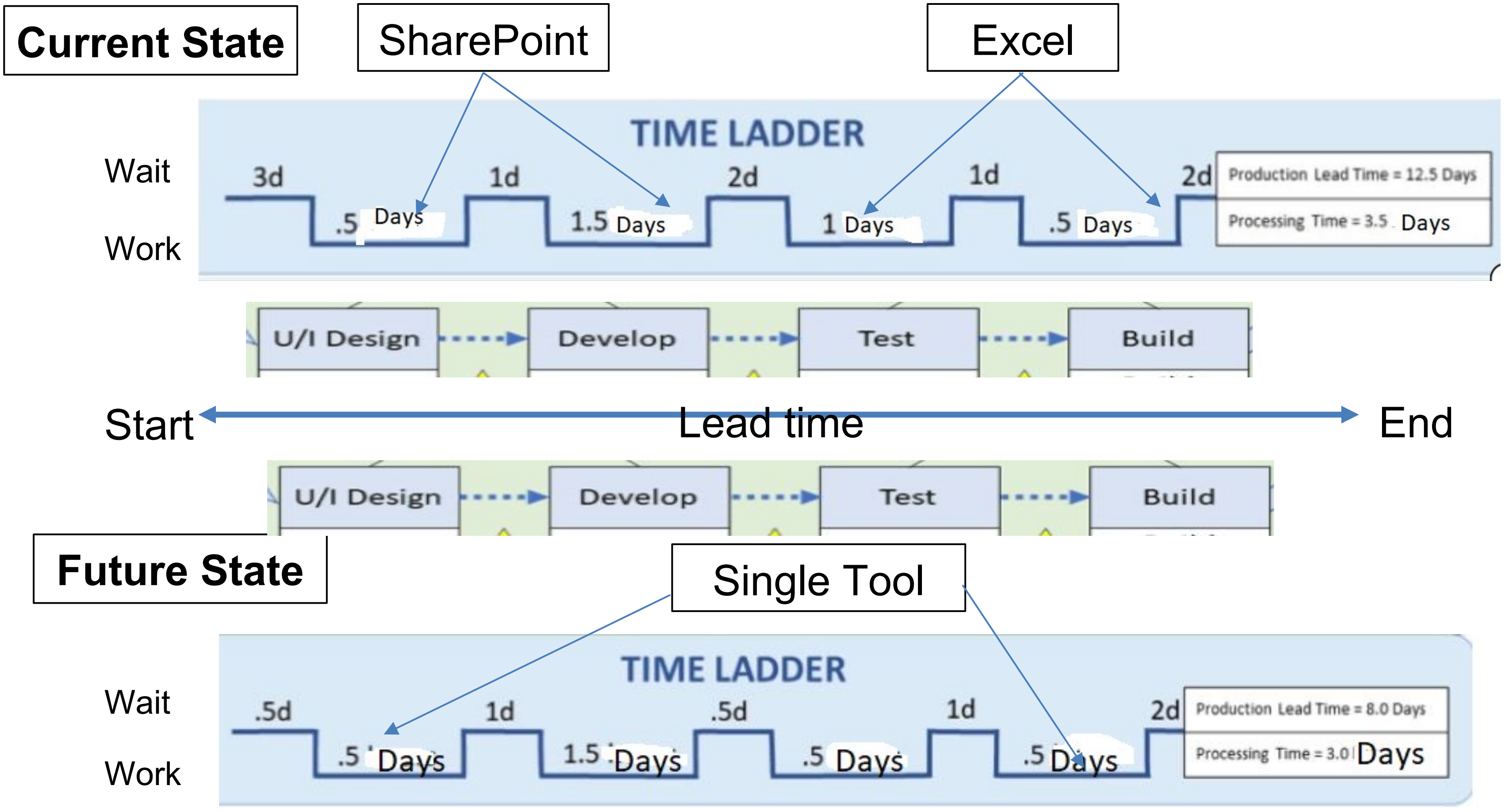
- Value Stream Mapping
- Kanban

Value	Value Stream	Value Stream Mapping
Anything that adds value to the customer–anything for which a customer is willing to pay.	The set of actions an organization takes to deliver value to a customer in the form of a product or service.	The process of visualizing the current state and designing the future state of a value stream to identify areas of improvement.



[Ref: What is Value Stream Mapping? Utilizing Value Streams | Planview](https://www.planview.com)

Value Stream Mapping



Process Efficiency = Cycle time/Lead Time * 100

Current State: 3.5/(12.5)*100 = 28%; Future state: 3/(8)*100 = 37.5%

Ref: https://www.plutora.com/blog/value-stream-mapping

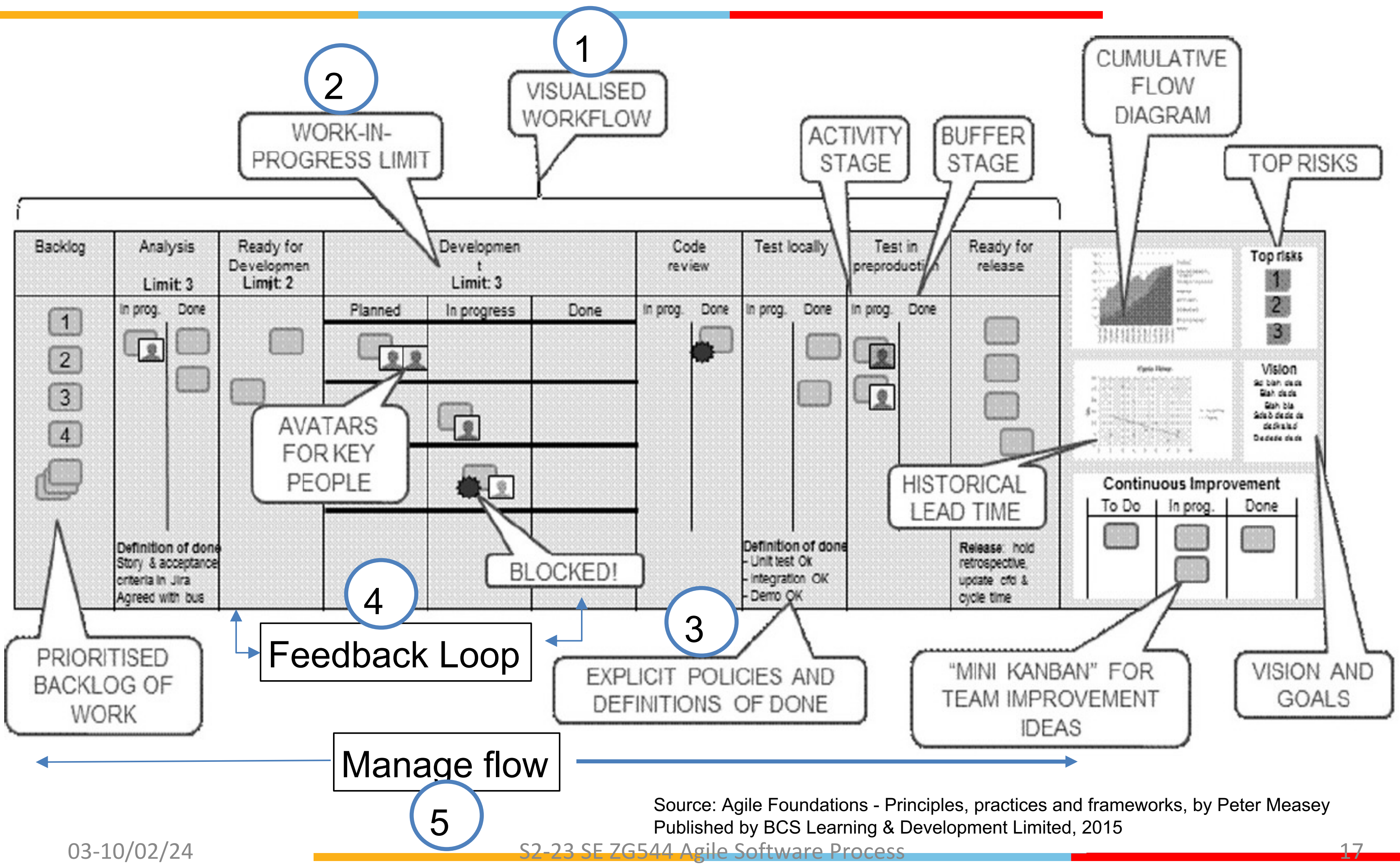
Kanban

- Kanban is a popular framework used to implement agile and DevOps software development. It requires real-time communication of capacity and full transparency of work

- Kanban is not an Agile software development method (or process) or a software engineering methodology

- Kanban is flow based methodology and a pull system.
- Kanban does not prescribe specific roles or process steps as it is built on the concept of evolutionary change.

