



BITS Pilani

Pilani Campus



BITS Pilani presentation

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



SE ZG544 S1-22-23 , Agile Software Processes
SE ZG544 S1-22-23
Lecture No. 1, Module-1 - Agile Methods - An Introduction

Introduction

1. Faculty introduction
2. Email Id : kanantharaman@wilp.bits-pilani.ac.in
3. e-learn portal: <https://elearn.bits-pilani.ac.in/>
4. [Course Handout](#)
5. Recorded Video Lectures in e-learn/Taxila portal
 - According to the course handout, grouped by module
 - You MUST go through each module before coming to the online session

Poll

-
- <https://forms.gle/wRadsyQREA3BpkE26>

Module-1 – Topics

- Traditional software development practices
- Need for Agile Methods
- Benefits of Agile Methods

Basic Project Management concepts



- What is a Project?
 - Definite Start-End date, Temporary, Scope(Produce Specific result) , Budget/Effort – Example: Building a house
- Project Management Life Cycle Phases
 - Initiation, Planning, Execution, Closeout, Monitoring & Control
- System Development Life Cycle/phases (SDLC)
 - Requirements, Design, Construction, Implementation

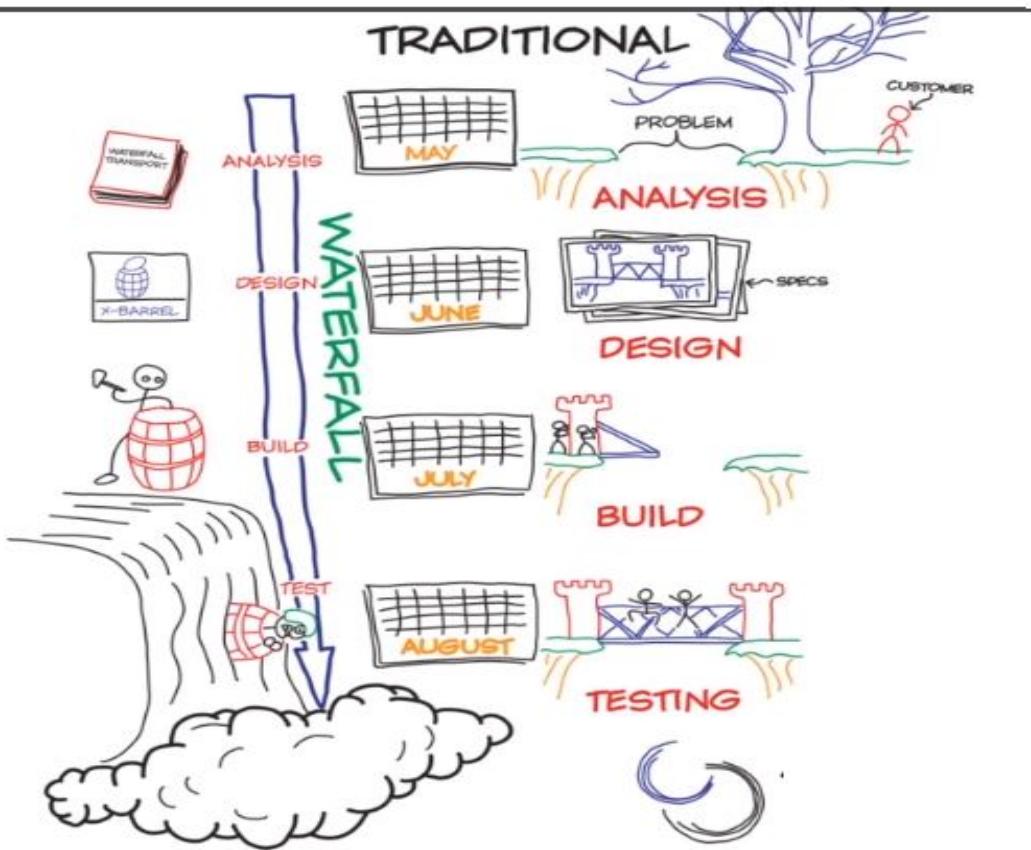
Questions?

- Q1,Q1_1,Q1_2
- <https://forms.gle/onYWuBBy8TAJ6QVAA>
- <https://forms.gle/oC9BhYDVc2EvsD5N9>
- <https://forms.gle/pocBLb1fA7RjdYYU7>



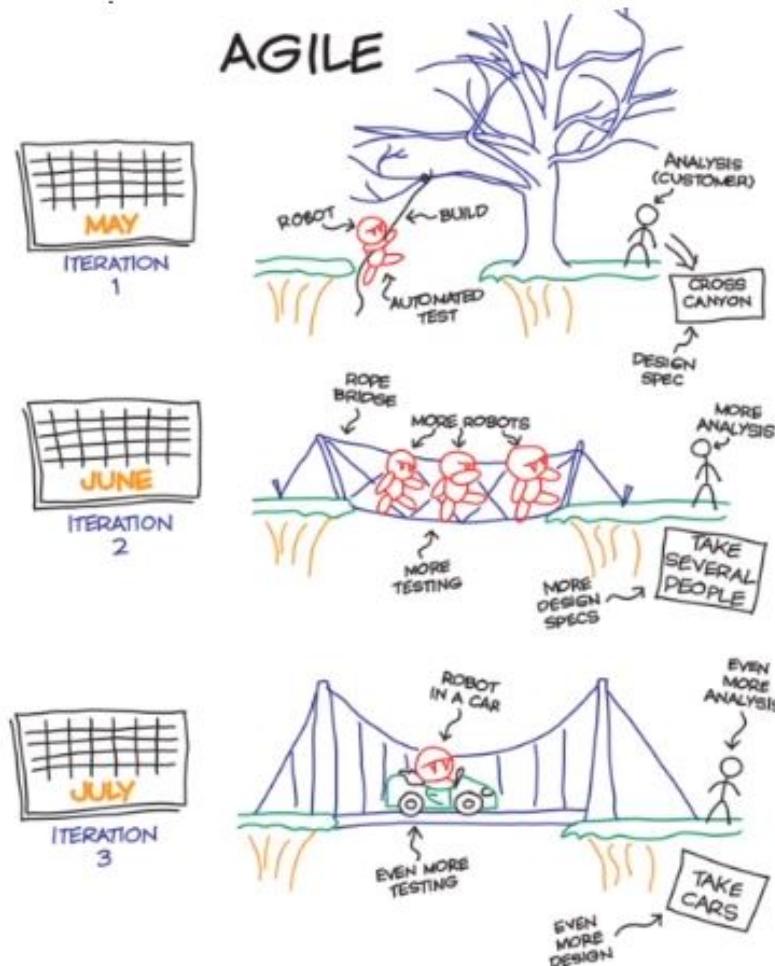
Project Management Model Water Fall Model and Agile

Traditional /Waterfall Development Approach(Rigid)



Reference : The Agile Sketchpad By [Dawn Griffiths](#), [David Griffiths](#), O'reilly media

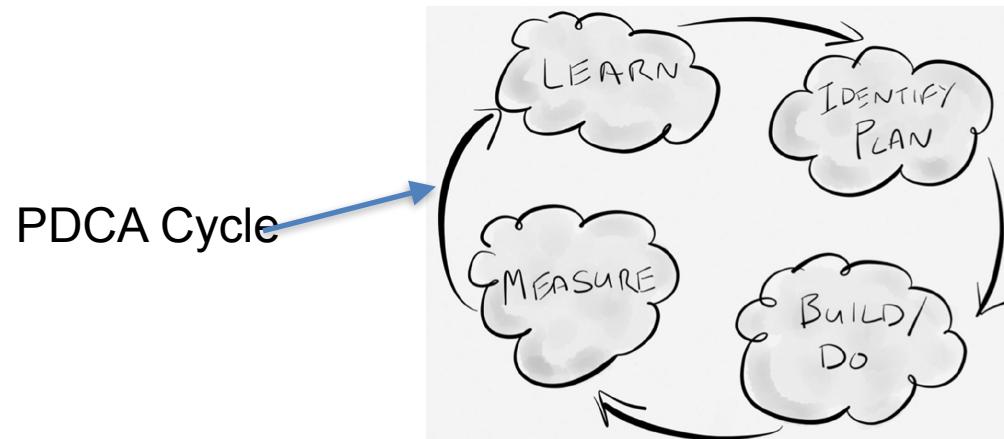
Agile Software Development Approach (Empirical)



Reference . The Agile Sketchpad By [Dawn Griffiths](#), [David Griffiths](#), O'reilly media

Empirical Process Control

- **Inspection**
 - inspect the product being created and how it is being created
- **Adaption**
 - adapt the product being created or the creation process if required
- **Transparency**
 - ensure everyone can easily see what is happening



Questions?

- Q2, Q3
- <https://forms.gle/gaqQUVnLeB1uoCpT9>
- <https://forms.gle/pKRRh3cFn6xCJrdj6>

Advantages and Disadvantages of Waterfall

Advantages:

- Sequential, Upfront planning

Disadvantages:

- Error propagation

- Good Documentation

- Missing requirements

- Scope of work is generally fixed

- Error correction is costly

- Late customer feedback

Advantages and Disadvantages of Agile Model



Advantages:

- Early delivery of business value
- Continuous improvement
- Scope flexibility
- Team input
- Delivering well-tested products

Disadvantages:

- Poor Resource planning
- Less Documentation
- Fragmented output

Application of Waterfall Model



- Most common Project Management approach
- Surpassed by Agile approach after 2008.
- Simple and small systems.
- Enhancements to software systems
- Mission critical systems.

Application of Waterfall and Agile Model



- Fast Changing deliverables - New Technology Emerging projects
- Projects without clear requirements in the beginning
- New Product Development Projects
- Early Visibility, Quality, Risk identification



Need for Agile Methods

Software Project Success and Failure



- In 2015, Standish Group did a study of 10,000 projects in USA. The results showed that:
- 29% of traditional projects failed outright
- 60 percent of traditional projects exceeded the budget
- 11 percent of projects succeeded.

Questions?

- Q4, Q5
- <https://forms.gle/hNGMkyCTuXdXHTP86>
- <https://forms.gle/N2diLqfF994mi7ZR7\>



Benefits of Agile Methods

Corporate World - Challenges and Inefficiencies



- Missed (or rushed) deadlines.
- Budget blow-outs
- Overworked and stressed employees.
- Knowledge silos.
- Technology innovations and Agile approaches that have enabled to overcome these challenges (IT and Manufacturing industries)

Benefits of Agile Methods/Approaches/ Practices/Techniques

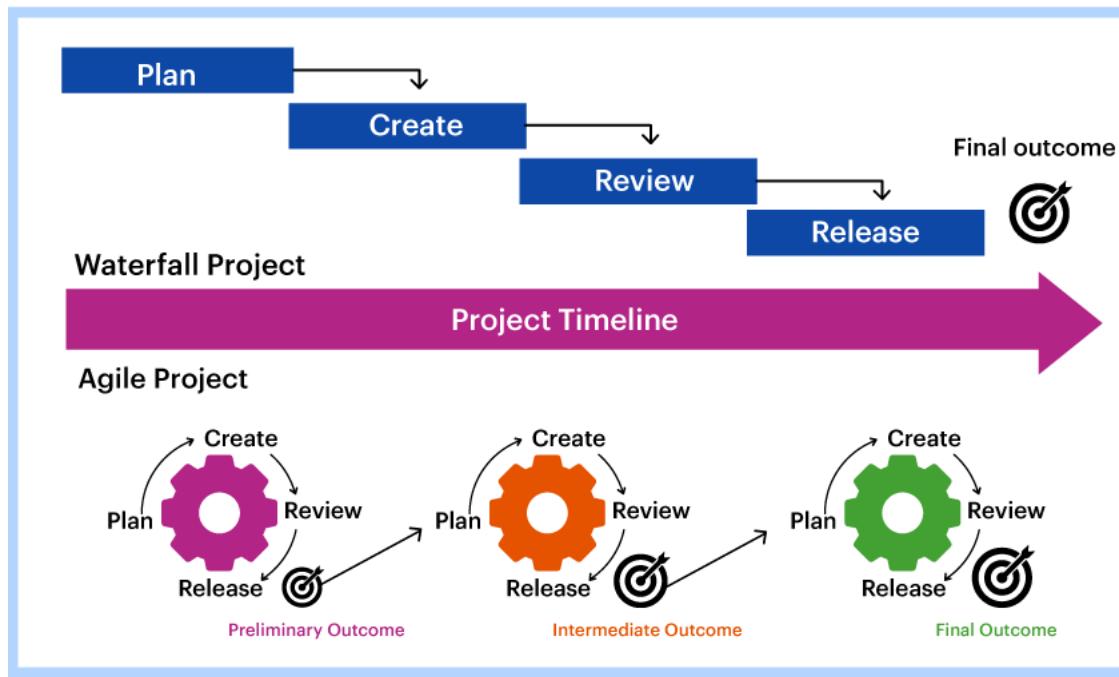
- Responsive planning
- Business-value-driven work
- Hands-on business outputs
- Direct stakeholder engagement
- Immovable deadlines
- Management by self-motivation
- 'Just-in-time' communication
- Immediate status tracking
- Waste management
- Constantly measurable quality
- Continuous improvement



Module-1-Additional Notes

Topics Module-1

- Traditional software development practices
- Need for Agile Methods
- Benefits of Agile Methods



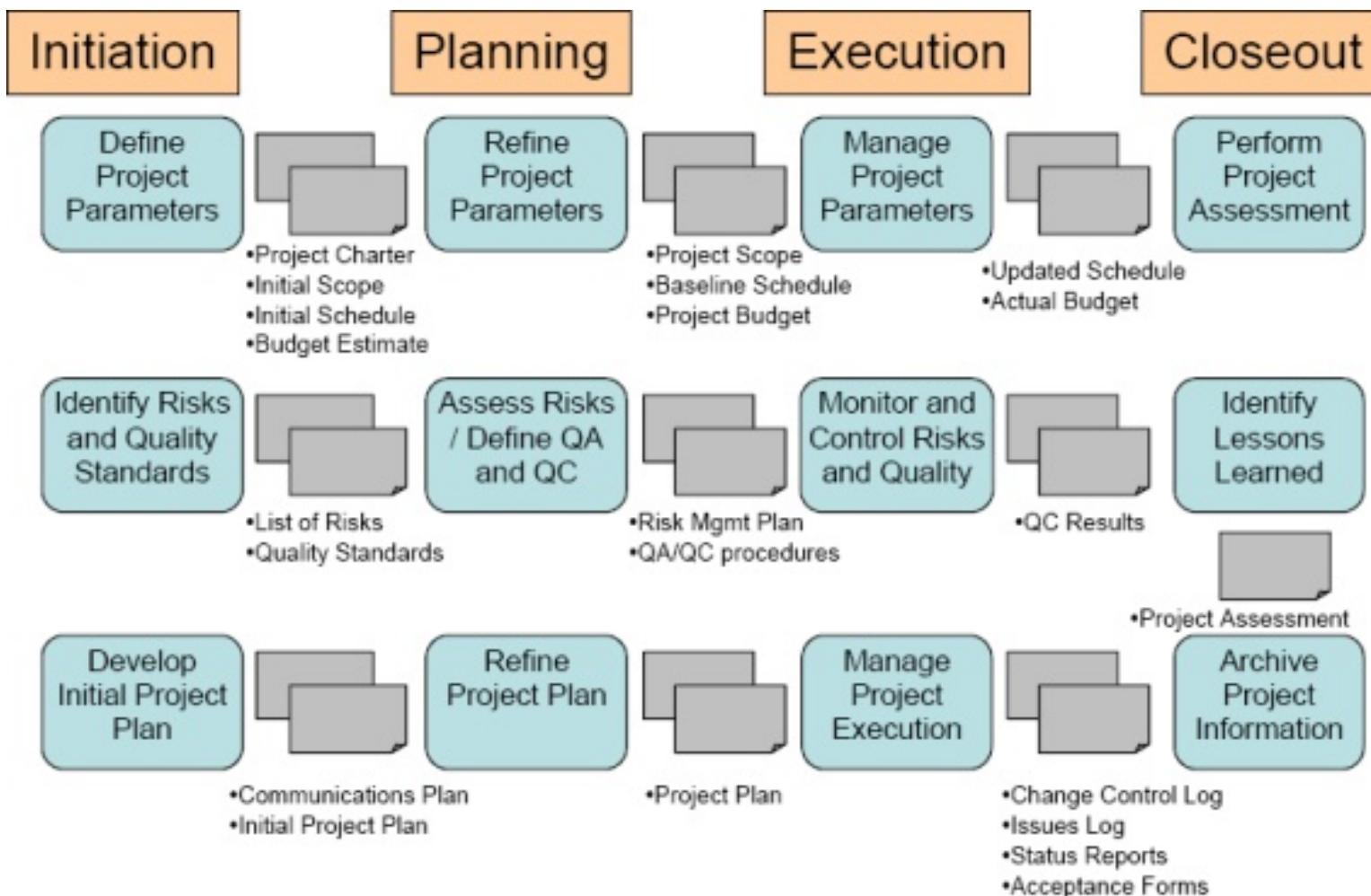
<https://kissflow.com/project/agile/traditional-vs-agile-project-management/>

What is a Project?

- A project is a planned program of work that requires a **definitive amount of time, effort, and planning** to complete.
- Projects have **goals and objectives** and often must be completed in **some fixed period of time and within a certain budget**.

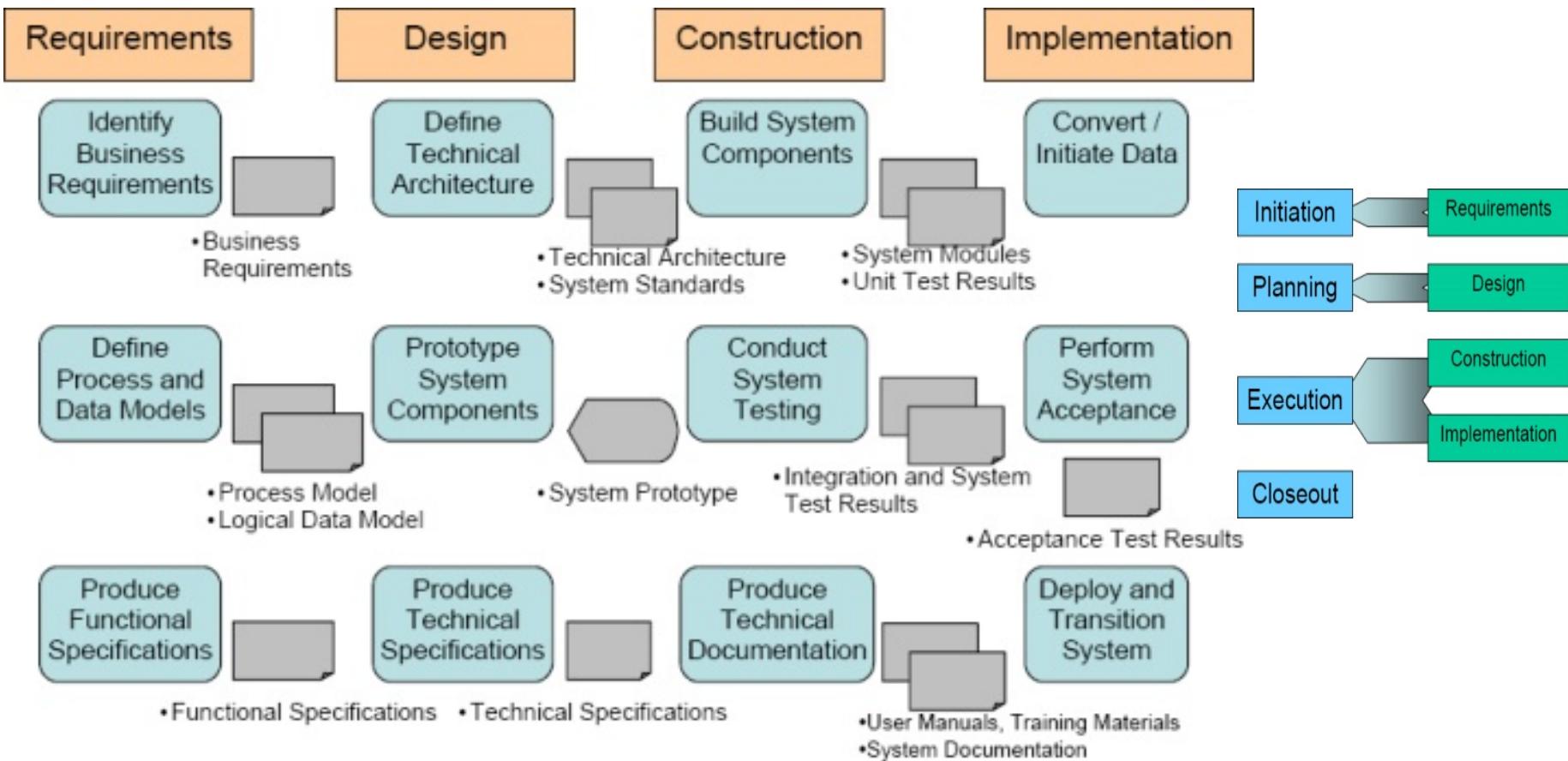
- Development Project
- Maintenance or Support Project (Operational work)

Project Management Phases



<https://www.pmi.org/learning/library/project-managing-sdlc-8232>

System Development Phases (Engineering activities)



Traditional Software Development Approaches

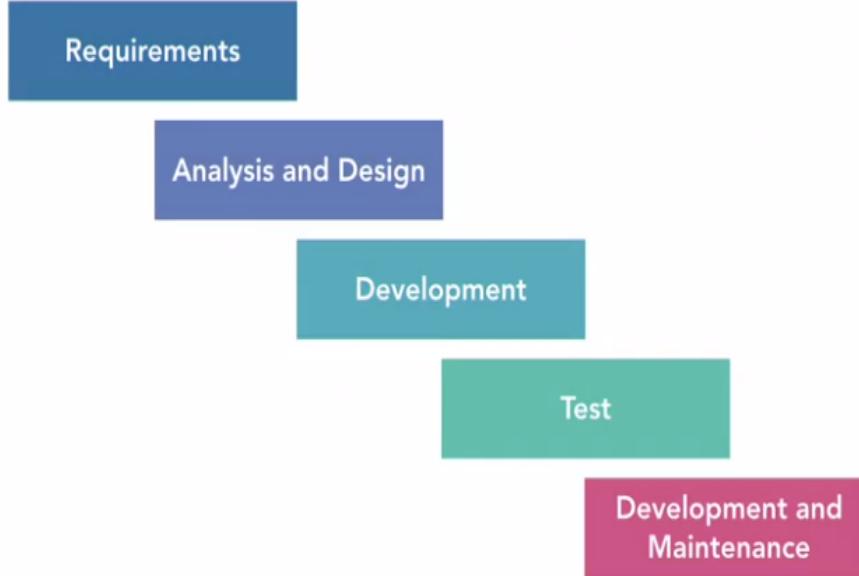
BITS Pilani

Pilani Campus



Traditional Software Development Model – Waterfall Model

Waterfall Approach

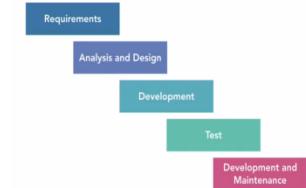


- Move to the next phase only when the prior one is complete — hence, the name waterfall.
- Origin from manufacturing like production plant
- **Upfront Planning**
- **Detailed documentation**
- **Scope of work is generally fixed.**
- Output of a phase becomes input to next phase
- Include well defined checklists, process and tools

<https://www.lynda.com/Developer-tutorials/Software-Development-Life-Cycle-SDLC/5030981-2.html>

Issues with Waterfall approach

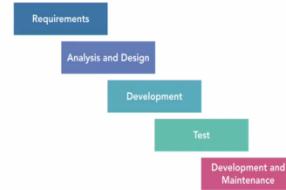
- **Error in one phase will propagate to next phase**
- **Missing requirements will result in missing software feature**
- **Error correction is costly** if it is detected at later phase
- **Customer does not get to see the product** before the early testing phase which is usually two-thirds the way through the product time line.



<https://www.lynda.com/Developer-tutorials/Software-Development-Life-Cycle-SDLC/5030981-2.html>

Issues with Waterfall approach ...

- You could be in the Deployment and Maintenance phase when you could realize that the product you are building was **no longer viable** due to **change in market conditions, or organizational direction**, or changed computer landscape
- (OR) You could realize that the product had a major **architectural flaw** that prevented it from being deployed.
- In other words, your product development initiative **could completely fail** after a lot of money and time had been spent on it.



<https://www.lynda.com/Developer-tutorials/Software-Development-Life-Cycle-SDLC/5030981-2.html>

Impact of Waterfall

- **Project failures**
 - Many organizations treated this failure as if there was a failure in a production factory. So they tried to fix their waterfall approach, by adding more comprehensive documentation.
 - **Comprehensive documentation**
 - Having a well documented software system is good. But the documentation by itself adds no value to the stake holders.
 - **Checklists and Coding standards**
 - Many software teams resorted to maintaining comprehensive checklist, to make sure they were producing systems of high quality. Checklist such as coding standards and architectural reviews are helpful. But you cannot produce a single recipe book for building software
- More time should be spent on delivering working software features early and often. And enlisting customer feedback

<https://www.lynda.com/Developer-tutorials/Software-Development-Life-Cycle-SDLC/5030981-2.html>

Application of Waterfall Model



- **Simple and small systems.**
 - **Enhancements to software systems** — specifically applicable if the development team has good domain knowledge.
 - **Mission critical systems.** Where you need gated checks to avoid catastrophic failures. An example is a software system where a defect can cause human causality. Comprehensive documentation is also very applicable here.
- *Waterfall model is the **most common project management approach** in software development until it was surpassed by improved approaches based on agile techniques around 2008.*

Application of Waterfall Model



- **Simple and small systems.**
 - **Enhancements to software systems** — specifically applicable if the development team has good domain knowledge.
 - **Mission critical systems.** Where you need gated checks to avoid catastrophic failures. An example is a software system where a defect can cause human causality. Comprehensive documentation is also very applicable here.
- *Waterfall model is the **most common project management approach** in software development until it was surpassed by improved approaches based on agile techniques around 2008.*

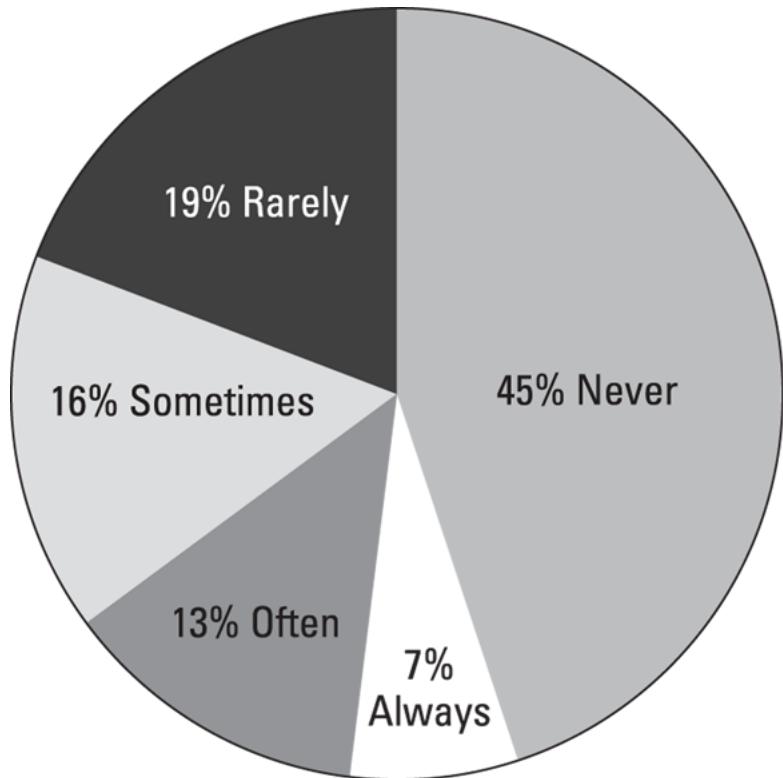
Software Project Success and Failure



- In 2015, Standish Group did a study of 10,000 projects in USA. The results showed that:
- 29% of traditional projects failed outright
 - The projects were cancelled before they finished and did not result in any product releases. These projects delivered no value whatsoever
- 60 percent of traditional projects exceeded the budget
 - The projects were completed, but they had gaps between expected and actual cost, time, quality, or a combination of these elements. The average difference between the expected and actual project results — looking at time, cost, and features not delivered — was well over 100 percent.
- 11 percent of projects succeeded.
 - The projects were completed and delivered the expected product in the originally expected time and budget.

The problem with Status Quo

- Traditional projects that do succeed often suffer from scope bloat.



- The numbers in Figure illustrate an enormous waste of time and money.
- Direct result of traditional project management processes that are unable to accommodate change.
- Project managers and stakeholders at the start of a project ask for :
 - Everything they need
 - Everything they think they may need,
 - Everything they want,
 - Everything they think they may want

Actual use of requested software features.

Project management Needed Makeover



- In software development, **everything changes**. Requirements, skills, people, environment, business rules, et cetera.
- As **time progresses, you learn better** techniques of doing things.
- Your stakeholders need to change requirements to match changing **organizational strategy or Technology trends or changing market conditions**.
- In other words, the only **guaranteed thing is change** and the shown process to refine our work.
- Software development is **inherently an iterative process** and does not work like a Waterfall cycle.
- **Over emphasis on checklists and controls does not help** because software development is human centric and is heavily dependent on judgment and creativity.
- Software is not a product designed to be built by assembly lines.



Need for Agile Methods

Software Project Success and Failure using Traditional Approach

- In 2015, Standish Group did a study of 10,000 projects in USA. The results showed that:
- 29% of traditional projects failed outright
- 60 percent of traditional projects exceeded the budget
- 11 percent of projects succeeded.
- Also, projects that do succeed often suffer from scope bloat. – Only 20% of features is often used, 80% - Sometime/Rarely/Never used.

Project management Needed Makeover



- **In software development:**
- Everything changes.
- As time progresses, you learn better techniques of doing things.
- Organizational strategy changes or Technology trends or changing market conditions. (e.g. Covid19 Situation)
- Software development is inherently an iterative process
- Over emphasis on checklists and controls does not help.
- Software is not a product designed to be built by assembly lines.

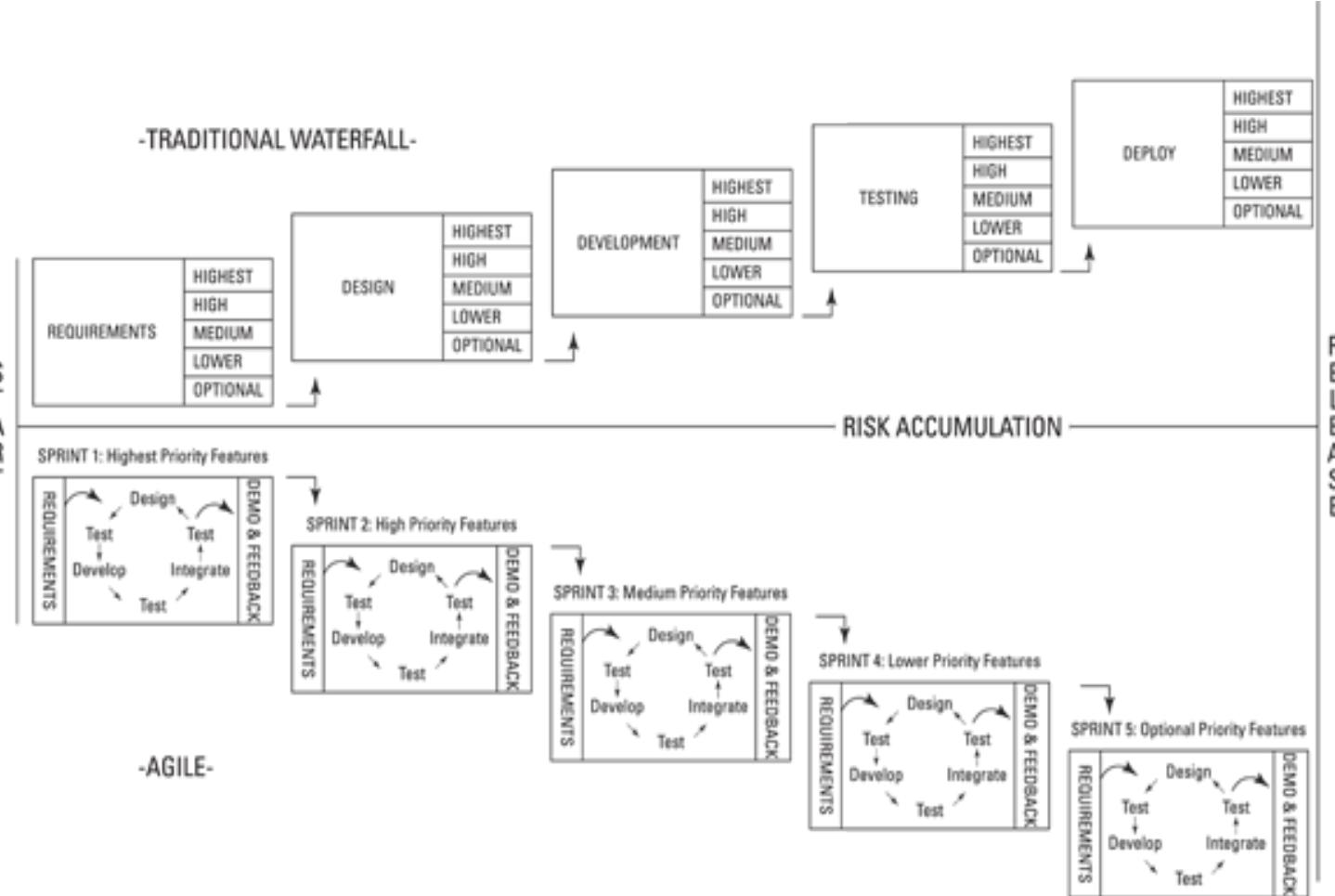
Definable Work

- Definable work projects are characterized by **clear procedures** that have proved successful on similar projects in the past.
- The production of a **car, electrical appliance, or home** after the design is complete are examples of definable work.
- The production domain and processes involved are usually **well understood** and there are typically low levels of execution uncertainty and risk.
- Definable work is **automated**.