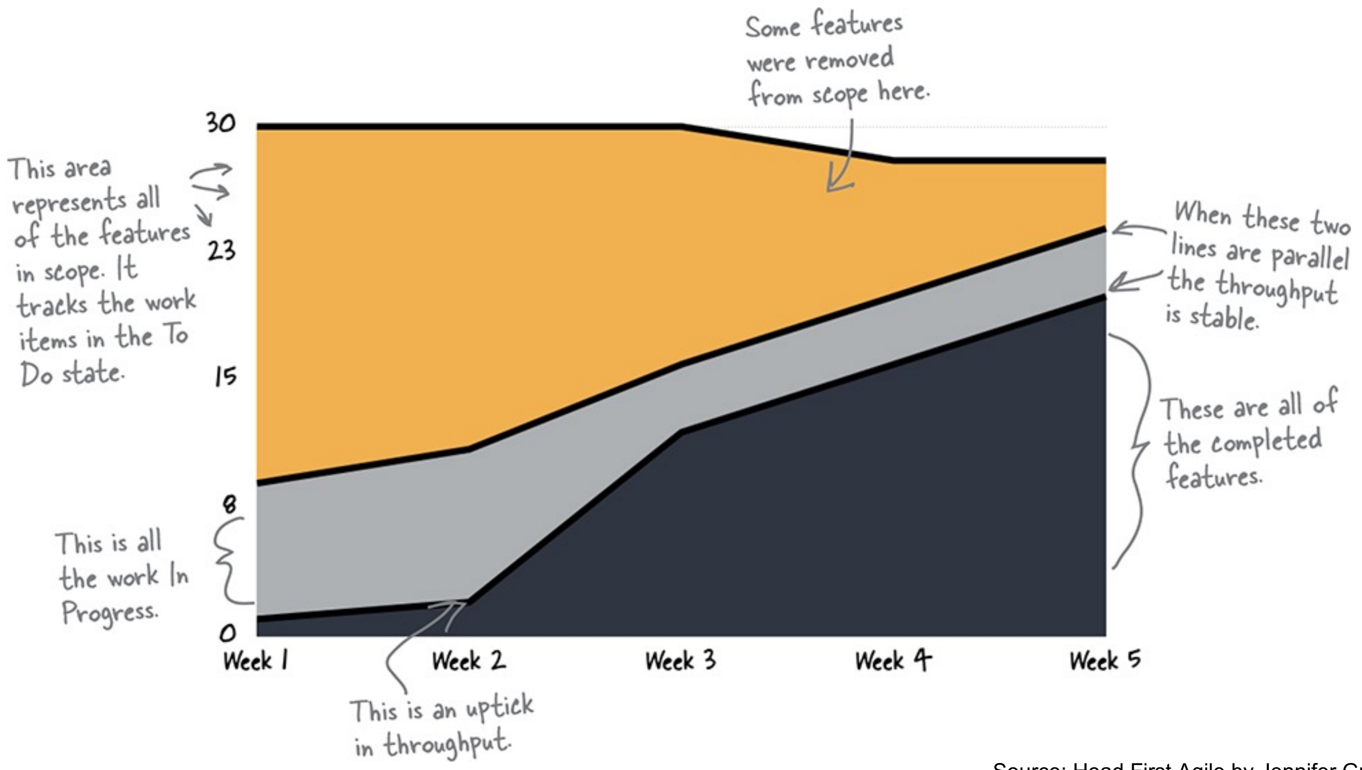
Typical Kanban Board



Source: Agile Foundations - Principles, practices and frameworks, by Peter Measey
Published by BCS Learning & Development Limited, 2015

Cumulative flow diagram (Example-1)

- Kanban teams use cumulative flow diagrams (or CFDs) to find out where they are systematically adding waste and interrupting their flow.



Source: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

Q&A

Common challenges faced in Scrum, XP, Lean, and Kanban methodologies (This is not an exhaustive list)

Methodology	Common Challenges	Possible Solutions
Scrum	Incomplete Backlog Items	Backlog grooming, stakeholder collaboration
	Scope Creep	Stable sprint backlog, focus on sprint goal
	Overcommitting	Realistic planning, consider team capacity
XP	Resistance to Pair Programming	Training, emphasize benefits of pair programming
	Lack of Test Automation	Integrate TDD, provide testing tools
	Continuous Integration Challenges	Regular CI/CD process review, automated deployment
Lean	Identifying Waste	Value stream mapping, eliminate non-value-added tasks
	Lack of Cross-Functional Collaboration	Cross-functional teams, open communication
	Limited Customer Involvement	Regular feedback sessions, user testing
Kanban	Work Overload	Set WIP limits, prioritize completion over new tasks
	Unclear Workflow	Define visual workflow, optimize and communicate
	Managing Blocked Items	Promptly identify and resolve blocked items

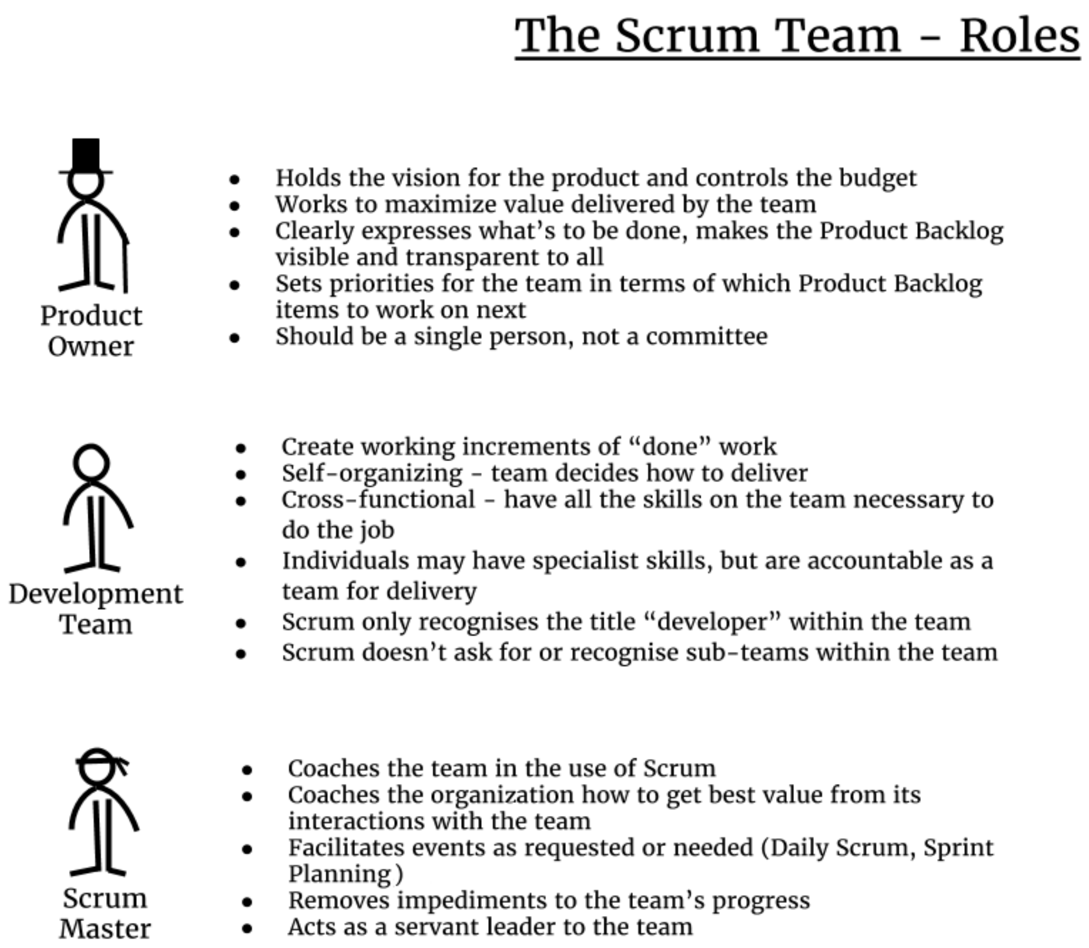
BITS Pilani
Pilani Campus

Module-4 - Agile Methodologies – Additional Notes

Scrum

- History and Origins
- Scrum is a **single-team process framework** used to manage product development.
- Empirical process framework
  - **Empirical method** : A process how you think something works, test it out, reflect on the experience, and make the proper adjustments
  - **Inspection, Adaption, Transparency**
  - Based on adaptive life cycle method (Iterative and Incremental)
- Scrum is the most common approach to agile for good reasons:
  - The **rules of Scrum** are straightforward and easy to learn and teams all around the world have been able to adopt them and improve their ability to deliver projects.
  - **Using Scrum effectively is not so simple**

Scrum Roles



The Scrum Team – Roles

Product Owner

- Holds the vision for the product and controls the budget
- Works to maximize value delivered by the team
- Clearly expresses what's to be done, makes the Product Backlog visible and transparent to all
- Sets priorities for the team in terms of which Product Backlog items to work on next
- Should be a single person, not a committee

Development Team

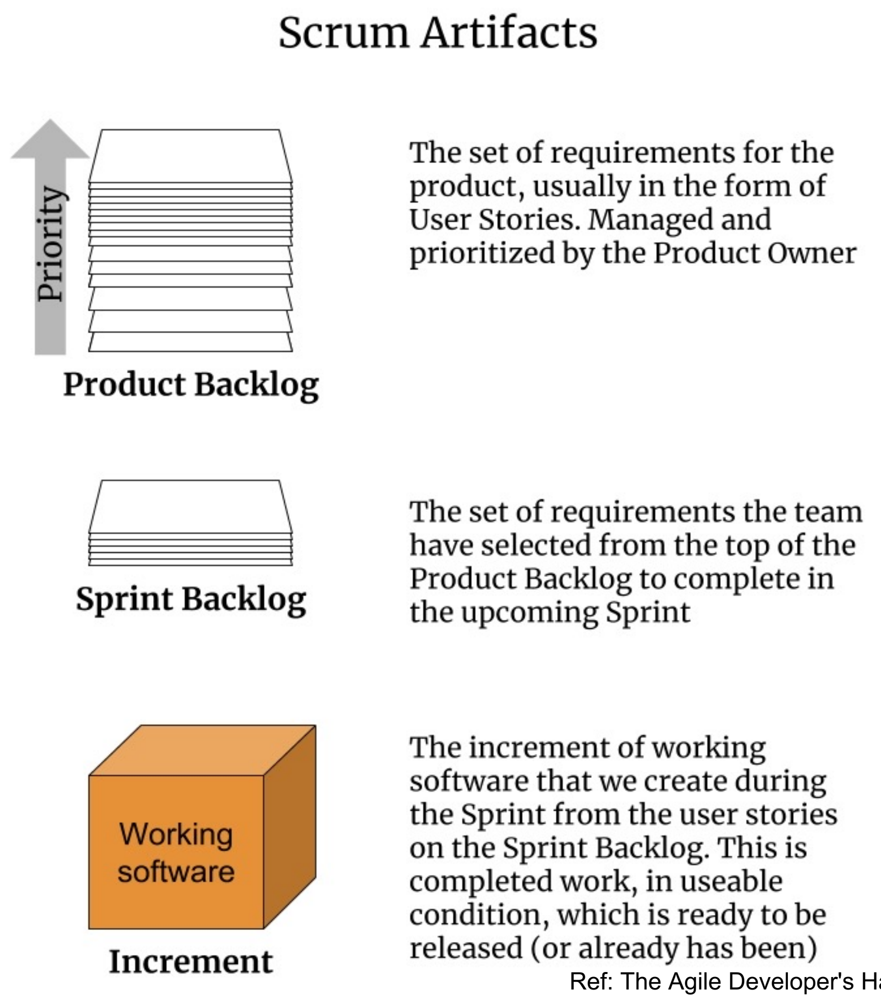
- Create working increments of "done" work
- Self-organizing – team decides how to deliver
- Cross-functional – have all the skills on the team necessary to do the job
- Individuals may have specialist skills, but are accountable as a team for delivery
- Scrum only recognises the title "developer" within the team
- Scrum doesn't ask for or recognise sub-teams within the team

Scrum Master

- Coaches the team in the use of Scrum
- Coaches the organization how to get best value from its interactions with the team
- Facilitates events as requested or needed (Daily Scrum, Sprint Planning)
- Removes impediments to the team's progress
- Acts as a servant leader to the team

Ref: The Agile Developer's Handbook, by Paul Flewelling,Published by Packt Publishing, 2018

Scrum Artifacts



The set of requirements for the product, usually in the form of User Stories. Managed and prioritized by the Product Owner

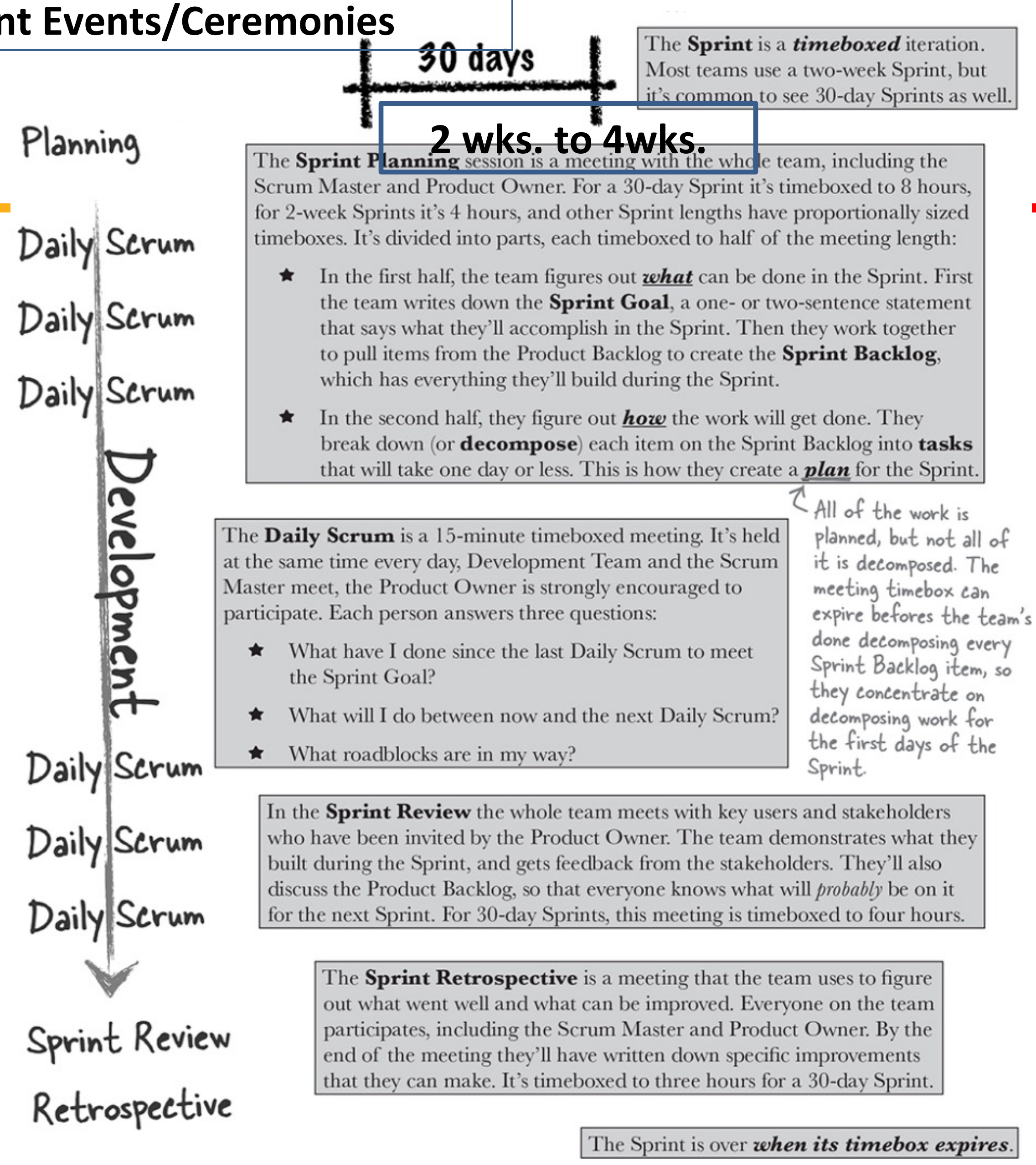
The set of requirements the team have selected from the top of the Product Backlog to complete in the upcoming Sprint

The increment of working software that we create during the Sprint from the user stories on the Sprint Backlog. This is completed work, in useable condition, which is ready to be released (or already has been)

Ref: The Agile Developer's Handbook, by Paul Flewelling,Published by Packt Publishing, 2018

Sprint Events/Ceremonies

2 wks. to 4wks.


30 days
Planning
Daily Scrum
Daily Scrum
Daily Scrum
Development
Daily Scrum
Daily Scrum
Daily Scrum
Sprint Review
Retrospective

The **Sprint** is a ***timeboxed*** iteration. Most teams use a two-week Sprint, but it's common to see 30-day Sprints as well.

The **Sprint Planning** session is a meeting with the whole team, including the Scrum Master and Product Owner. For a 30-day Sprint it's timeboxed to 8 hours, for 2-week Sprints it's 4 hours, and other Sprint lengths have proportionally sized timeboxes. It's divided into parts, each timeboxed to half of the meeting length:

★ In the first half, the team figures out ***what*** can be done in the Sprint. First the team writes down the **Sprint Goal**, a one- or two-sentence statement that says what they'll accomplish in the Sprint. Then they work together to pull items from the Product Backlog to create the **Sprint Backlog**, which has everything they'll build during the Sprint.

★ In the second half, they figure out ***how*** the work will get done. They break down (or **decompose**) each item on the Sprint Backlog into **tasks** that will take one day or less. This is how they create a ***plan*** for the Sprint.

All of the work is planned, but not all of it is decomposed. The meeting timebox can expire befores the team's done decomposing every Sprint Backlog item, so they concentrate on decomposing work for the first days of the Sprint.

The **Daily Scrum** is a 15-minute timeboxed meeting. It's held at the same time every day, Development Team and the Scrum Master meet, the Product Owner is strongly encouraged to participate. Each person answers three questions:

★ What have I done since the last Daily Scrum to meet the Sprint Goal?

★ What will I do between now and the next Daily Scrum?

★ What roadblocks are in my way?

In the **Sprint Review** the whole team meets with key users and stakeholders who have been invited by the Product Owner. The team demonstrates what they built during the Sprint, and gets feedback from the stakeholders. They'll also discuss the Product Backlog, so that everyone knows what will *probably* be on it for the next Sprint. For 30-day Sprints, this meeting is timeboxed to four hours.

The **Sprint Retrospective** is a meeting that the team uses to figure out what went well and what can be improved. Everyone on the team participates, including the Scrum Master and Product Owner. By the end of the meeting they'll have written down specific improvements that they can make. It's timeboxed to three hours for a 30-day Sprint.

The Sprint is over ***when its timebox expires***.