High Uncertainty Work

- **New design, problem solving**, and not-done-before work is **exploratory**. It requires subject matter experts to collaborate and solve problems to create a solution.
 - Examples of people encountering high-uncertainty work include software systems engineers, product designers, doctors, teachers, lawyers, and many problem-solving engineers.
- High-uncertainty projects have **high rates of change, complexity, and risk**.
 - These characteristics present problems for traditional predictive approaches that aim to determine the bulk of the requirements upfront and control changes through a change request process.
- **Instead, agile approaches were created to explore feasibility in short cycles and quickly adapt based on evaluation and feedback.**

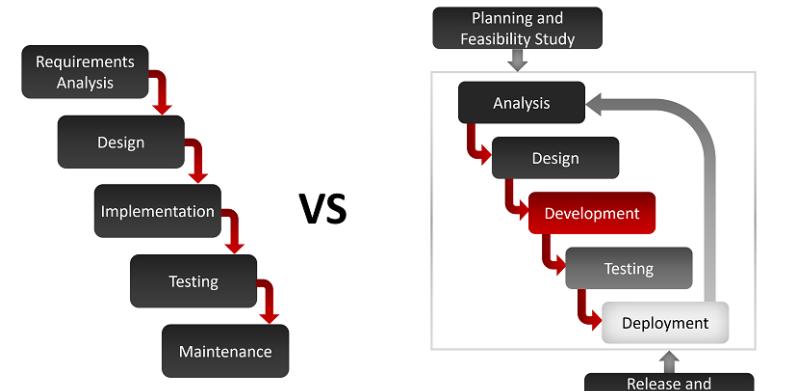
Waterfall vs agile project



Mixing traditional project management methods with agile approaches:

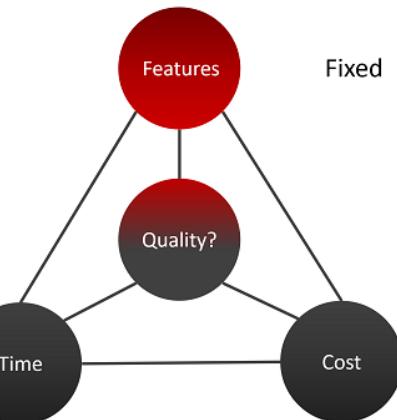
The answer, of course, is **you can't**. If you fully commit to an agile approach, you will have a better chance of deriving benefits of Agile project

Summary: Difference between Traditional and Agile Project Management

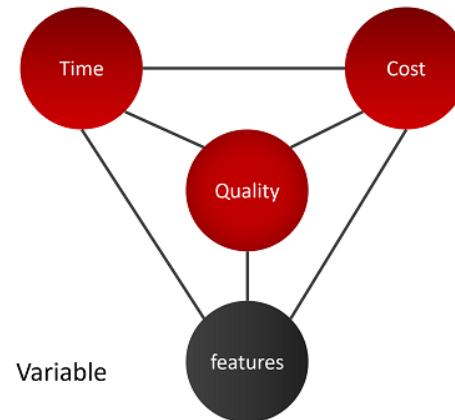


Traditional Approach

Agile Approach



Fixed



Variable

1. Flexibility (Rigid Vs Adaptive)
2. Ownership & Transparency (Project Manager vs Team ownership)
3. Problem Solving (Unexpected obstacles- Escalation vs Team take decision)
4. Checkpoints and Monitoring progress: (No Frequent check-ins vs Quicker Iteration delivering value)

Ref: <https://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies#:~:text=The%20main%20difference%20between%20traditional,in%20Agile%2C%20it%20is%20iterative.>



Evolution of Agile Project Management

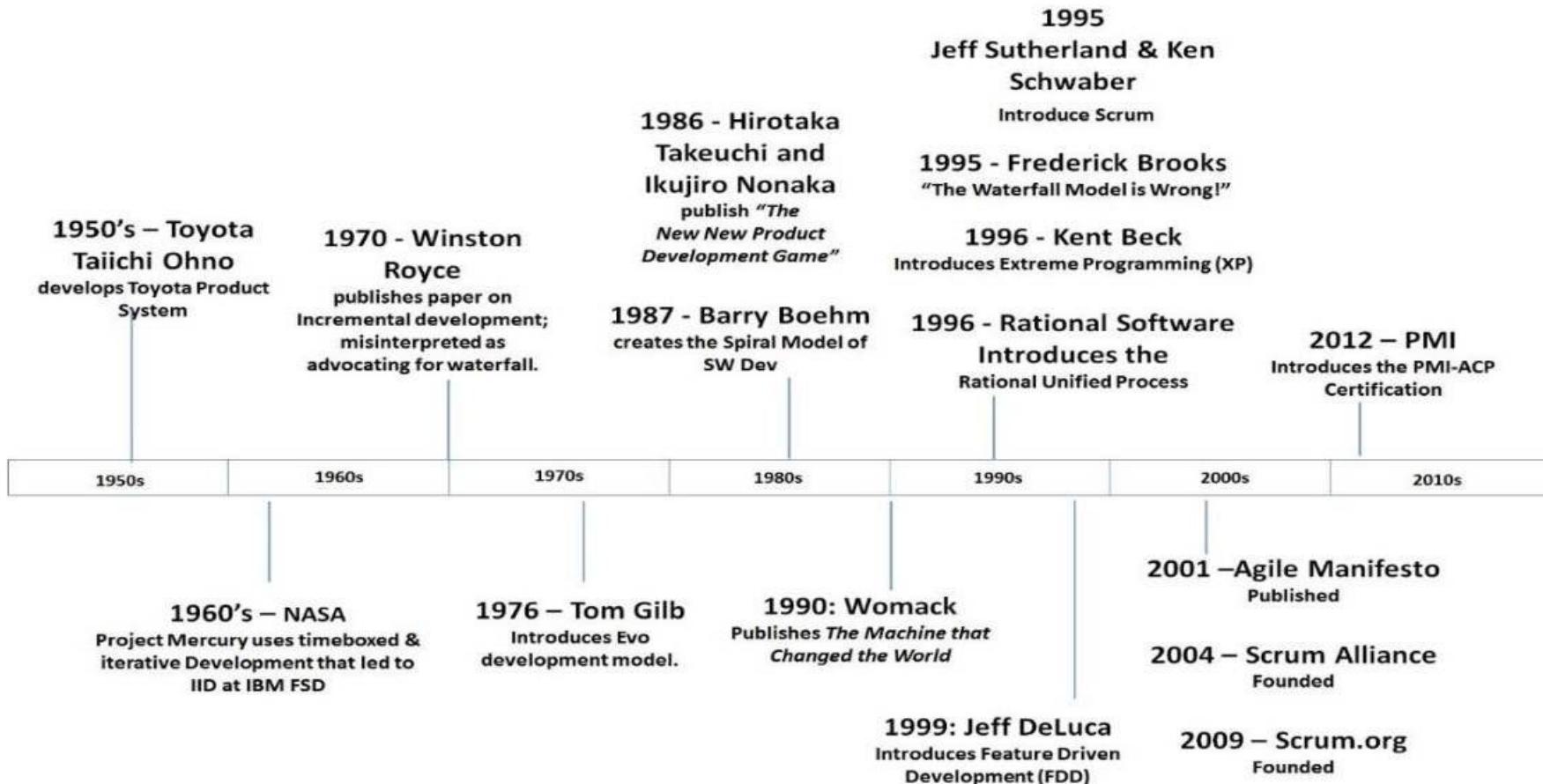
Agile Project Management

- *Agile project management* is a style of project management that focuses on :
- **Early delivery of business value**
- **Continuous improvement** of the project's product and processes
- Scope **flexibility**
- Team **input**
- Delivering well-tested products **frequently** that reflect customer needs.

Evolution of Agile Frameworks



A Brief History of Agile



Evolution of Agile Frameworks ...

- In 1986, Hirotaka Takeuchi and Ikujiro Nonaka published an article called **“New New Product Development Game” in the Harvard Business Review**.
- Takeuchi and Nonaka’s article **described a rapid, flexible development strategy** to meet **fast-paced product** demands.
- This article first paired the term **scrum** with product development. (Scrum originally referred to a player formation in **rugby**.)
- **Scrum** eventually became one of the **most popular** agile project management frameworks.

Evolution of Agile

- In 2001, a group of software and project experts got together to talk about what their successful projects had in common.
- This group created the *Agile Manifesto*, a statement of values for successful software development:
- ***We will see more details about Agile Manifesto in the next Module***



How Agile Project Work

How agile projects work

- Agile approaches are based on an ***empirical control method*** — a process of making decisions **based on the realities observed in the project**.
- In the context of software development methodologies, an empirical approach can be **effective in both new product development and enhancement and upgrade** projects.
- By using **frequent and firsthand inspection** of the work to date, you can make immediate adjustments, if necessary.

Why Agile Projects Work Better



- The Standish Group study, mentioned earlies slide “Software project success and failure,” found that while **29 percent of traditional projects failed outright, that number dropped to only 9 percent** on agile projects.
- The decrease in failure for agile projects is a result of agile project teams making **immediate adaptations** based on **frequent inspections** of progress and **customer satisfaction**.

Why Agile Projects Work Better ...



- Some key areas where agile approaches are superior to traditional project management methods:
 - **Project success rates:** The risk of catastrophic project failure falls to almost nothing on agile projects. Agile approaches of prioritizing by business value and risk ensure early success or failure. Agile approaches to testing throughout the project help ensure that you find problems early, not after spending a large amount of time and money.
 - **Scope creep:** Agile approaches accommodate changes throughout a project, minimizing scope creep. On agile projects, you can add new requirements at the beginning of each sprint without disrupting development flow. By fully developing prioritized features first, you prevent scope creep from threatening critical functionality.
 - **Inspecting and adaptation:** Agile project teams — armed with frequent feedback from complete development cycles and working, shippable functionality — can improve their processes and their products with each sprint.



Benefits & Challenges of Agile Methods

Corporate World - Challenges and Inefficiencies



- Most organizations (Small/Large/Public/Private/Startup) share the same core challenges and inefficiencies, including:
 - Missed (or rushed) deadlines.
 - Budget blow-outs
 - Overworked and stressed employees.
 - Knowledge silos.
- Technology innovations and Agile approaches that have enabled them: (IT & Manufacturing industries)
 - Genuinely create more efficient work environments, to consistently manage their work within allocated budgets, and to regularly deliver high business-value (and high-quality) outputs on time.

Benefits of Agile Methods/Approaches/ Practices/Techniques



- ***Responsive planning***: involves breaking down long-term objectives into shorter delivery cycles; and then *adapting* ongoing work (and funding) based on the outcomes of each delivery cycle.
- ***Business-value-driven work***: involves prioritizing work in accordance with the amount of primary and secondary business value that each activity is likely to bring to the organization.

Benefits of Agile Methods/Approaches/

Practices/Techniques ...

Hands-on business outputs: involves regularly inspecting outputs firsthand in order to determine whether business requirements are being met – and whether business value is being delivered for the organization.

Direct stakeholder engagement: involves actively engaging internal and external customers throughout a process to ensure that the resulting deliverables meet their expectations.

Benefits of Agile Methods/Approaches/ Practices/Techniques ...



Immovable deadlines: are fixed time commitments that encourage staff members to deliver regular ongoing value to the organization.

Management by self-motivation: involves using the power of self-organized teams to deliver outcomes under the guidance and oversight of the customer.

'Just-in-time' communication: replaces traditional corporate meetings with techniques for more effective communication and knowledge transfer (**Differ Commitment**)

Benefits of Agile Methods/Approaches/ Practices/Techniques ...



Immediate status tracking: provides tools that enable staff to keep others in the organization continuously aware of the status of the work that they are doing.

Waste management: involves maximizing the value of the organization's resources by reducing and, where possible, eliminating low business-value activities.

Constantly measurable quality: involves creating *active checkpoints* where organizations can assess outputs against both qualitative and quantitative measurements.

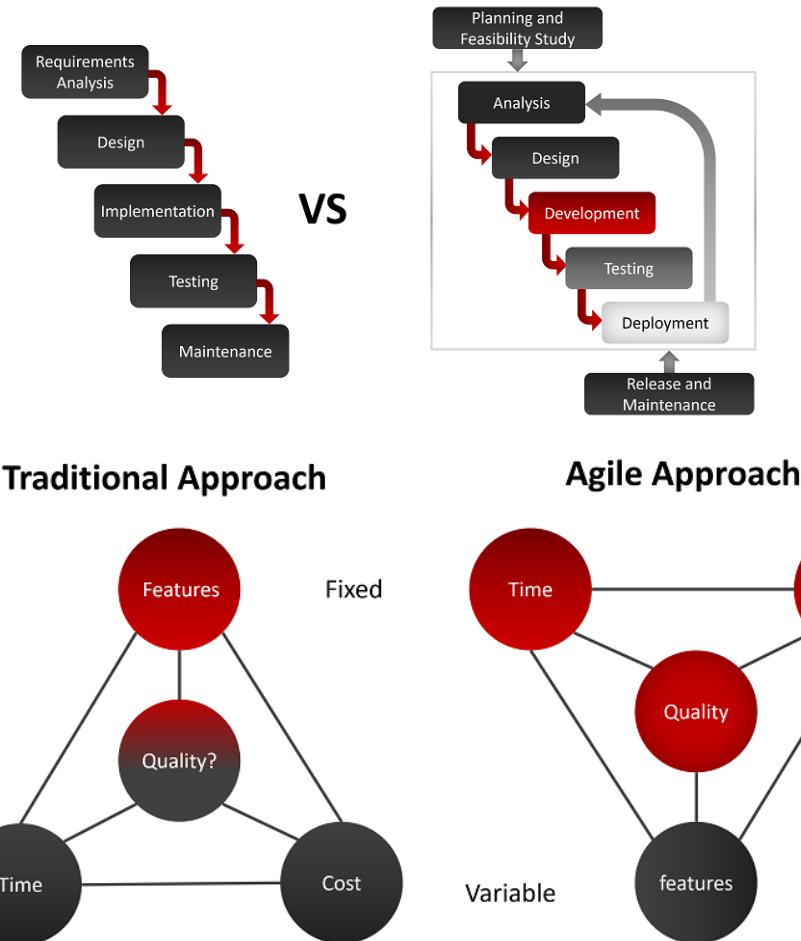
Benefits of Agile Methods/Approaches/ Practices/Techniques ...



Rearview mirror checking: provides staff with tools for regularly monitoring and self-correcting their work.

Continuous improvement: involves regularly reviewing and adjusting business activities to ensure that the organization is continuing to meet market and stakeholder demand.

Summary – Agile Methods (Module-1)



Difference between Traditional and Agile Project Management:

1. Flexibility (Rigid Vs Adaptive)
2. Ownership & Transparency (Project Manager vs Team ownership)
3. Problem Solving (Unexpected obstacles-Escalation vs Team take decision)
4. Checkpoints and Monitoring progress: (No Frequent check-ins vs Quicker Iteration delivering value)

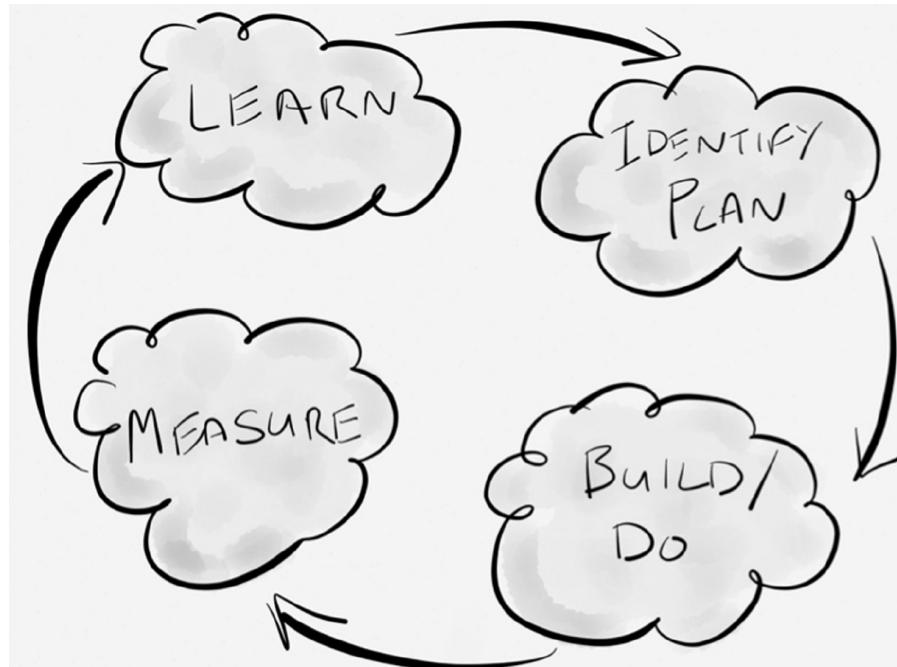
<https://www.kppartners.com/blog/traditional-vs-agile-software-development-methodologies#:~:text=The%20main%20difference%20between%20traditional,in%20Agile%2C%20it%20is%20iterative.>

How agile projects work

- Agile approaches are based on an ***empirical control method*** — a process of making decisions **based on the realities observed in the project**.
- In the context of software development methodologies, an empirical approach can be **effective in both new product development and enhancement and upgrade** projects.
- By using **frequent and firsthand inspection** of the work to date, you can make immediate adjustments, if necessary.