Ref: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

BITS Pilani
Pilani Campus

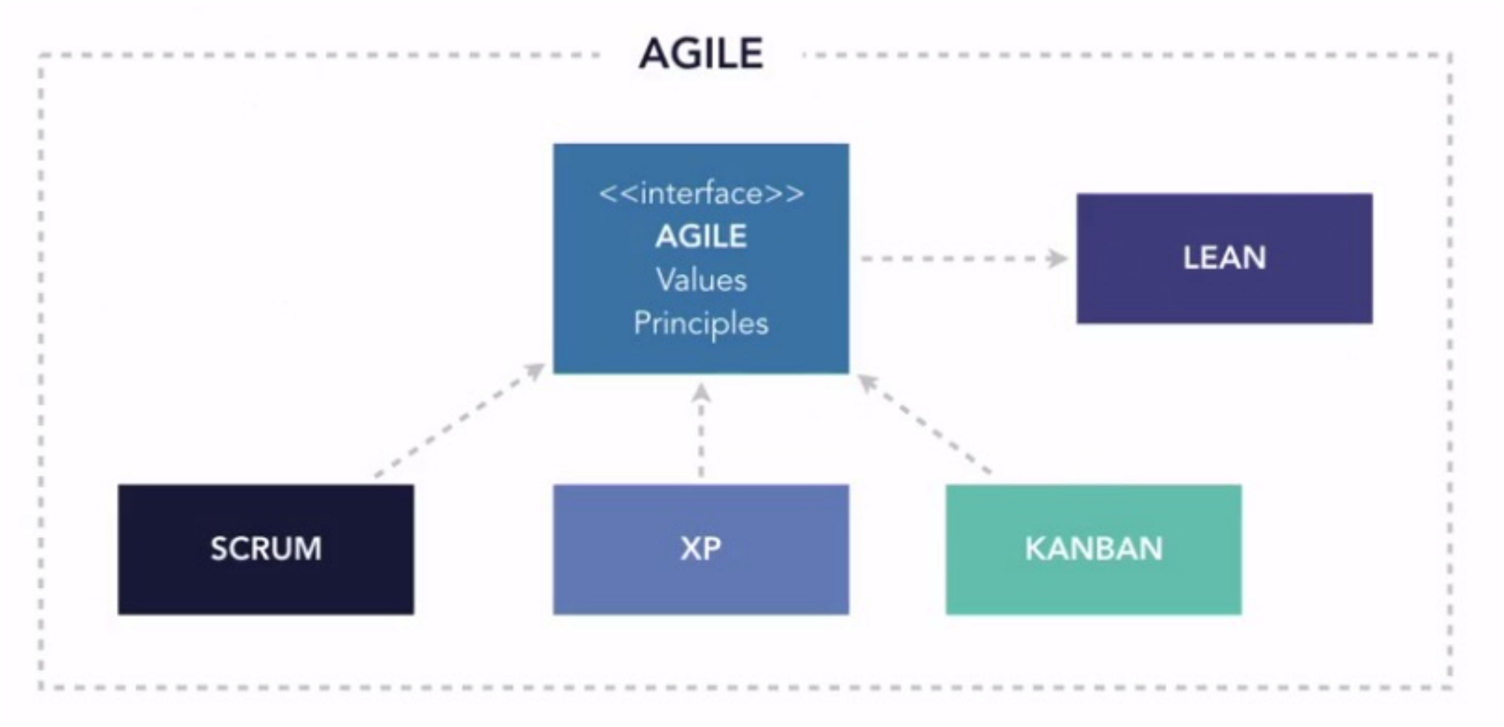
XP- Extreme Programming

References: Agile Foundations - Principles, practices and frameworks, by Peter Measey
Published by BCS Learning & Development Limited, 2015
Scaling Software Agility: Best Practices for Large Enterprises by Dean Leffingwell
Published by Addison-Wesley Professional, 2007

What is eXtreme Programming?

- XP is a widely used agile software development method developed by Ken Beck,2000.
- Key Practices:
  - A team of five to **10 programmers work at one location** with **customer representation on site.**
  - Development occurs in **frequent builds or iterations**, each of which is releasable and delivers incremental functionality.
  - Requirements are specified as **user stories,** each a chunk of new functionality the user requires.
  - **Programmers work in pairs**, follow strict coding standards, and do their own unit testing.
  - Requirements, architecture, and **design emerge** over the course of the project.
  - XP is prescriptive in scope. It is best applied to **small teams** of under 10 developers, and the **customer should be either integral to the team** or readily accessible
- What is Extreme?
  - – Practices are to its purest, simplest form, P-Programming- innovative and sometimes controversial practices for the actual writing of software.
-

How XP fits in Agile



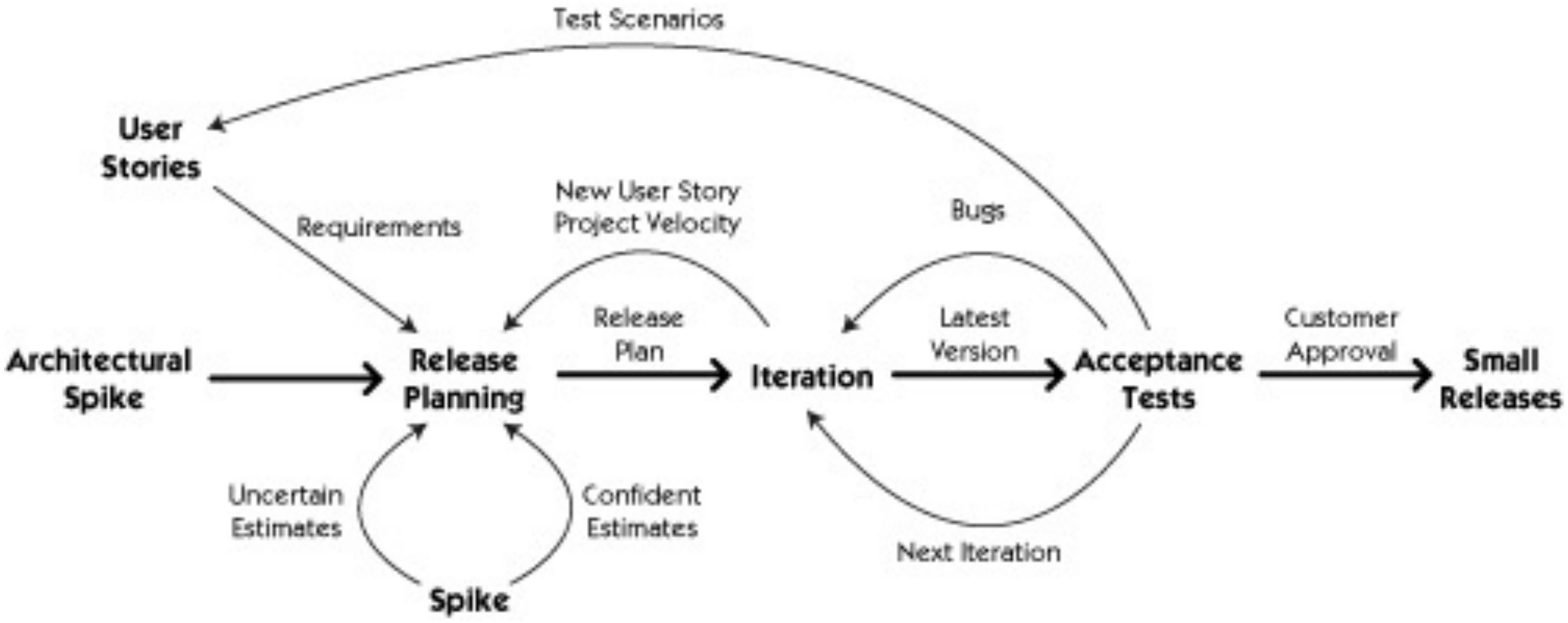
Source :LinkedIn Learning/XP overview

XP Theme

- **The primary theme of XP is that if something hurts, do it all the time.**
  - If code reviews are good, we'll review code all the time (pair programming).
  - If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
  - If design is good, we'll make it part of everybody's daily business (refactoring).
  - If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).
  - If architecture is important, everybody will work at defining and refining architecture all the time (metaphor).
  - If integration testing is important, we will integrate and test several times a day (continuous integration).
  - If short iterations are good, we will make the iterations really, really short—seconds and minutes and hours, not weeks and months and years.
- The Extreme case!

Source :Lynda.com/XP overview

A visual process model for XP



XP Core Values

- **Communication** (Key to Product Quality)
  - Planning, Estimation, Co-location, Pair-programming, Unit tests
- **Feedback** (Ensures stay on course)
  - Short iterations, On-site customer, State of functional tests shows the current development of the project and unit tests shows state of the code base.
- **Simplicity**(Simple design has least bugs and easy to modify
  - "Do the simplest thing that could possibly work" philosophy, focusing on solutions for the current iterations of work and contribute to rapid development of stories.
  - No extra functionality.
- **Respect** (Respect each other ideas we are creating together)
  - Respecting oneself and other team members in the team
- **Courage** (It takes courage to do things you know are right)
  - It takes courage to highlight issues/Architecture flaw even late in the day, it takes courage to throw away the code when you recognize that there is a better design. takes courage to refactor the another developer code, Courage to fail. Change.

Basic XP principles

- Humanity
  - XP's first principle is the simple principle of humanity.
  - Focus on people, empower people, provide benefits to people, and you and your people are likely to find a way to a process that engages people in working together and solving problems in new and innovative ways.
- Rapid feedback
  - Seek feedback at the **earliest possible moment**, interpret it appropriately and **apply learning** from it back into the system. In practice this is **achieved** by the different Planning, **testing activities**, **direct communication with the customer** ,**sharing of knowledge and code** across the whole team.
- Assume simplicity
  - Choose the simplest solution that could solve the problem. By applying the principle of simplicity to development, design and code becomes leaner, resulting in quicker development.
  - The phrase 'You Ain't Gonna Need It' **(YAGNI**) was coined to embody this principle., **DRY (**Don't Repeat Yourselves)

XP Principles …

- Incremental change/Baby Steps
  - Small manageable steps/tasks. Work incrementally, one/two week iterations
- Embrace change
- Quality work
  - An XP team is committed to the principle of doing a good job.
  - By producing quality work, members of an XP team will be proud of their contribution to the project, which becomes a **motivating factor**.
  - Sacrificing quality will only have a negative effect on a project. As one of the fundamental principles of XP, **it should not be optional.**
- Reflection.
  - Retrospect and improve

Further principles

These principles are more specific to particular situations.

- Teach learning
- Small initial investment (Focus on innovation)
- Play to win
- Concrete experiments
- Open and honest communication
- Work with people's instincts, not against them
- Accepted responsibility
- Local adaptation
- Travel light
- Honest measurement

Key XP Practices

The planning game:

Release planning: (Monthly or Quarterly)

- **User-stories,** Customer responsibility, Stakeholders
- **Exploration phase** (Elaboration, estimation – **Whole Team** activity)
- **Commitment phase** (Based on the business value, combined with the estimates, the customer will decide the scope and date of the next release)
- **Steering phase (**Weekly cycle - executed over the remaining time till release)
  - Feedback from each iteration's delivery is used to **steer** the project. Both the customer and team have opportunities during the steering phase to make changes.