Empirical Process

- Empirical processes (see Figure) incorporate repeated inspection and adaptation of a product to ensure the right product is delivered in the right way. This is especially important in **environments** that **experience high variability** and are therefore **most suited to Agile** working.



Ref: Agile Foundations - Principles, practices and frameworks by Peter Measey

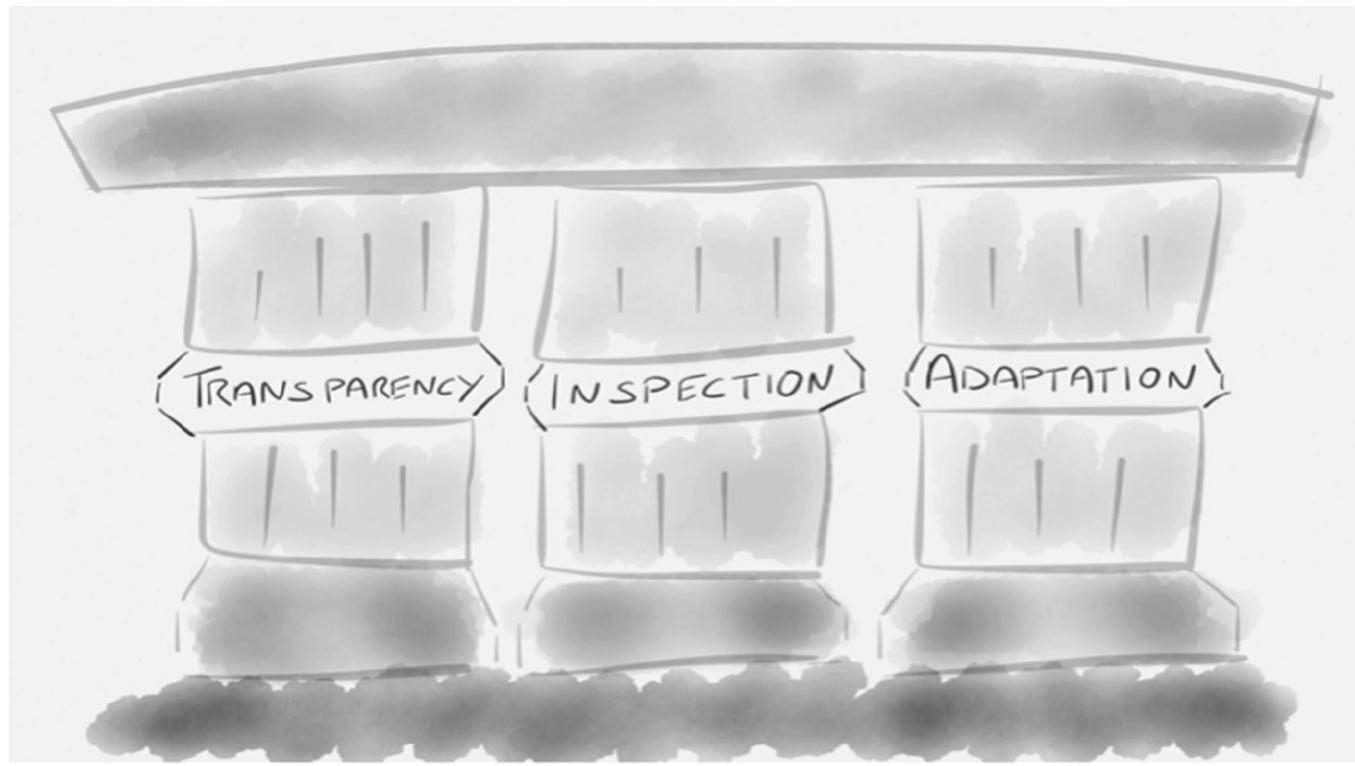
23-Aug-22

Agile Software Process SE SG544 S1-22-23

63

BITS Pilani, Pilani Campus

Pillars of Empirical control Method



Inspection – inspect the product being created and how it is being created **Adaptation** – adapt the product being created or the creation process if required
Transparency – ensure everyone can easily see what is happening

Frequent Iterations

- To accommodate frequent inspection and immediate adaptation, agile projects work in ***iterations*** (smaller segments of the overall project).
- An agile project involves the **same type of work** as in a **traditional waterfall** project:
 - You create **requirements and designs**, **develop the product**, **document it**, and if necessary, integrate the product with other products. You test the product, fix any problems, and deploy it for use.
 - However, instead of completing these steps for all product features at once, as in a waterfall project, you **break the project into iterations**, also called **sprints**.

Examples of Empirical models

- **PDCA** Plan, Do, Check, Act – Edward Deming (Deming, n.d.).
- **POOGI** Process of On-Going Improvement – Theory of Constraints (Goldratt and Cox, 1984).
- **OODA** Observe, Orient, Decide, Act – John Boyd (Boyd, n.d.).
- **BML** Build, Measure, Learn – Lean Start-up (Ries, 2011).
- **DMAIC** Define, Measure, Analyse, Improve, Control (Six Sigma, 2006).
- **TAC** Thought, Action, Conversation – DSDM Agile Project Framework (DSDM Consortium, 2014b).
- **Kaizen** A Japanese word which means ‘good change’, used to describe a philosophy of continuous improvement (Liker, 2004).

Thank you

Thank you



BITS Pilani
Pilani Campus

BITS Pilani presentation

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



SE ZG544 , Agile Software Process

Lecture No. 2 – Module-2 : Agile Software Development

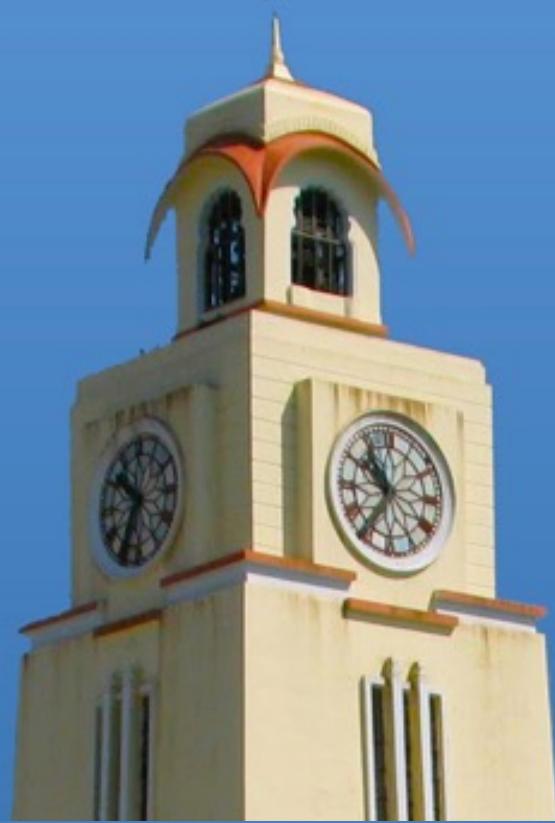
Module-2 Topics

1. Project Life Cycle Models

- Iterative, Incremental and (Adaptive or Agile) Approaches

2. Early Agile Models

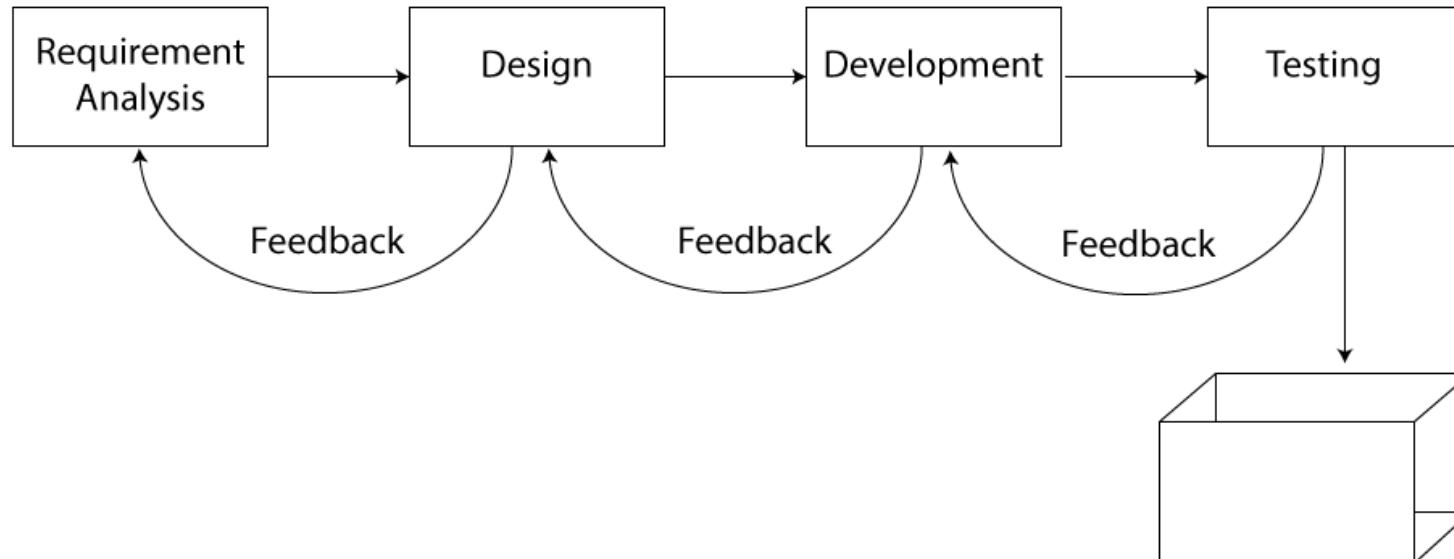
3. Popular Agile Methods



Project Life cycle models

Predictive Project Development Life Cycle

- Fully Plan-Driven aka Waterfall - Risky Invite Failure



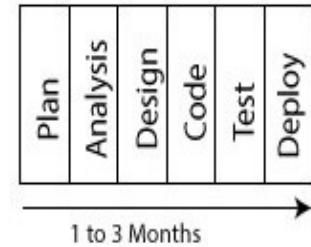
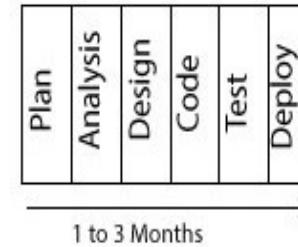
- Goal, Characteristics: Cost, Requirements-Fixed, Single Delivery

Iterative & Incremental Project Development Life Cycle



Plan

Iterative



- Goal, Characteristics:
- Iteration Model: Accuracy/Correctness, Single Delivery
- Incremental: Speed, Multiple Deliveries
- Requirements-Dynamic
- **Iteration life cycle:** Product Increment/output may **not** be usable
- Example: Customized outfit/coat, Website
- **Incremental life cycle:** Product Increment/output is usable
- Example: Visit to a restaurant

Source: <https://www.izenbridge.com/>

Q&A

Q1, Q2

<https://forms.gle/E8rdswFxp6B2pWfD9>

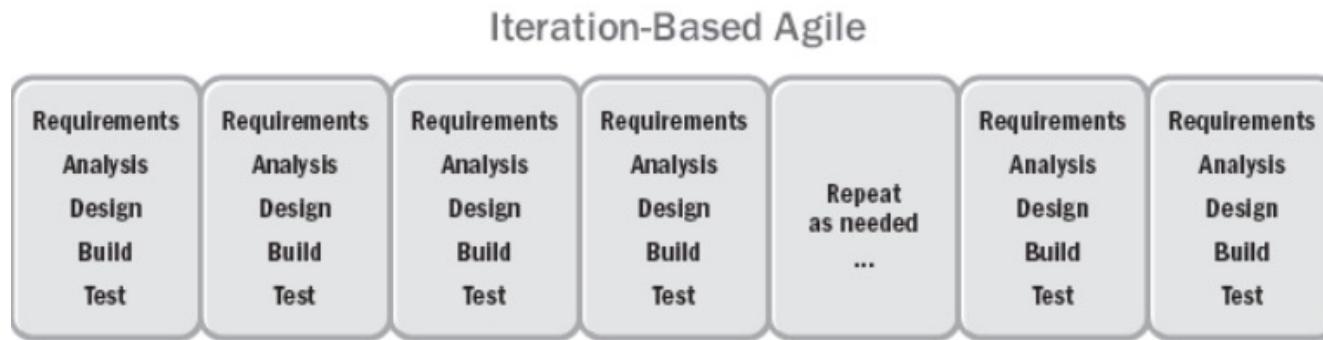
<https://forms.gle/A9W5ikfCJPaT5tbg8>

Agile Life Cycle Models

- Iterative
- Flow-Based

Agile/Adaptive Life Cycle- Iteration Based Agile

Plan



NOTE: Each timebox is the same size. Each timebox results in working tested features.

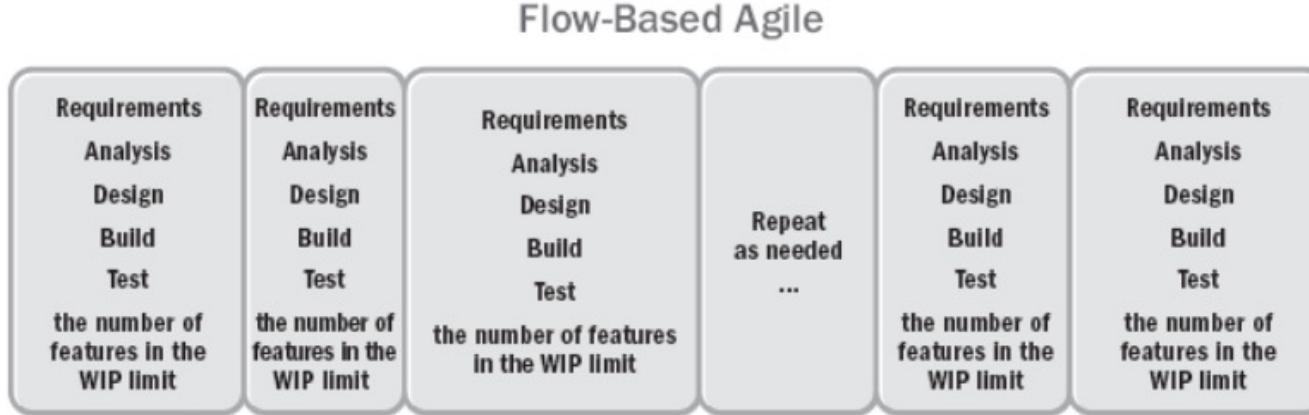
Fixed Time box : 1-4 weeks equal duration for each iteration

Goals and Characteristics: Value, Multiple deliveries

Agile/Adaptive Life Cycle- Flow-based Based Agile

- The project life cycle that is **iterative and incremental**

Plan



NOTE: In flow, the time it takes to complete a feature is not the same for each feature.