XP Practices

- Small releases
  - Small releases can start to gather feedback that can be used in steering the system’s subsequent development, as well as potentially delivering business value early.
- Metaphor
  - Used to form a form an understanding of the system by whole team through the project
  - Example: Shopping cart as metaphor to discuss e-Commerce application requirements.
- Simple design
- Testing
  - All **stories** are to have automated functional tests
  - Indications of progress as new tests are shown to be successful.
  - Confidence in the system as existing tests are shown to be successful.
  - Team is driven by tests , **Test Driven development (TDD)**

XP Practices …

- Refactoring
  - Refactoring is the process of simplifying the internal structure of code without affecting its external behavior
  - TDD = Test first + Refactoring
- Pair programming
  - Code is created by **two** developers using **one** machine.
  - When One person codes, other person in in different perspective - about the design, different solutions, how code fits into overall solutions.
  - After a period of time or at a convenient point, the developers swap places.
  - **Benefits:**
  - Conversation during the process helps to quickly move the solution on.
  - Knowledge is shared as developers pair with different individuals.
  - Code is reviewed in real-time; teams that practice pair programming often eschew code reviews.
  - The practice promotes collective code ownership.

XP Practices …

- Collective ownership
  - Developers can improve any part of the code at any time.
  - This practice also avoids code becoming owned by individuals, which can lead to bottlenecks in development and poorly designed code.
- Continuous integration
  - The codebase should be integrated and automated tests run frequently.
  - Developers working locally on their machines should check-in their changes frequently, ensuring that code conflicts are identified and resolved quickly.
- 10-Minute build
  - Build, Deploy and Test - all in 10 minutes
  - Build Server/Integration server – Builds the code automatically - pull the code from the source control system and compiles the integrated code, then deploy the code on a test/stage environment and run automated tests) Example: Jenkins integration server
  - Continuous integration is a practice of integrating the code several times a day
  - Having a build server/integration server alone is not continuous integration

XP Practices …

- Forty-hour week (Energized work)
  - Teams aim for sustainable phase
  - XP does not forbid overtime, but it has a clear rule – *You can't work a second week of overtime.*
- On-site customer
  - A real customer should sit with the team.
  - This person will be someone who will use the system, who has the knowledge and authority to answer questions and who can provide business related clarification so that issues don't block the progress of the iteration.
- Coding standards
  - A common coding standard, agreed by all developers, must be adopted across the team.
- Informative workspace (Information Radiator)
  - Visual board, Managers can assess status and see what people are working on by simply walking through the team area.

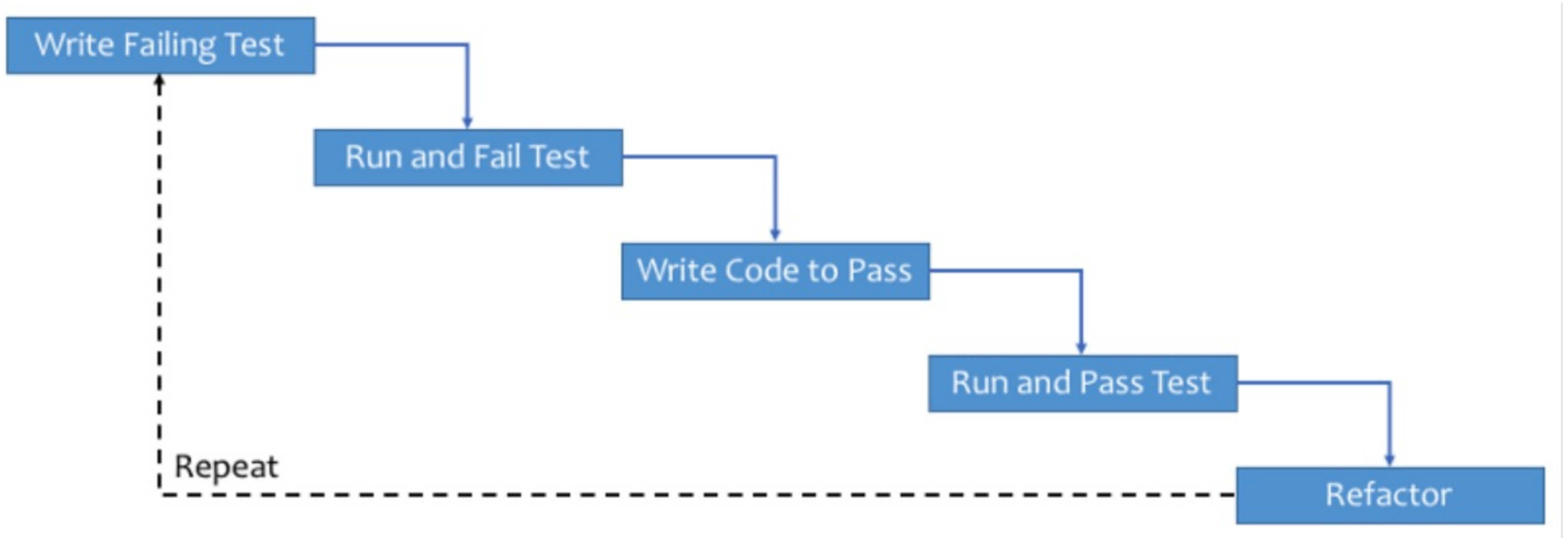
innovate achieve lead

BITS Pilani
Pilani Campus

Test Driven Development (TDD)

Test Driven Development- General work flow

- A Process where the Developer takes responsibility of the Quality of their code
- Unit tests are written before the production code.
- Don’t write all tests at once
- Tests and Production code are written in small bits of functionality
- TDD is a XP process and created by Ken Buck.



https://www.freecodecamp.org/news/learning-to-test-with-python-997ace2d8abe/

innovate achieve lead

BITS Pilani
Pilani Campus

Lean Software Development

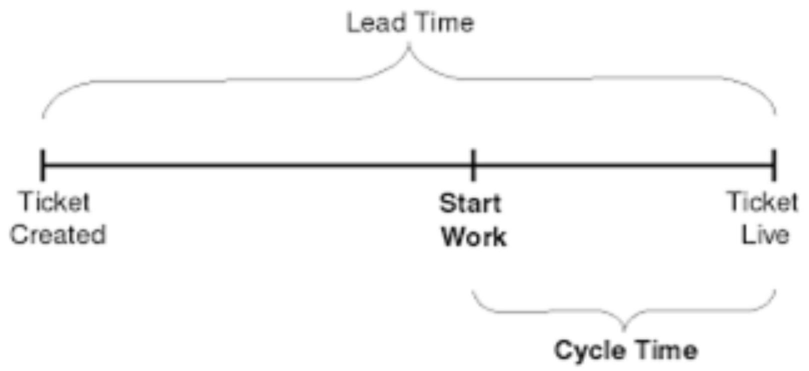
Ref: Lynda.com :DevOps Foundations: Lean and Agile with Ernest Mueller and Karthik Gaekwad

What is Lean?

- Lean is a systematic method to eliminate waste and maximize the flow of value through a system. Value is defined as something your customer will pay money for.
- Lean employs something called Value stream mapping.
- This practice is widely used in Manufacturing world.

Lead time & Cycle Time

- Lead time tracks the total amount of time it takes from when work is requested until it's delivered.
- Cycle time tracks the amount of time we spend working on it. (Also called Processing time or Throughput time)



Value Stream Mapping

- Value is defined as something your customer will pay money for.
- Value Stream Mapping practice generates a diagram that shows the exact places where value is created in your system and how it flows through your organization.

BITS Pilani
Pilani Campus

7 Principles of Lean Software Development

1. Eliminate Waste

Type of waste in SW Development

- **Unnecessary code or functionality:** Delays time to customer, slows down feedback loops
- **Starting more than can be completed:** Adds unnecessary complexity to the system, results in context-switching, handoff delays, and other impediments to flow
- **Delay in the software development process**: Delays time to customer, slows down feedback loops
- **Unclear or constantly changing requirements:** Results in rework, frustration, quality issues, lack of focus
- **Bureaucracy:** Delays speed
- **Slow or ineffective communication:** Results in delays, frustrations, and poor communication to stakeholders which can impact IT’s reputation in the organization

2. Build Quality In

- **Pair programming:** Avoid quality issues by combining the skills and experience of two developers instead of one
- **Test-driven development**: Writing criteria for code before writing the code to ensure it meets business requirements
- **Incremental development** and constant feedback
- **Minimize wait states:** Reduce context switching, knowledge gaps, and lack of focus
- **Automation:** Automate any tedious, manual process or any process prone to human error

3. Create Knowledge

- Pair Programming
- Code reviews
- Documentation
- Wiki – to let the knowledge base build up incrementally
- Chat, Chatops
- Thoroughly commented code
- Knowledge sharing sessions
- Training
- Use tools to manage requirements or user stories

4. Defer Commitment

- Don’t make decision/commit if you can differ it at later point in time. Keep options open.

- Continuously collect and analyze the data or information

5. Deliver Fast

- Build a simple solution.
- Put it in front of customers
- Enhance incrementally based on customer feedback.
- Speed to market is an incredible competitive advantage esp. for Software.
- What slows them down?
  - Thinking too far in advance about future requirements
  - Blockers that aren't responded to with urgency
  - Over-engineering solutions and business requirements

6. Respect People

- Communicating proactively and effectively
- Encouraging healthy conflict
- Surfacing any work-related issues as a team (Blameless postmortem)
- Empowering each other to do their best work.

7.Optimize the Whole

- Value Stream mapping
- System thinking
- Operating with a better understanding of capacity and the downstream impact of work.

Lean Principles that are Proven to Work

- Small deliverables
- Limiting work in progress
- Information radiators and visibility into flow,
- Gathering, broadcasting and implementing customer feedback
- Empowered development teams who are free to experiment and improve.

innovate achieve lead

BITS Pilani
Pilani Campus

Kanban

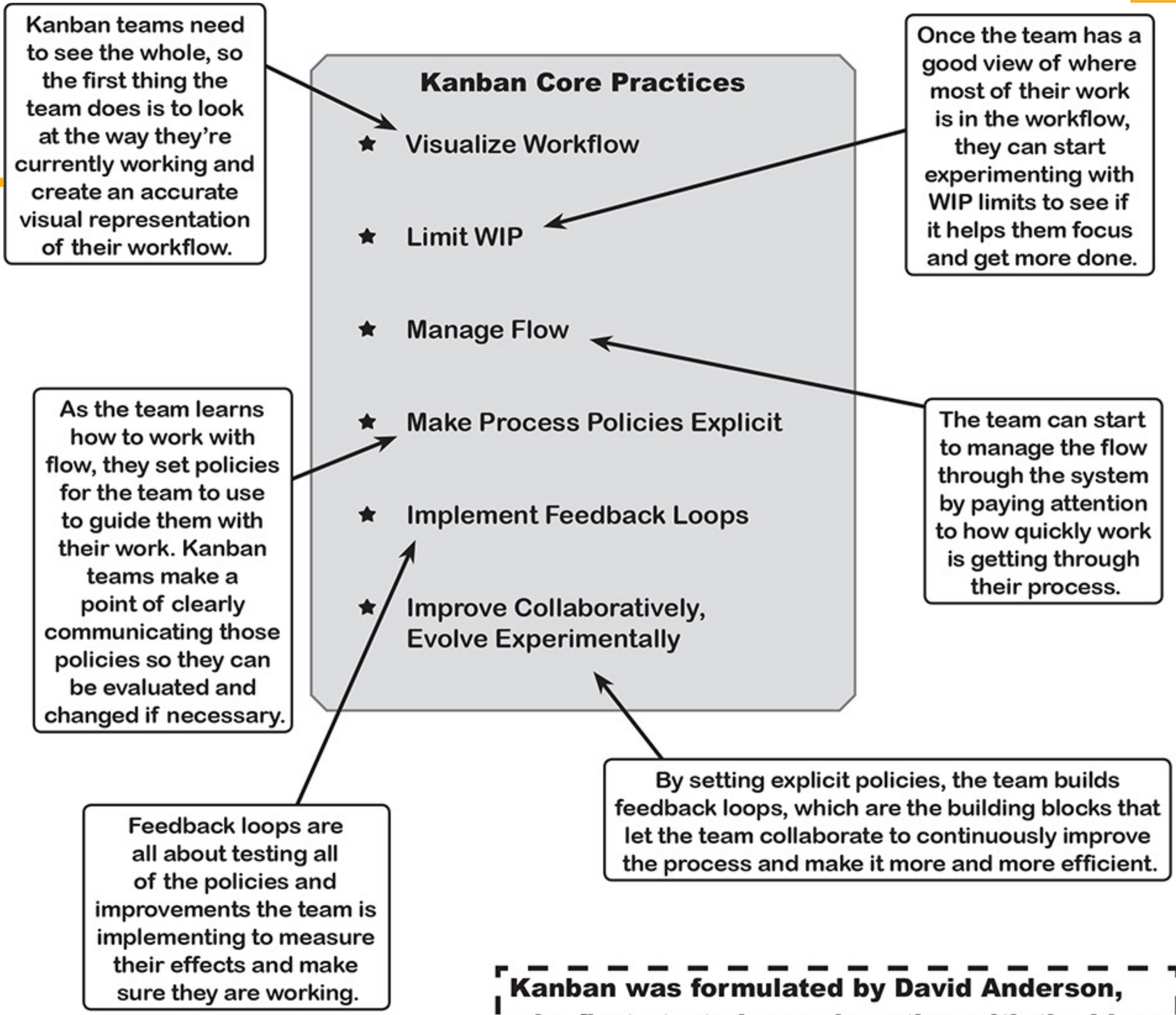
Kanban

- Taiichi Ohno, who was an industrial engineer at Toyota, developed the Kanban methodology in 2004 to improve efficiency at the manufacturing plant.

- The Kanban method is an approach to continuously improving service delivery that emphasizes the smooth, fast flow of work.

- Kanban is not an Agile software development method (or process) or a software engineering methodology
- Kanban is flow based methodology and a pull system.

- Kanban does not prescribe specific roles or process steps as it is built on the concept of evolutionary change.



Kanban Core Practices

- Visualize Workflow
- Limit WIP
- Manage Flow
- Make Process Policies Explicit
- Implement Feedback Loops
- Improve Collaboratively, Evolve Experimentally

Kanban teams need to see the whole, so the first thing the team does is to look at the way they're currently working and create an accurate visual representation of their workflow.

Once the team has a good view of where most of their work is in the workflow, they can start experimenting with WIP limits to see if it helps them focus and get more done.

The team can start to manage the flow through the system by paying attention to how quickly work is getting through their process.

As the team learns how to work with flow, they set policies for the team to use to guide them with their work. Kanban teams make a point of clearly communicating those policies so they can be evaluated and changed if necessary.

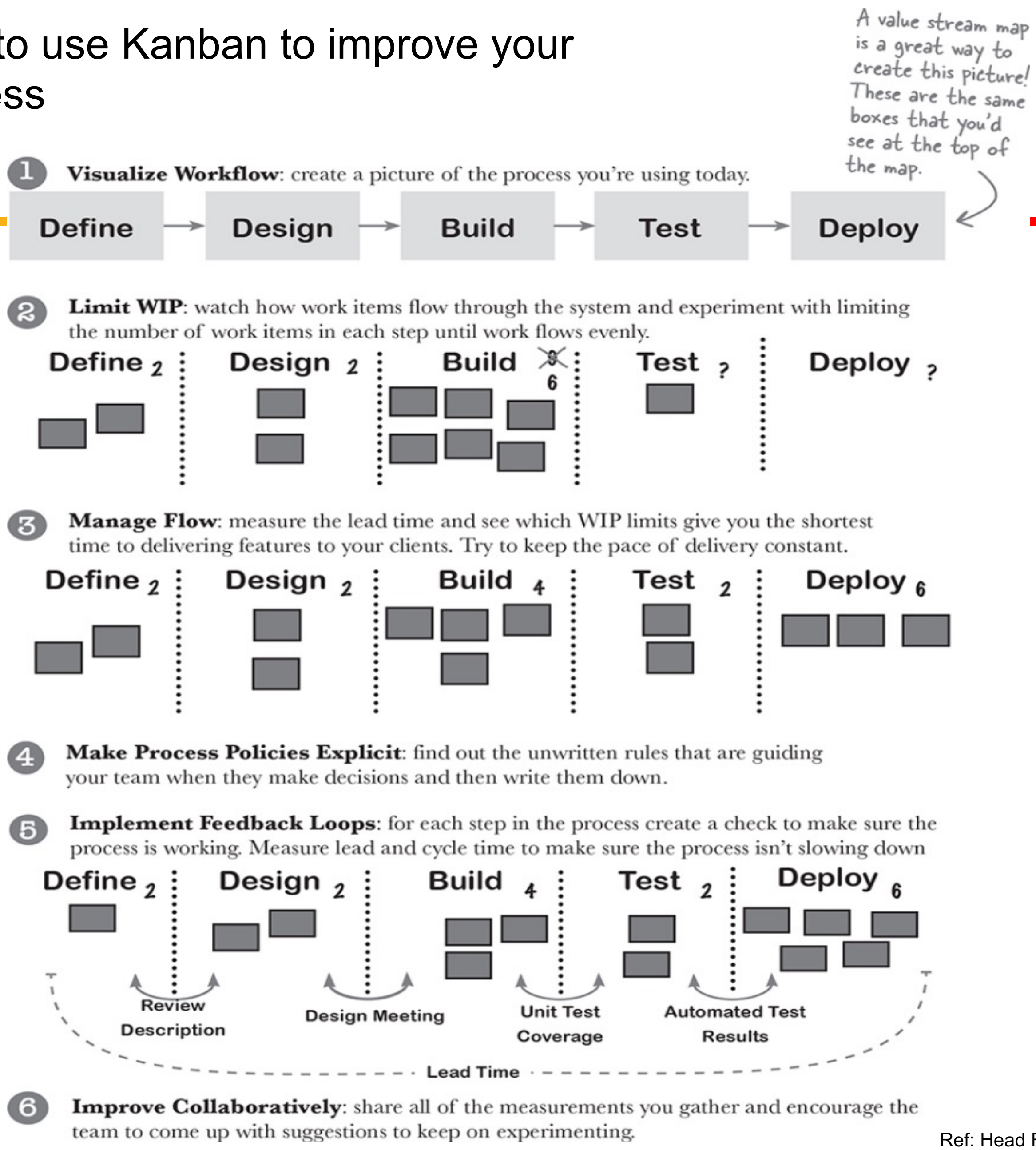
Feedback loops are all about testing all of the policies and improvements the team is implementing to measure their effects and make sure they are working.

By setting explicit policies, the team builds feedback loops, which are the building blocks that let the team collaborate to continuously improve the process and make it more and more efficient.

Kanban was formulated by David Anderson, who first started experimenting with the ideas of Lean while working at Microsoft and Corbis.

Source:Head First Agileby Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

How to use Kanban to improve your process



A value stream map is a great way to create this picture! These are the same boxes that you'd see at the top of the map.

1 **Visualize Workflow**: create a picture of the process you're using today.

2 **Limit WIP**: watch how work items flow through the system and experiment with limiting the number of work items in each step until work flows evenly.

3 **Manage Flow**: measure the lead time and see which WIP limits give you the shortest time to delivering features to your clients. Try to keep the pace of delivery constant.