Variable Time Box :

Goals and Characteristics: Value, Multiple deliveries

Popular Early* Iterative and Agile Models



* Year ~2000 before

- Iterative
 - Spiral
 - RUP
- Agile
 - DSDM
 - FDD
 - Crystal

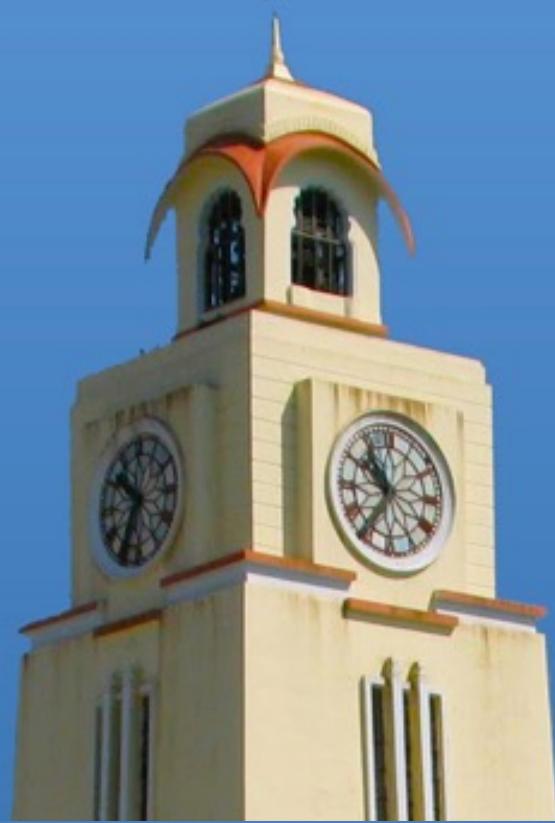
Q&A

Q3,Q4,Q5

<https://forms.gle/bz3784DRQYHD5BnN9>

<https://forms.gle/TZpHt55eRhKjnZT9>

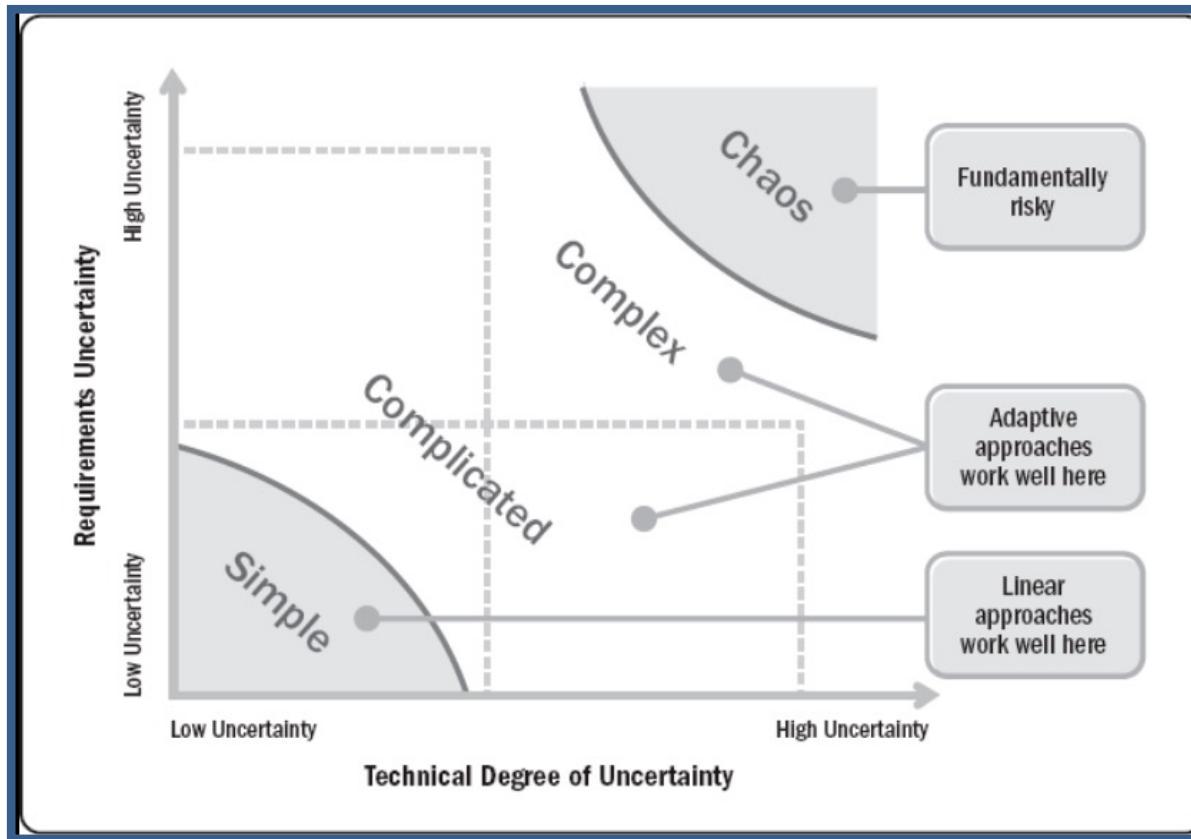
<https://forms.gle/7XCdp9E7yr9ABG2r8>



Project Classifications/Decision making models

Agile Suitability - Project Environment

Stacey's Complexity Model



Ref: Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)

Cynefin framework - A Leader's Framework for Decision Making



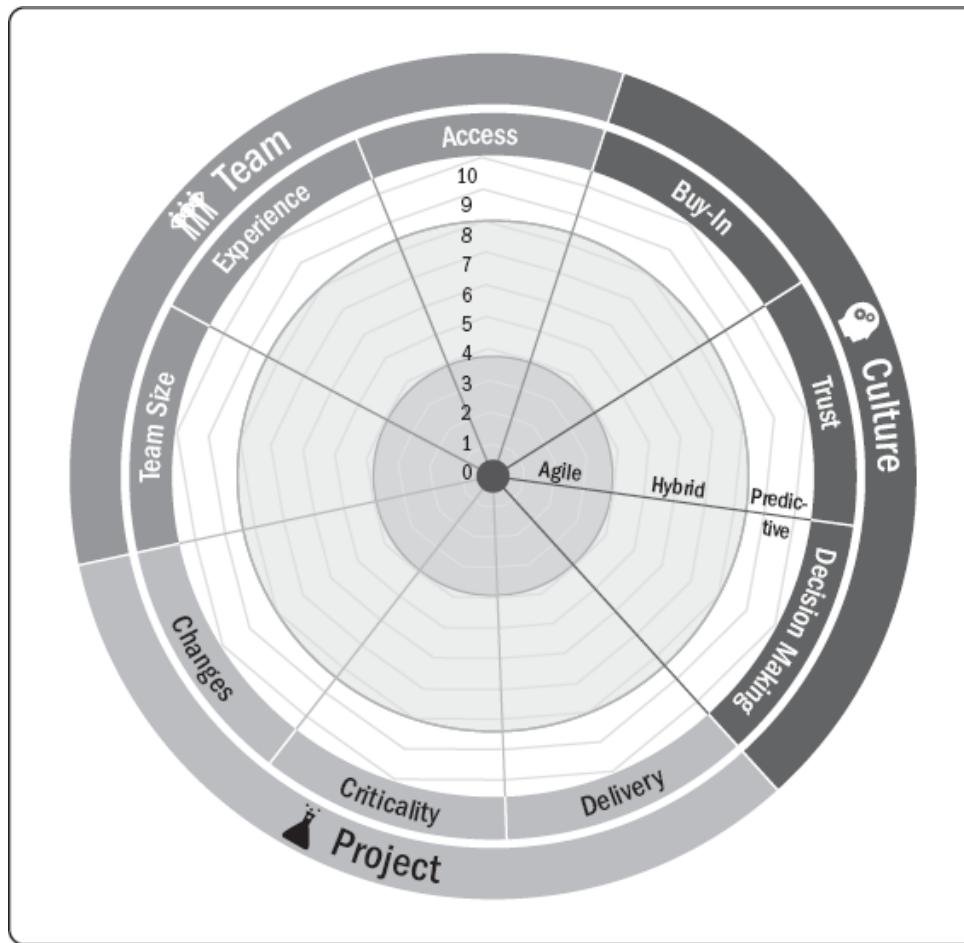
Pronounced as: kun-ev-in

Emergent

Developed in the early 2000s by David Snodown

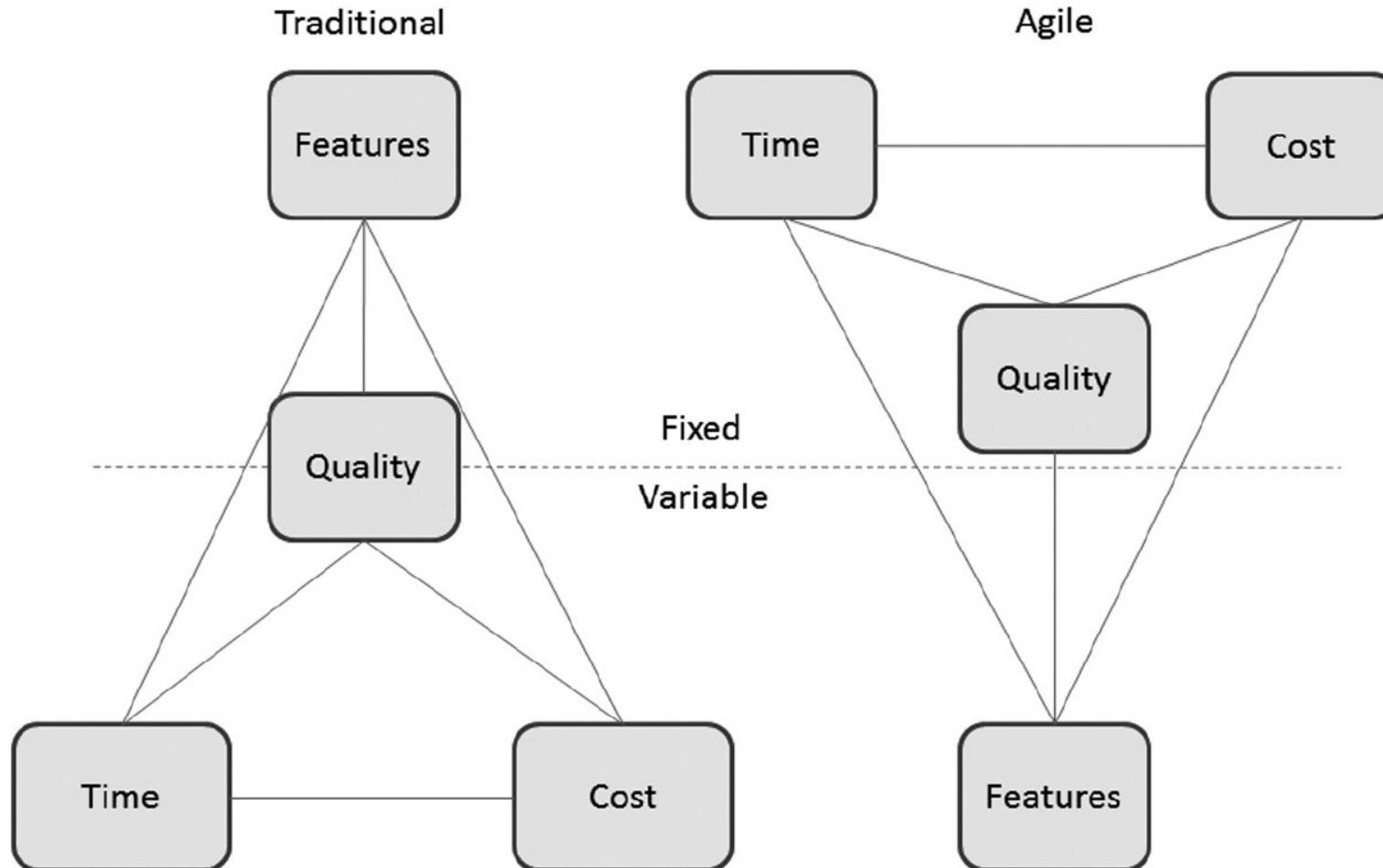


Agile Suitability Filter

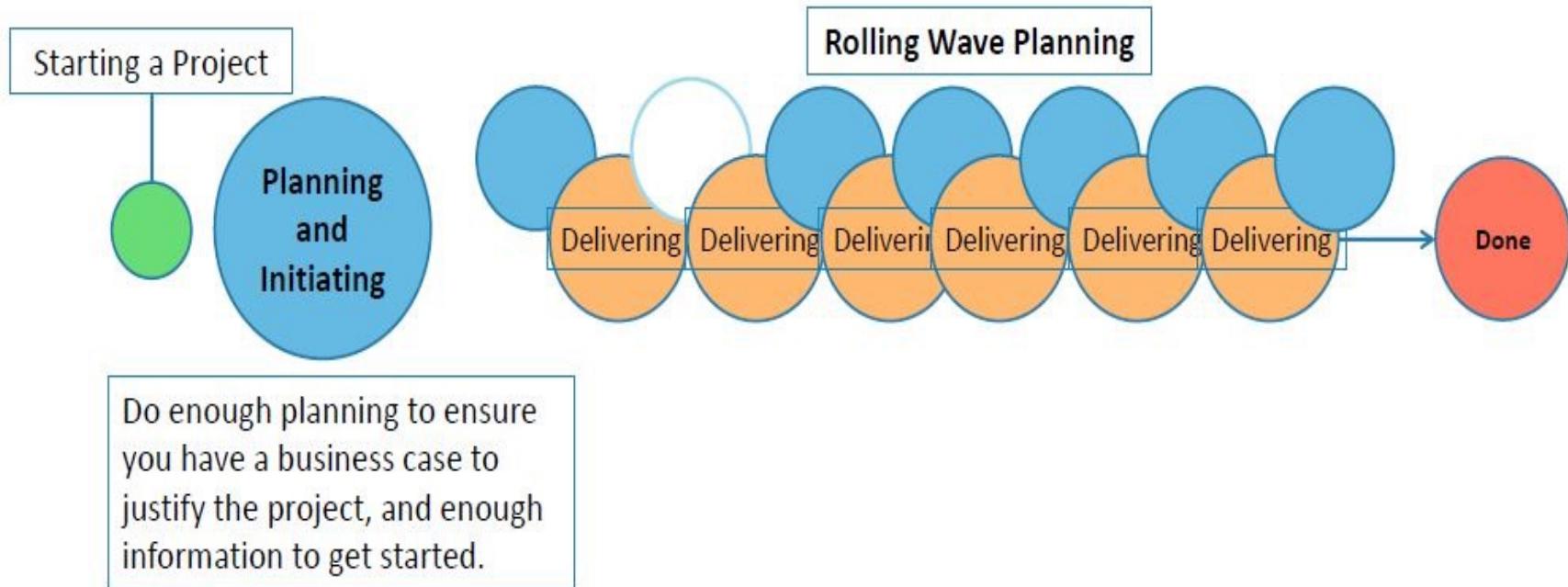


Attributes	Assessment
Buy-In	0-Yes, 5-Partial, 10-No
Trust	0-Yes, 5-Probably, 10-No
Decision Making	0-Yes, 5-Probably, 10-Unlikely
Incremental Delivery	0-Yes, 5-Maybe/Sometimes, 10-Unlikely
Criticality	0-Low, 5-Medium, 10-High
Changes	0-High, 5-Medium, 10-Low
Team Size	1-Small (<10), 5-Medium (>80), 10- Large (>200)
Experience	0-Yes, 5-Partial, 10-No
Access to business Info/Project Info	0-Yes, 5-Partial, 10-No

The “IRON” Triangle – Triple Constraints

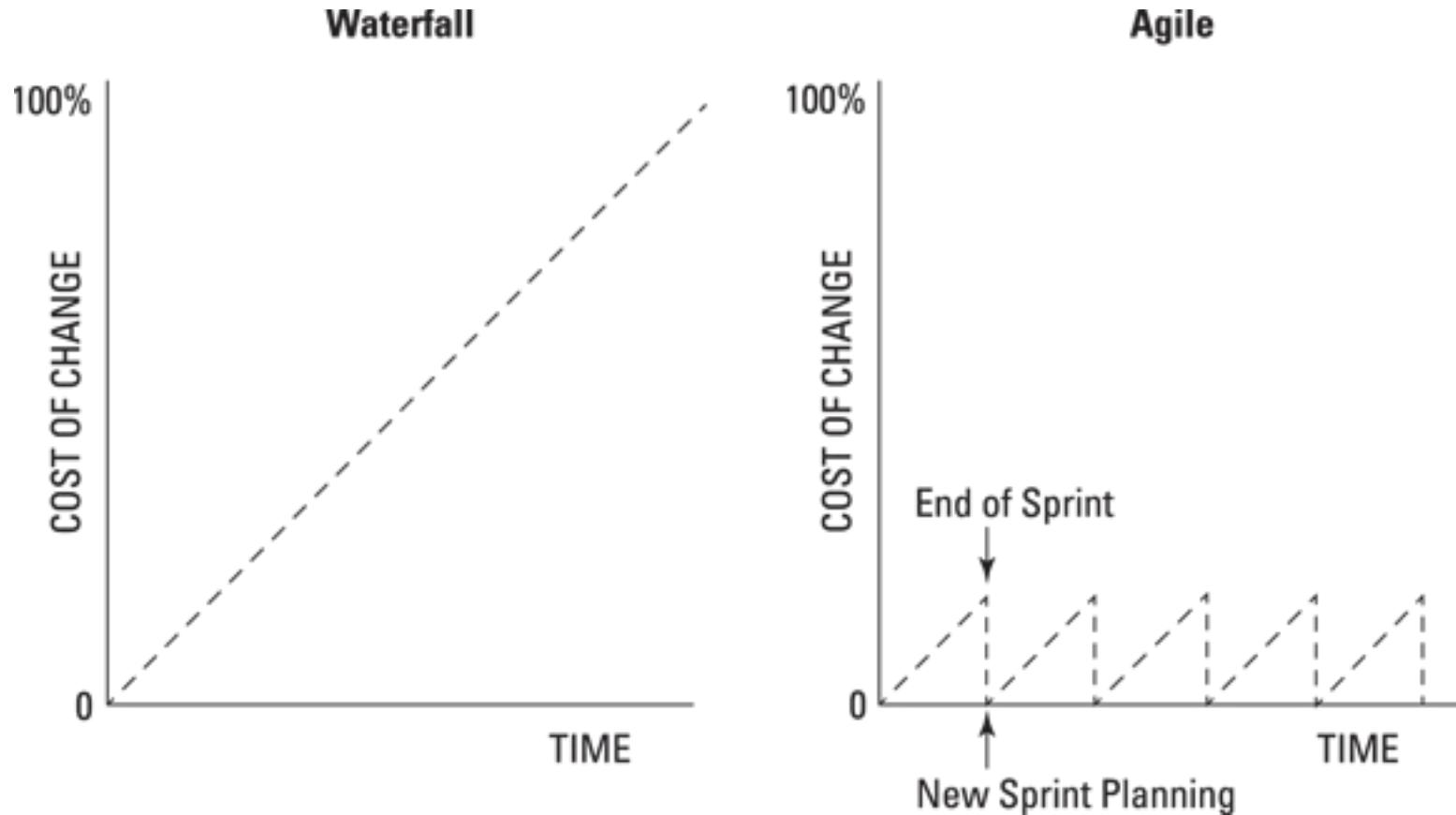


Rolling Wave Planning or Progressive Elaboration



<https://www.simplilearn.com/adaptive-planning-part-1-tutorial>

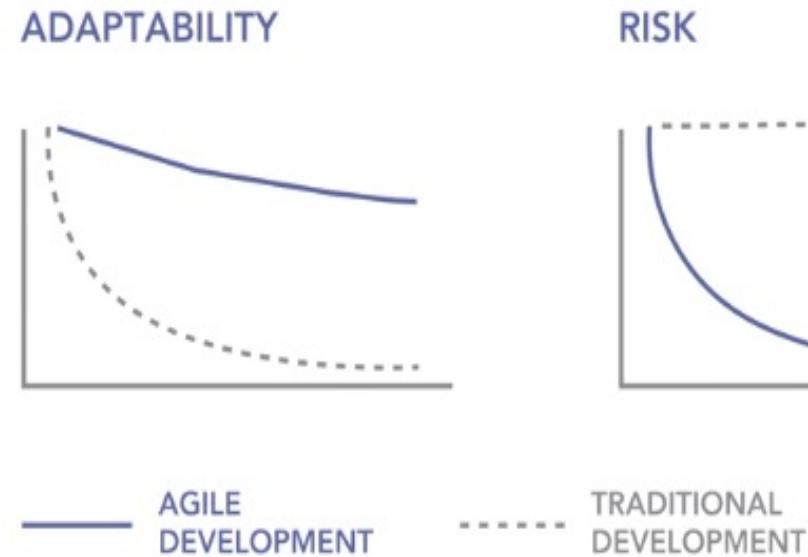
Cost of Changes: Agile vs Traditional Development



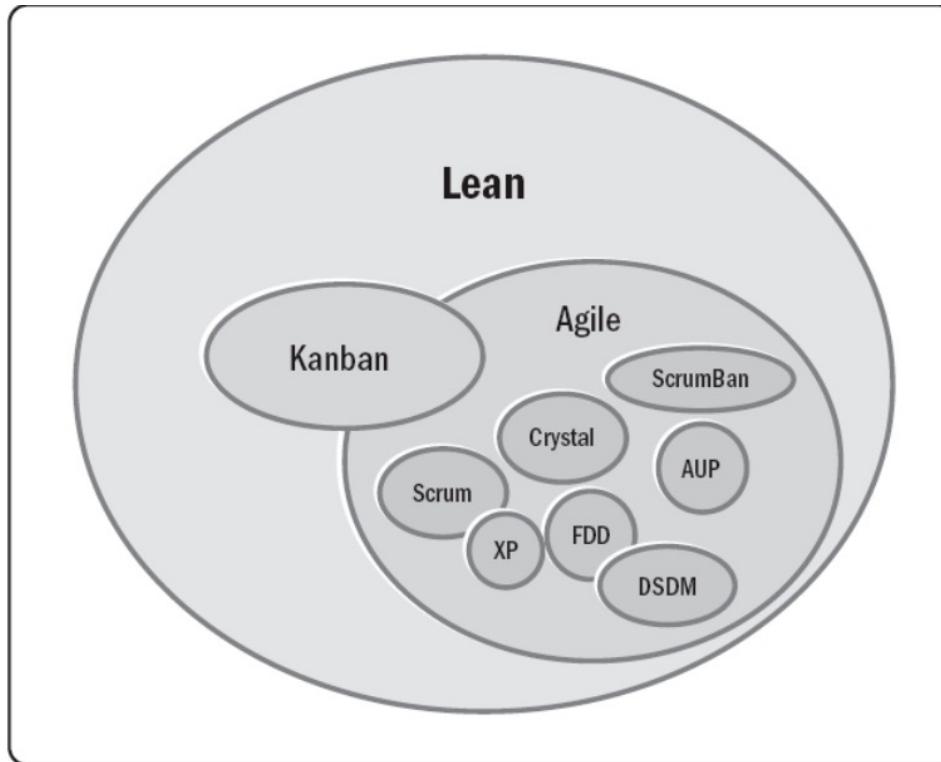
Adaptability and Risk Agile vs Traditional Development



Agile vs. Traditional Development



Popular Agile Methods



- They all have one important thing in common:
- they focus on changing your team's **mindset**.

Mindset

	The Agile mindset	The bureaucratic mindset
Goal	<i>The Law of the Customer</i> —an obsession with delivering steadily more value to customers.	<i>The Law of the Shareholder</i> : A primary focus on the goal of making money for the firm and maximizing shareholder value.
How work gets done	<i>The Law of the Small Team</i> —a presumption that all work be carried out by small self-organizing teams, working in short cycles, and focused on delivering value to customers	<i>The Law of Bureaucrat</i> : A presumption that individuals report to bosses, who define the roles and rules of work and performance criteria.
Organizational Structure	<i>The Law of the Network</i> —the presumption that firm operates as an interacting network of teams,	<i>The Law of Hierarchy</i> : the presumption that the organization operates as a top-down hierarchy, with multiple layers and divisions.

Source: <https://www.forbes.com/sites/stevedenning/2019/08/13/understanding-the-agile-mindset/?sh=5a66a5545c17>

Q&A

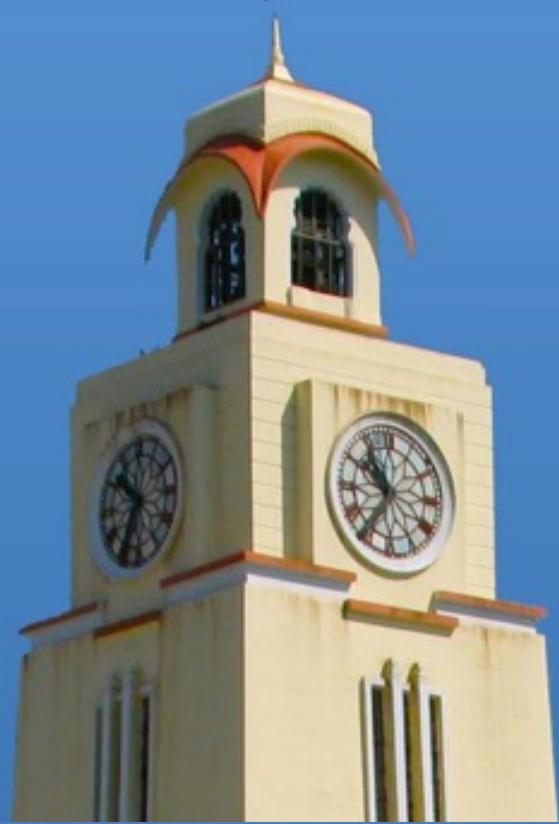
Q6,Q7,Q8

<https://forms.gle/9SUn7mtfuxcZ3kZUA>

<https://forms.gle/2f6AqYXnGMyNbKdA>

<https://forms.gle/3USQSAg2vKScGPZf7>

End Contact Session-2



Module-2 Additional Notes

Life Cycle

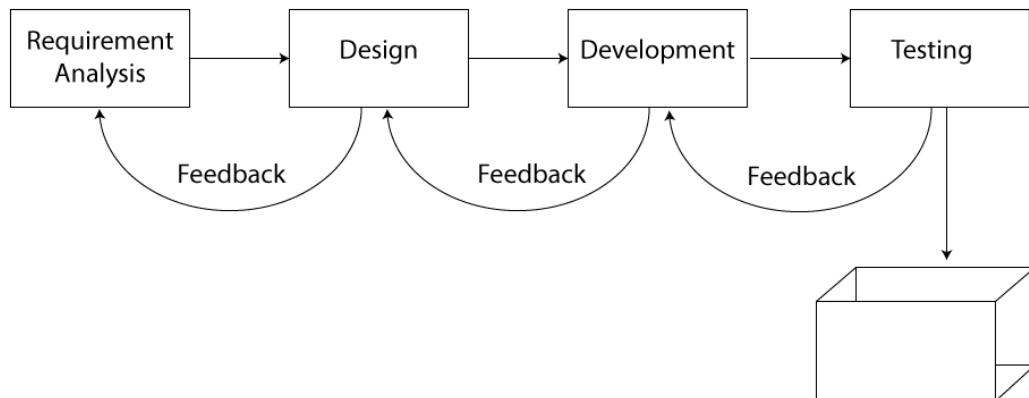
- The **sequence of actions** that must be **performed** in order to **build a software** system
- **Ideally** thought to be a **linear** sequence: **plan, design, build, test, deliver**
 - This is the waterfall model
- **Realistically** an **iterative process**
 - Iterative, Incremental, Agile Process

Predictive Project Development Life Cycle (Fully Plan-Driven aka Waterfall)



- A more **traditional approach**, with the bulk of **planning** occurring **upfront**, then executing in a **single pass**; a **sequential** process

Plan



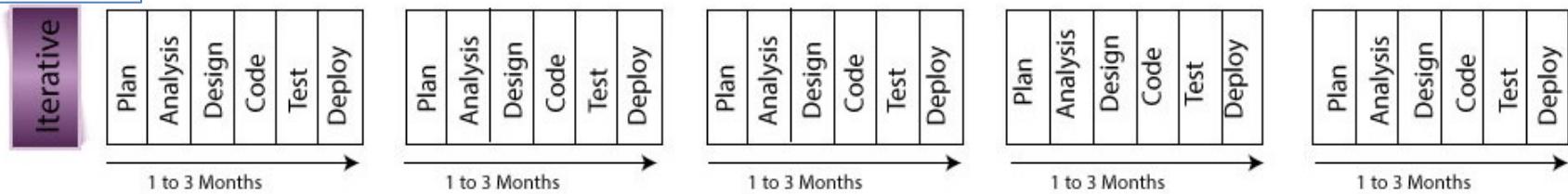
- Requirements/Scope is fixed
- Single delivery
- Goal: Manage Cost
- Minimal feedback changes
- Team is matured in estimation, technology etc..
- Project governance model exists
- **Don't expect long feedback cycle**, If this happens, this lifecycle not suitable for the project

Source : <https://www.izenbridge.com/blog/project-management-life-cycle-iterative-adaptive/>

Iterative Project Development Life Cycle

- Iterative development is when an attempt is made to **develop a product with basic features**, which then goes through a **refinement process** successively to **add to the richness** in features.

Plan



- Goal: **Correctness** of Solution
- Repeat** until Correct
- Show and **receive feedback**
- Add richness or features**
- Single Final Delivery**

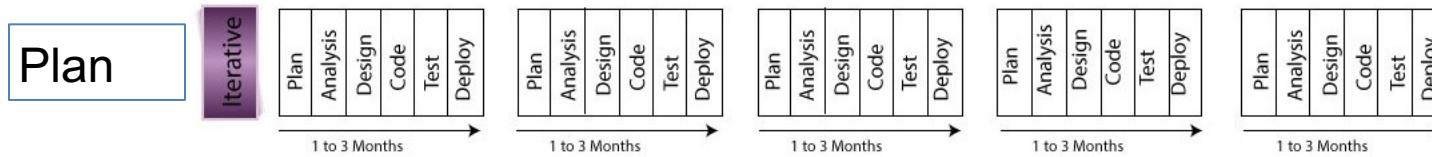
- Deliver result** at the end of each iteration.
- Result may **not be usable**
- E.g. 1 year project divided into 3 to 4 iterations

Examples of Iterative Development

- When you are getting a customized coat made
 - You may be required to go for a trial to check for the fitting.
 - Even though the you may find the coat fitting well, you may not be able to use it as it has not been finished.
 - The fitting test was to give you an idea of the final product, which may not be ready for your consumption.
 - This is an example of iterative prototyping.
- Developing a Website
 - Develop a prototype of the Website with basic functionality
 - Demo to Customer and receive feedback
 - Add to the richness or feature to the product in subsequent iteration

Incremental Life Cycle

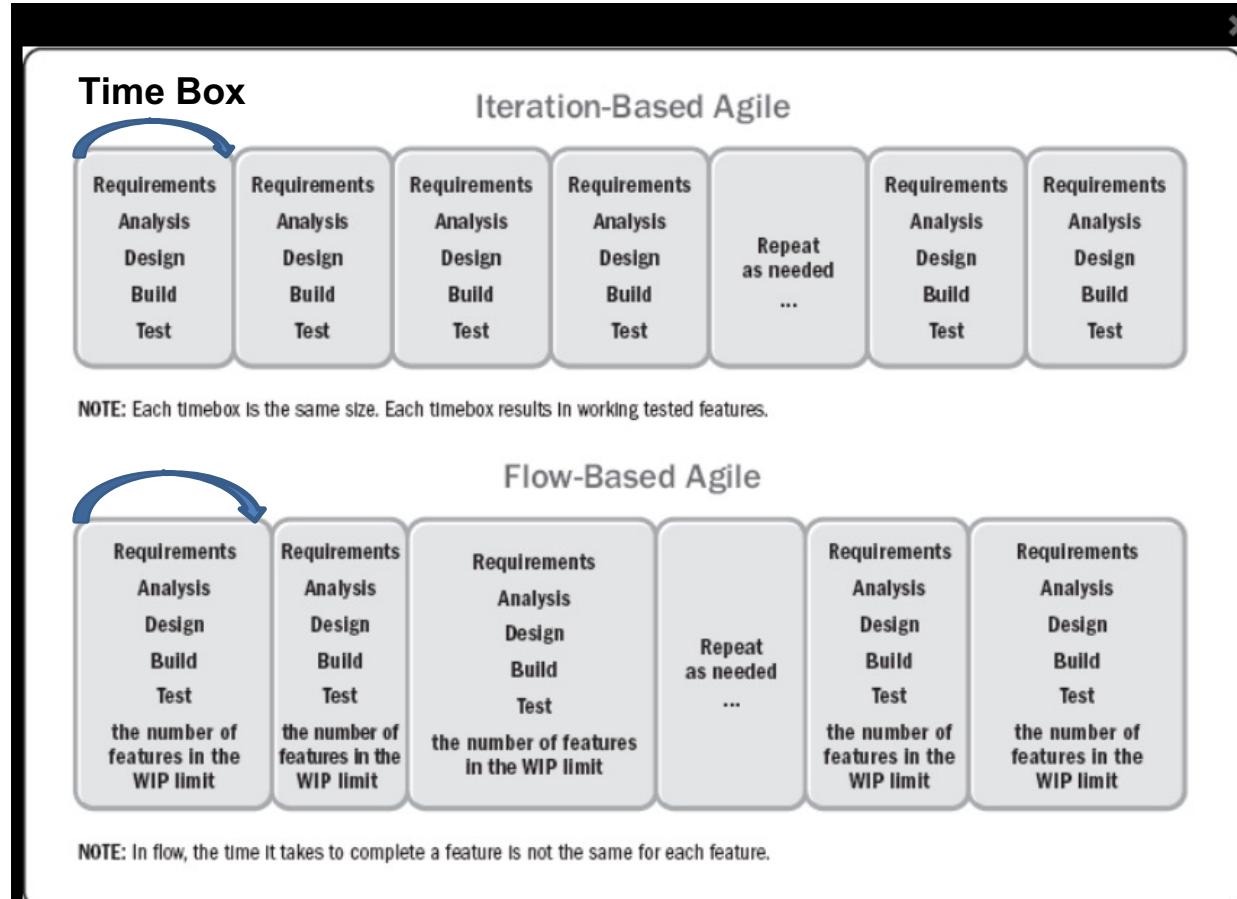
- In an incremental approach, one aims to **build pieces of program/product that is complete in features and richness**. Product **increment** is **usable**.
- In this case, each functionality is built to its fullest and **additional functionalities are added in an incremental fashion**.



Example: You can compare this to a visit to restaurant. You get served starters first and on completion of its main course and then dessert. You get served incrementally and you consume it.

Agile/Adaptive Life Cycle

- The project life cycle that is **iterative and incremental**



Fixed Time box : 1-4 weeks equal duration for each iteration

Limit WIP (work in Progress)
Optimize the flow

Ref: Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)

Project Life Cycles

Characteristics

Characteristics				
Approach	Requirements	Activities	Delivery	Goal
Predictive	Fixed	Performed once for the entire project	Single delivery	Manage cost
Iterative	Dynamic	Repeated until correct	Single delivery	Correctness of solution
Incremental	Dynamic	Performed once for a given increment	Frequent smaller deliveries	Speed
Agile	Dynamic	Repeated until correct	Frequent small deliveries	Customer value via frequent deliveries and feedback

- It should be emphasized that **development life cycles are complex and multidimensional**.
- Often, the **different phases in a given project employ different life cycles**, just as distinct projects within a given program may each be executed differently.