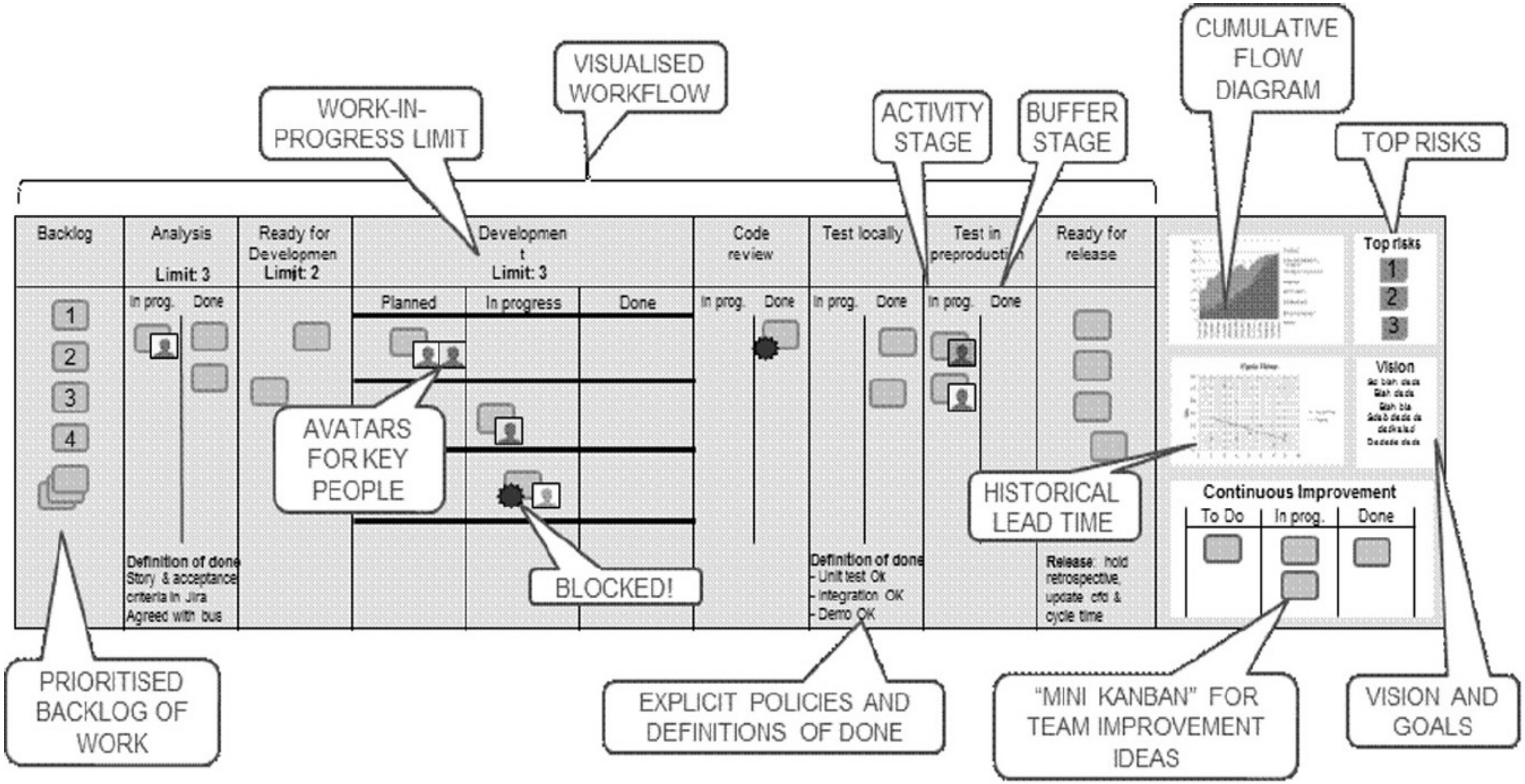
4 **Make Process Policies Explicit**: find out the unwritten rules that are guiding your team when they make decisions and then write them down.

5 **Implement Feedback Loops**: for each step in the process create a check to make sure the process is working. Measure lead and cycle time to make sure the process isn't slowing down

6 **Improve Collaboratively**: share all of the measurements you gather and encourage the team to come up with suggestions to keep on experimenting.

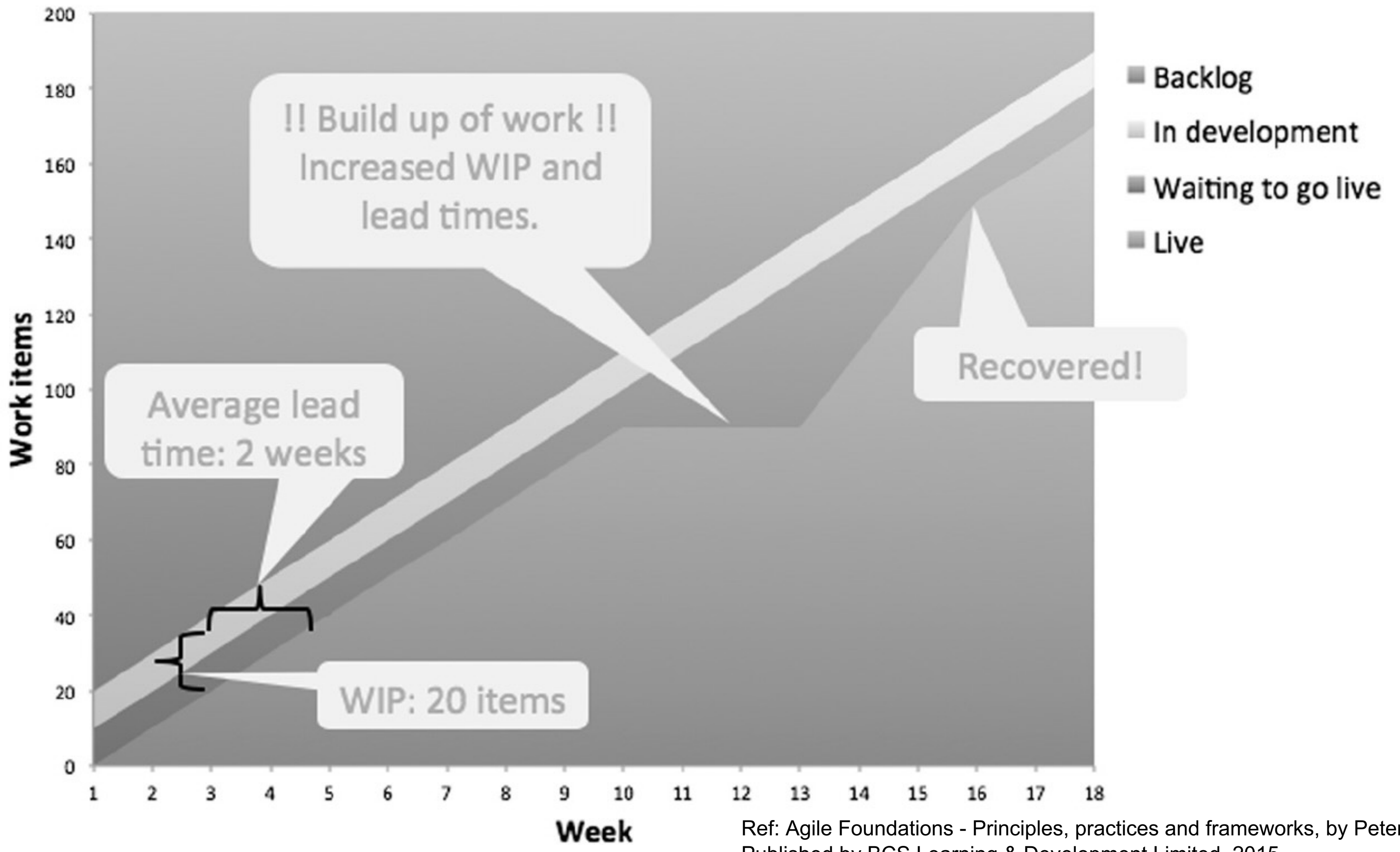
Ref: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

Typical Kanban board


VISUALISED WORKFLOW
WORK-IN-PROGRESS LIMIT
ACTIVITY STAGE
BUFFER STAGE
CUMULATIVE FLOW DIAGRAM
TOP RISKS
Backlog
Analysis Limit: 3
Ready for Development Limit: 2
Development Limit: 3
Code review
Test locally
Test in preproduction
Ready for release
In prog. Done
Planned In progress Done
Top risks 1 2 3
Vision
Continuous Improvement To Do In prog. Done
AVATARS FOR KEY PEOPLE
BLOCKED!
HISTORICAL LEAD TIME
Definition of done Story & acceptance criteria in Jira Agreed with bus
Definition of done - Unit test Ok - Integration OK - Demo OK
Release: hold retrospective, update cfd & cycle time
PRIORITISED BACKLOG OF WORK
EXPLICIT POLICIES AND DEFINITIONS OF DONE
"MINI KANBAN" FOR TEAM IMPROVEMENT IDEAS
VISION AND GOALS

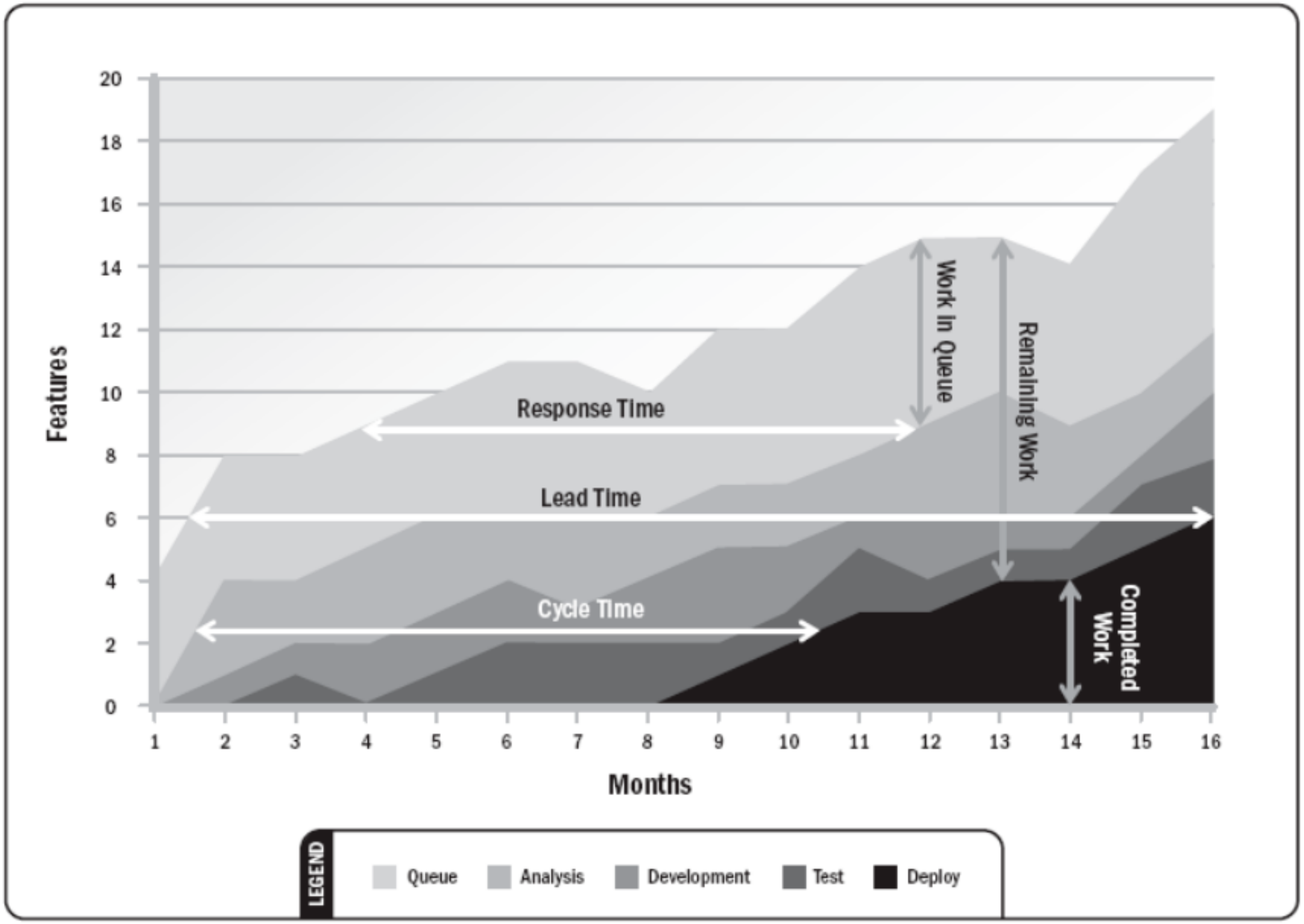
Ref: Agile Foundations - Principles, practices and frameworks, by Peter Measey
Published by BCS Learning & Development Limited, 2015

Cumulative flow diagram (Example-2)



Ref: Agile Foundations - Principles, practices and frameworks, by Peter Measey
Published by BCS Learning & Development Limited, 2015

Cumulative flow diagram (Example-3)



BITS Pilani

Pilani Campus

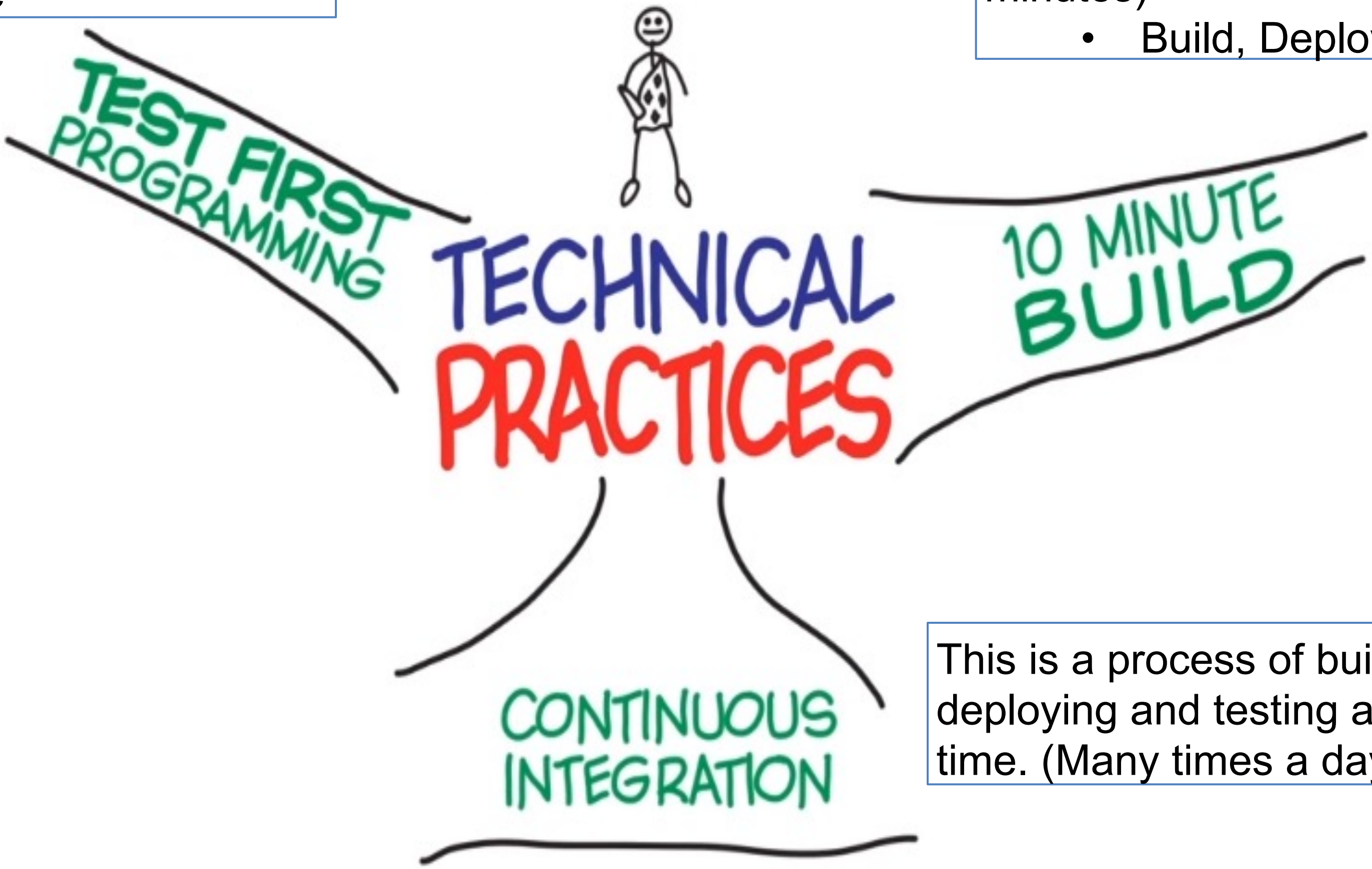
Module-4 XP-Technical Practices

XP-Technical Practices

TDD: Write tests before you write the code

Find any new problem as quickly as possible.(within 10 minutes)

- Build, Deploy, Test



This is a process of building, deploying and testing all the time. (Many times a day)

Source: The Agile Sketchpad Dawn Griffiths David Griffiths

TDD Helps You Avoid Scope Creep

Scope Creep:

•One common reason why scope creep occurs is lack of documentation with clearly defined requirements.

•This problem can be mitigated through test driven development.

•In a TDD environment, developers write unit tests to test particular segments – units – of code. Unit tests serve as specifications that describe the precise features that should be implemented.

•Therefore, well-specified tests prevent developers from writing superfluous code.

•TDD helps developers focus on what’s necessary and prevents gold plating – adding unnecessary or unwanted features that weren’t specified in the project requirements.

TDD can serve as living documentation

- Code comments can get out of date , but the automated tests should be up-to-date, otherwise the tests will break the code. By looking at the test we can infer what the code supposed to do
- Tests can serve as documentation to a developer. If you're unsure of how a class or library works, go and have a read through the tests.

TDD Can Lead to Better Design

- A good test suite allows you to refactor, which allows you to improve your design over time.
- The TDD cycle is very detail-oriented and requires you to make some design decisions when writing tests, rather than when writing production code. I find this helps to think about design issues more deeply.
- TDD makes some design problems more obvious.
- Using automated tests, code needs to work with the rest of the application and also with unit tests. The chances are the code can be reused elsewhere in your system.
- If you have difficulty in writing tests for the code, the changes are the code need to redesign.

TDD helps in Drive Progress

- One of the fundamental ideas behind the concept of test-first development is to let the tests show you what to implement in order to make progress on developing the software.
- You’re not just coding away, oblivious to the requirements of the piece of code you’re writing. You’re satisfying an explicit, unambiguous requirement expressed by a test. You’re making progress, and you’ve got a passing test to show for it.
- Forced to make the code to perform something useful because the tests are passed.

Unit and Acceptance Tests

- **Unit tests** are written by programmers to ensure that the code does what they intend it to do.
- **Acceptance tests** are written by business people (and QA) to make sure the code does what they intend it to do

10-Minute Build

- One of the practices recommended by Extreme Programming (XP) is to keep a ten-minute build. Kent Beck and Cynthia Andres write in Extreme Programming Explained (Second Edition):
- "Automatically build the whole system, Deploy and run all of the tests in ten minutes. A build that takes longer than ten minutes will be used much less often, missing the opportunity for feedback."

Continuous Integration

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.
- By integrating regularly, you can detect errors quickly, and locate them more easily

BITS Pilani

Pilani Campus

Thank you