Ref: Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)

Delivery Environments and Agile Suitability

BITS Pilani

Pilani Campus

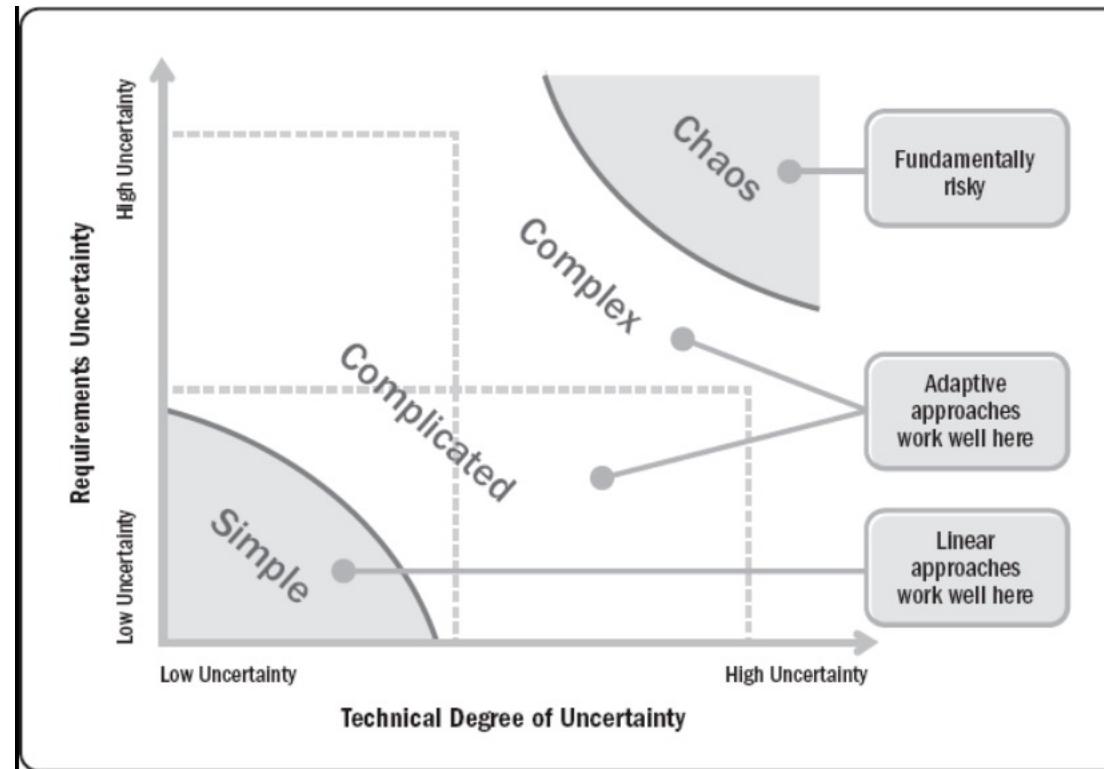


Delivery Environments and Agile Suitability

Delivery Environments

- The environment within which **Project/Product delivery** will occur should largely drive the delivery and **governance framework(s)** that will be implemented.
- For example, in a delivery environment where **high variability** is likely to be encountered (like IT product development), an **Agile framework** would be suited
- In an environment where **variability** is likely to be **low**, a **more defined process** may be more suited (like '**Waterfall**').

Understanding the Delivery Environments: Stacey's Complexity Model



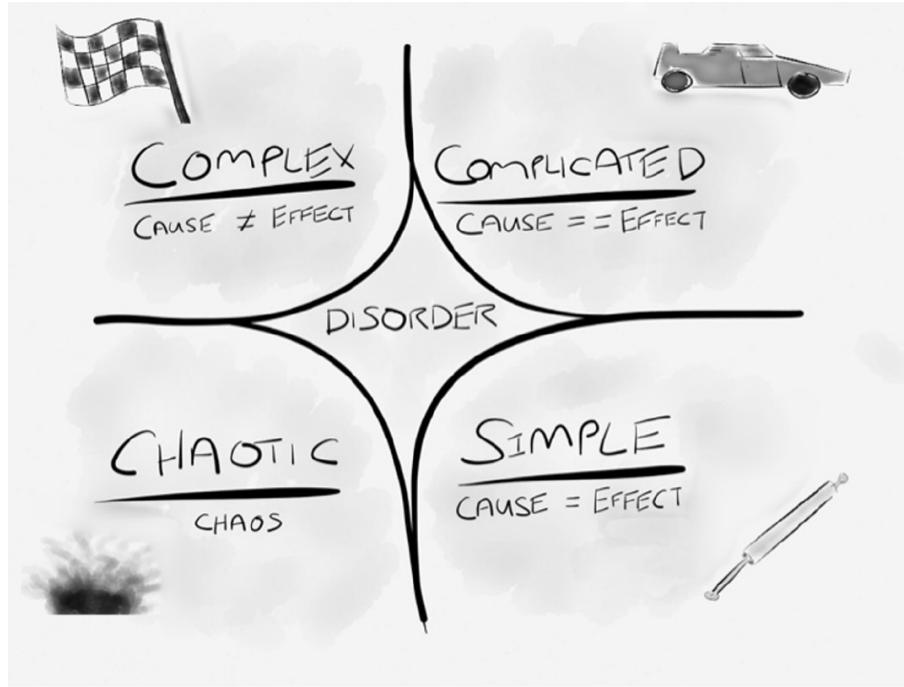
- Simple Environment: Use defined Process like Waterfall
- Complicated/Complex/Anarchy Environment: Use Empirical process like Agile.
Example: New IT product development

When trying to understand types of environments, it is important to take into account the amount of innovation that is being sought or considered for a new product or service. As the level of innovation increases, so does the move towards complexity, and a high variability is likely to be present.

Ref: Agile Foundations - Principles, practices and frameworks by Peter Measay

Cynefin Framework for Decision Making

- The Cynefin framework (Snowdon and Boone, 2007) gives an alternative framework for determining and understanding simple, complicated and complex environments



- The central idea of the framework is to offer decision-makers a “sense of place” to view their perceptions in dealing with a situation or problem. Not all situations are equal, and this framework helps to define which response is required for a given situation or problem.

<https://txm.com/making-sense-problems-cynefin-framework/>

Cynefin identifies five domains:

- **Simple (obvious) domain:**
 - In this domain the relationship between cause and effect is obvious and therefore it is relatively easy to predict an outcome. In this domain predictive planning works well as everything is pretty well understood. Teams can define up front how best to deliver a product, and they can then create a defined approach and plan. The Waterfall model works well in these types of environments with little variability.
- **Complicated domain**
 - In this domain, the relationship between cause and effect becomes less obvious; however, after a period of analysis it should generally be possible to come up with a defined approach and plan. Such a plan will normally include contingency to take into account the fact that the analysis may be flawed by a certain amount. Again, the Waterfall model is suitable for this environment as there is an element of definition up front; however, a more empirical process, like Agile, may be more suited.

<https://txm.com/making-sense-problems-cynefin-framework/>

Cynefin identifies five domains:

Complex domain

- In this domain the relationship between cause and effect starts to break down as there tend to be many different factors that drive the effect. While it may be possible to identify retrospectively a relationship between cause and effect, the cause of an effect today may be different to the cause of the same effect tomorrow. Creating a defined up-front approach and plan is not effective within this domain and therefore an Agile way of working is recommended.

Chaotic domain

- In this domain, there is no recognizable relationship between cause and effect at all, making it impossible to define an approach up front or to plan at all. Instead, teams must perform experiments (e.g. prototyping, modelling) with the aim to move into one of the other less chaotic domains. An Agile approach can work in this domain, for example Kanban which does not require up-front plans.

Disorder

- Being in this environment means that it is impossible to determine which domain definition applies. This is the most risky domain as teams tend to fall into their default way of working, which may prove unsuitable for what they are trying to achieve.

A Note on Cynefin identifies five domains:

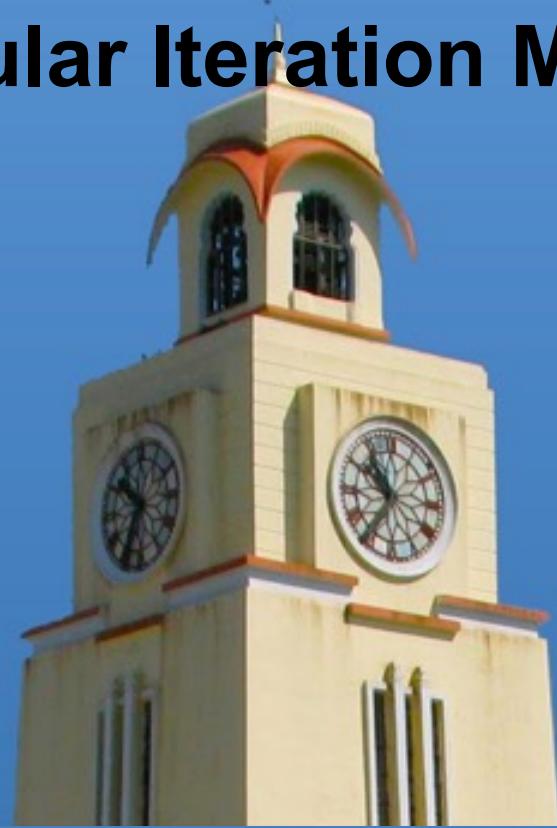
- During a product's development and evolution there may be **elements of delivery spread across** all the Cynefin domains at the same time.
- There **may be aspects of a large system that are simple**, while **others may be in the complicated domain**; and there could also be areas where innovation is necessary and which require a move towards the complex or even towards the chaotic domain.

<https://txm.com/making-sense-problems-cynefin-framework/>

Some Popular Iteration Models

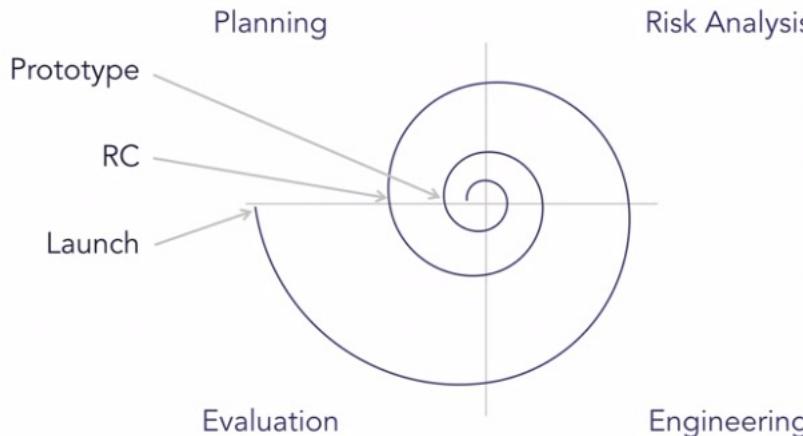


BITS Pilani
Pilani Campus



Some Popular Iteration Models

Spiral Risk Driven Customer driven Planning



- Developed by Barry Boehm, 1986.
- Easier management of risks (Theme)
- Mix of water fall and iterations
- Y-Axis represents Cost
- X-Axis represents Review
- Prototype-1, Prototype-2
- Operational Prototype
- Final Release

Four Phases

Planning: Requirements Identification and Analysis

Risk Analysis: Risk identification, Prioritization and Mitigation

Engineering: Coding, Testing and Deployment

Evaluation: Review and plan for next iteration

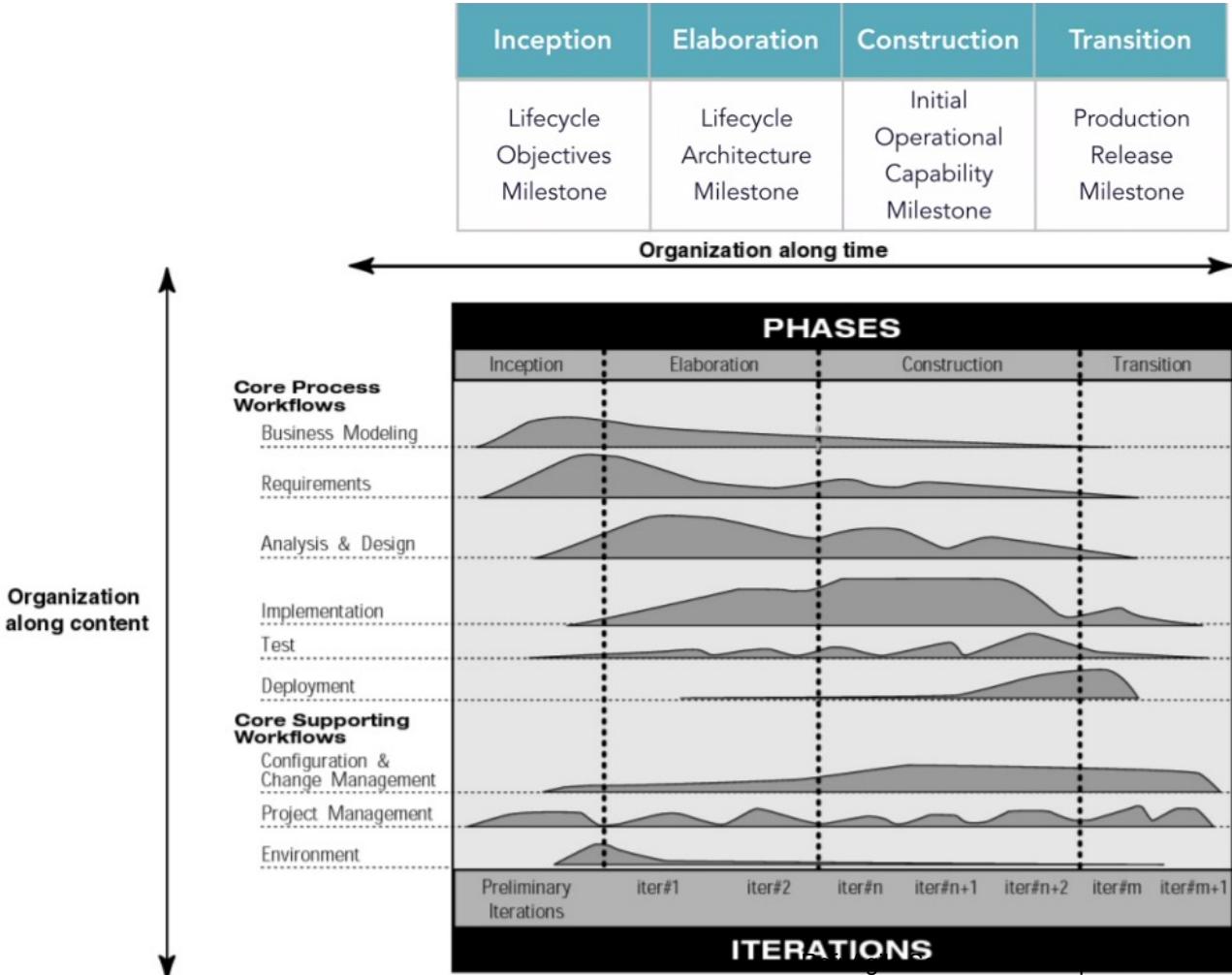
Rational Unified Process (RUP)



- 1990s, Rational Software developed the Rational Unified Process as a software process product.
- IBM acquired Rational software in 2006 (**era of OOAD, UML**)
- Rational Unified Process, or RUP, was an attempt to come up with a comprehensive **iterative software development** process.
- RUP is essentially a **large pool of knowledge**. RUP consists of **artifacts, processes, templates, phases**, and **disciplines**.
- RUP is defined to be a **customizable** process that would work for building small, medium, and large software systems.

Ref: Agile Software Development with Shashi Shekhar, LinkedIn Learning

RUP Iterative Model

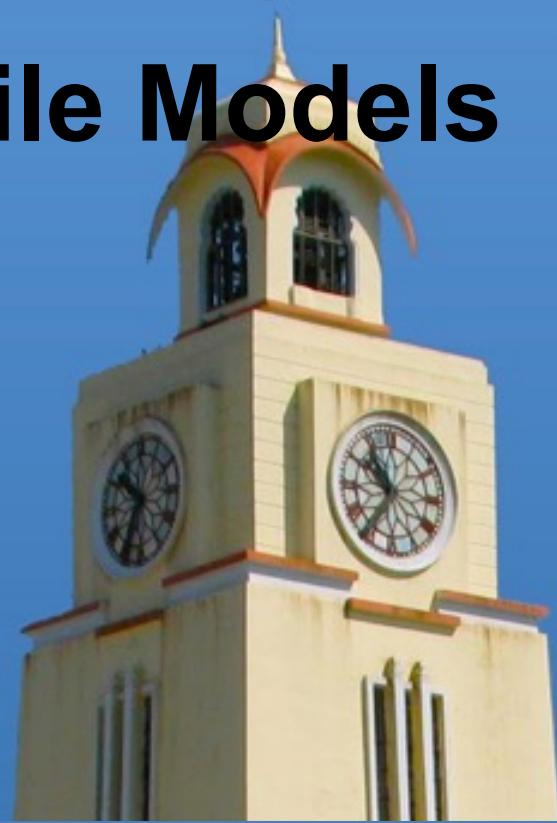


X-Axis: RUP Phases, Dynamic

Y-Axis: Organization Process, Static

Early Agile Models

BITS Pilani
Pilani Campus



Early Agile Methods

Dynamic System Development Method (DSDM)



The eight Principles of DSDM:

- Focus on the business need
- Deliver on time
- Collaborate
- Never compromise quality
- Build incrementally from firm foundations
- Develop iteratively
- Communicate continuously and clearly
- Demonstrate control

- Developed in 1994
- Era where organization slowly moving away from waterfall model
- During this time RAD model came into existence
- RAD approach is very agile but has no formal process
- DSDM was formed by group of organizations
- **Project development standard in Europe** for several years
- In 2016 DSDM is changed its name to **Agile business consortium**

<https://www.agilebusiness.org/>

Source: Agile Lynda.com, Agilebusiness.com

Feature Driven Development (FDD)



- Lightweight Agile process
 - Software is a collection of features
 - Software feature = “working functionality with business value”

Feature Example:

Calculate monthly interest on the account balance

(action) (result) (object)

- Deliver working software (working feature)
 - Short iterative process with five activities
 - Develop over all, Build Feature list, Plan by feature, Design by Feature Build by feature
 - FDD is used to build large banking systems successfully

Ref: Agile Software Development with Shashi Shekhar, LinkedIn Learning

Crystal Method- Selecting a Model

Life						
Essential Money						
Discretionary Money						
Comfort						
	1-6	7-20	21-40	41-80	81-200	Large
Team Size						

- Different crystal methodologies based on team size.
- If Criticality increases tweak the process to address the extra risk

Comfort: System malfunction

Discretionary Money: Extra savings

Essential Money: Revenue loss

Life: Loss of life , Critical software

- Crystal methods are people-centric, light-weight, and highly flexible. Focus on People, Interactions, Collaborations.
- Developed by Alistair Cockburn , 1991

Thank you



BITS Pilani
Pilani Campus



BITS Pilani presentation

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



SS ZG544 , Agile Software Processes

Lecture No. 3- Agile Manifesto & Principles

Agile Manifesto & Agile Principles



- <https://agilemanifesto.org/>
- Agile Practices
 - Agile Manifesto → Agile Principles → Agile Practices
 - Agile Practices → Project Outcome
- Sprint Planning, Product Backlog, Sprint Review, Planning Game, Frequent Delivery, Retrospective
- Definition of Done
- Whole Team, Osmotic Communication, Daily Scrum
- TDD, Pair Programming, Continuous Integration, 10-minutes Build

Q&A

Q.1 <https://forms.gle/biAfBryfBpevNVHdA>

Q.9 <https://forms.gle/tu1jJH6ok8UqFxyq9>

Agile Manifesto-1(Anti-Patterns)

When applying the Agile Manifesto:

Individuals and interactions over processes and tools.

- The tool makes us Agile
- Relentless automation
- Hierarchies
- Over-standardization

Ref: Agile From First Principles , Lynda Girvan, Simon Girvan. Published by BCS, The Chartered Institute for IT

Agile Manifesto-2(Anti-Patterns)

When applying the Agile Manifesto:

Working software over comprehensive documentation

- Because they asked us for it: Other parts of the organization often say they require additional documentation or reporting.
- We will need this later
- Documentation as collaboration
- Write only documentation

Q&A

Q.3 <https://forms.gle/biAfBryfBpevNVHdA>

Q.5 <https://forms.gle/bGYxP7Xipteqp1Sa8>

Agile Manifesto-3(Anti-Patterns)

When applying the Agile Manifesto:

Customer collaboration over contract negotiation

- Detailed story descriptions
- Fixed standards or processes
- Restricting who can talk to the customer
- Not considering cultural difference
- Lacking collaboration skills

Agile Manifesto-4(Anti-Patterns)



When applying the Agile Manifesto:

Responding to change over following a plan

- Iterations planned in advance
- The tool makes us plan
- Focus on the tasks not the value
- Small stories on the backlog

Ref: Agile From First Principles , Lynda Girvan, Simon Girvan. Published by BCS, The Chartered Institute for IT

Agile Principles – Customer Centric (Anti- Patterns)



Slanted toward customers

- 1** Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- 2** Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
- 3** Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- 4** Business people and developers must work together daily throughout the project

- Proxy customers (Business Analysts, Architect acting as customer)
- Considering plans and roadmaps as commitments
- Expecting too much detail
- Not engaging Out of sight, out of mind- Stakeholders

Ref: Agile From First Principles , Lynda Girvan, Simon Girvan. Published by BCS, The Chartered Institute for IT

Agile Principles – (Anti-Patterns)



Slanted toward managers

- 5** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
- 6** The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
- 7** Working software is the primary measure of progress
- 8** Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

- One size fits all approach towards team management
- Chasing the metrics
- Ignoring the environment
- Multiple deployment environments

Ref: Agile From First Principles , Lynda Girvan, Simon Girvan. Published by BCS, The Chartered Institute for IT

Q&A

Q.6 <https://forms.gle/xRSq1kwGNwfALi3C7>

Q.7 <https://forms.gle/DEPtdyZiTF5mzRfT8>

Q4 <https://forms.gle/iZcp4fkvHWiGPvT29>

Agile Principles – (Anti-Patterns)



Slanted toward the team

- 9** Continuous attention to technical excellence and good design enhances agility
- 10** Simplicity – the art of maximizing the amount of work not done – is essential
- 11** The best architectures, requirements, and designs emerge from self-organizing teams
- 12** At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

- Over-complicating things/Future proof everything
- Insisting on Sign-off processes
- Just in case' development-setting things up for later features
- Management focus on individuals

Ref: Agile From First Principles , Lynda Girvan, Simon Girvan. Published by BCS, The Chartered Institute for IT

Q&A

Q.10 <https://forms.gle/3Yp8aeJDR7956pvu6>



BITS Pilani
Pilani Campus

BITS Pilani presentation

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

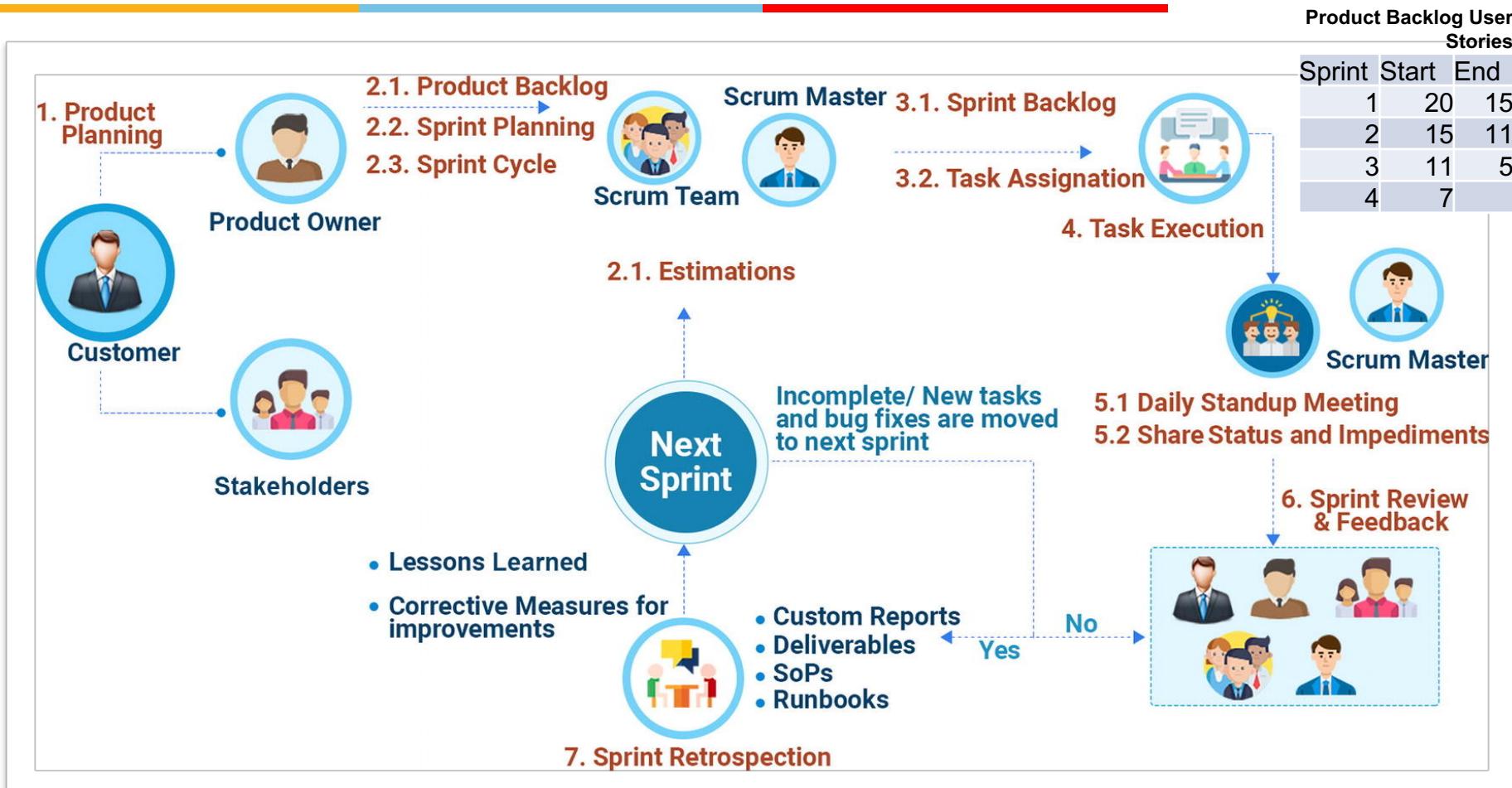


SE ZG544 , Agile Software Process Lecture No. 4 - Agile Methodologies

Agenda

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

Scrum Model



Progress Tracking - Scrum Task Board



Story	To Do		In Process	To Verify	Done
As a user, I... 8 points	Code the... 9	Test the... 8	Code the... DC 4	Test the... SC 6	Code the... D Test the... SC 8 Test the... SC Test the... SC Test the... SC 6
As a user, I... 5 points	Code the... 8	Test the... 8	Code the... DC 8		Test the... SC Test the... SC Test the... SC 6
	Code the... 2	Code the... 8			
	Test the... 8	Test the... 4			

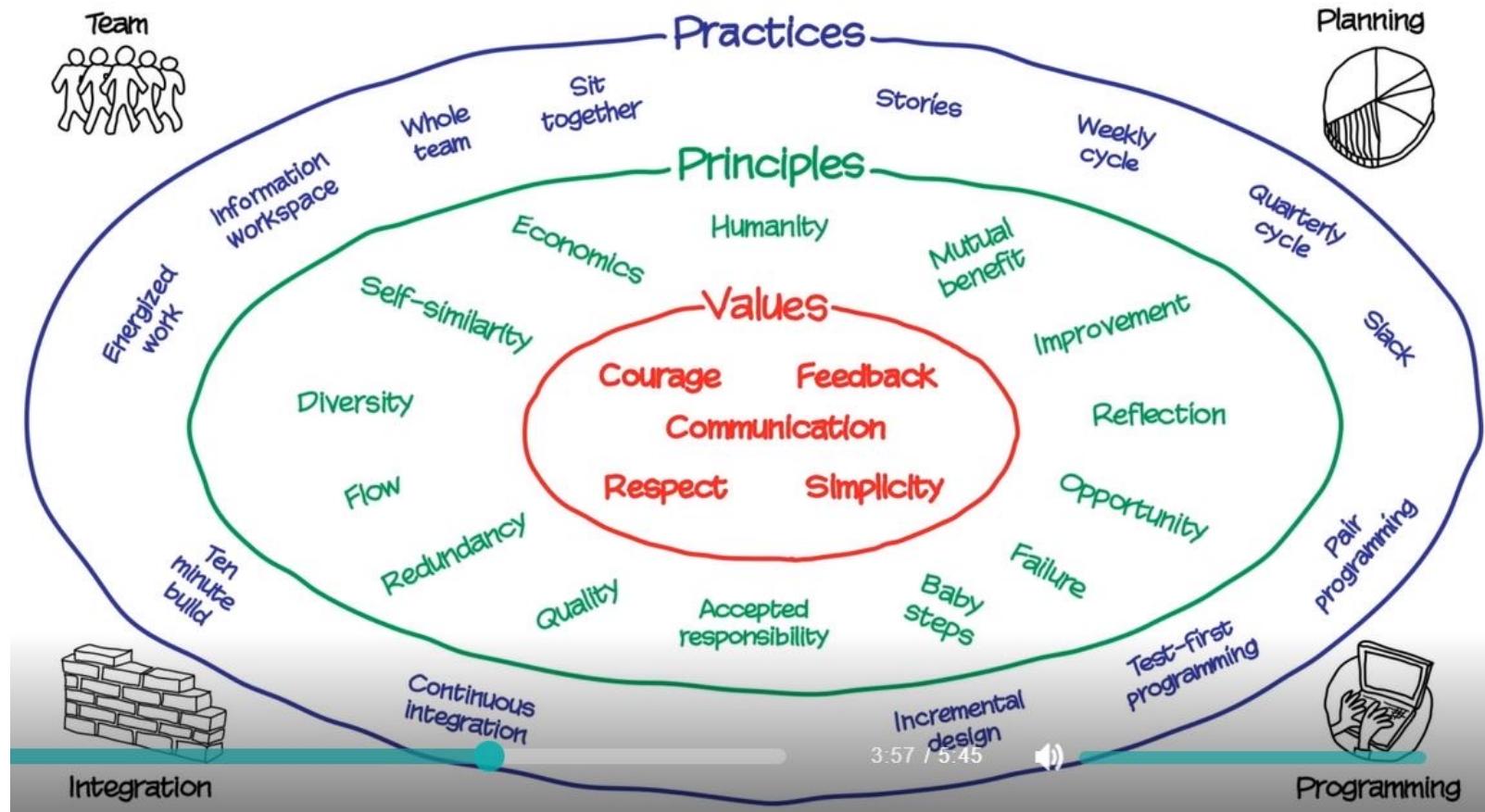
Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 2005

Q&A

- » Q1 <https://forms.gle/acdHAF5B1snczGt27>
- » Q2 <https://forms.gle/MkMUpFgMnxG6ddnw8>
- » Q5 <https://forms.gle/5d2oLmcT5DfrLHbJ8>
- » Q6 <https://forms.gle/TWVNxLATMoRG1hVr9>

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

Q&A

- » Q3 <https://forms.gle/ATAexHWAH1QuFWkL6>
- » Q7 <https://forms.gle/8svB5tSqTi6kCU2p8>
- » Q8 <https://forms.gle/uXC9yi2AVNZrjXeX6>
- » Q10 <https://forms.gle/zGBDQksPEi6yYcjn9>
- » Q9 <https://forms.gle/zes6nMXGkNYRLZUr9>

What is Lean Software Development? &

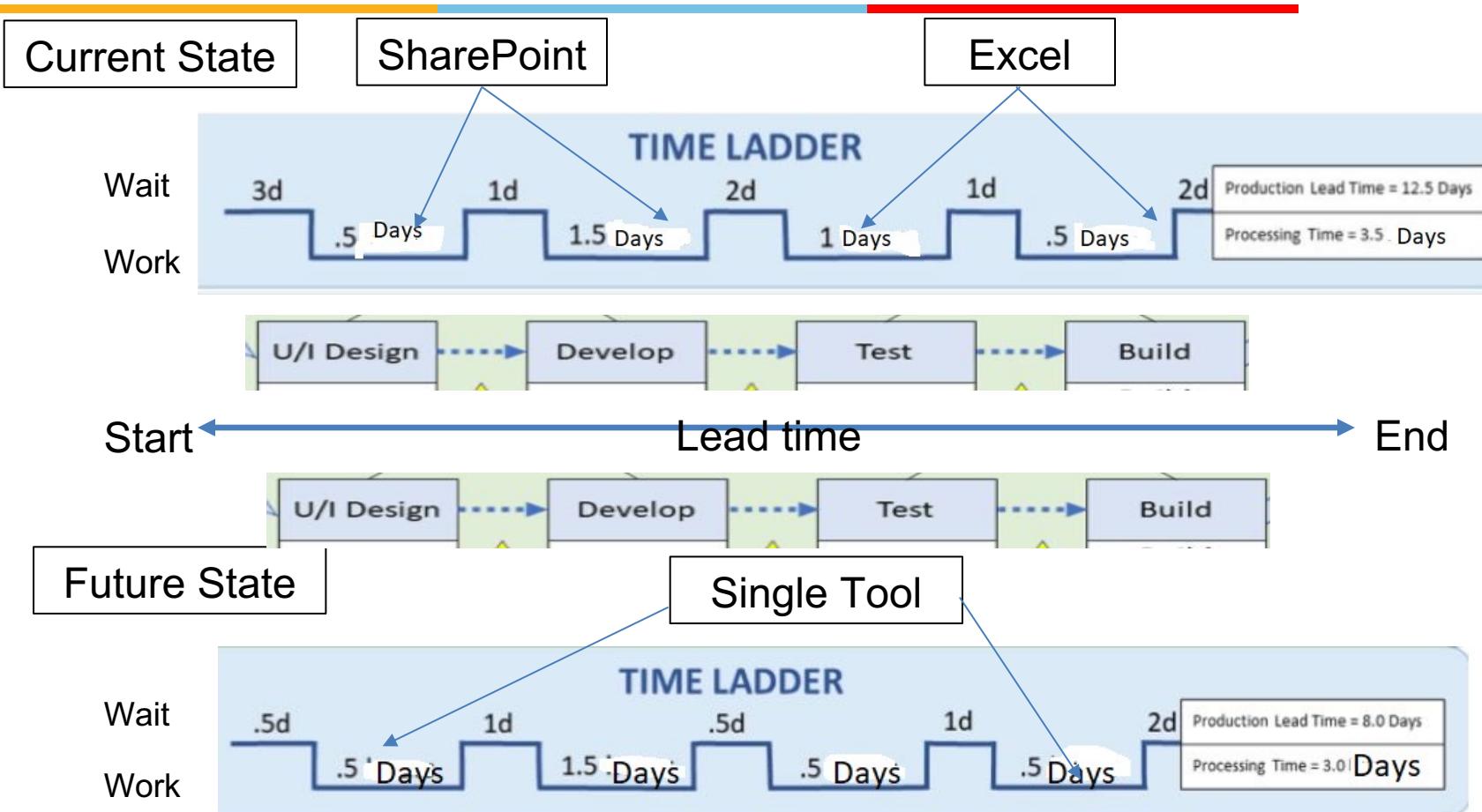
Seven Principles of Lean

- 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 Stream Mapping



Process Efficiency = Cycle time/Lead Time * 100

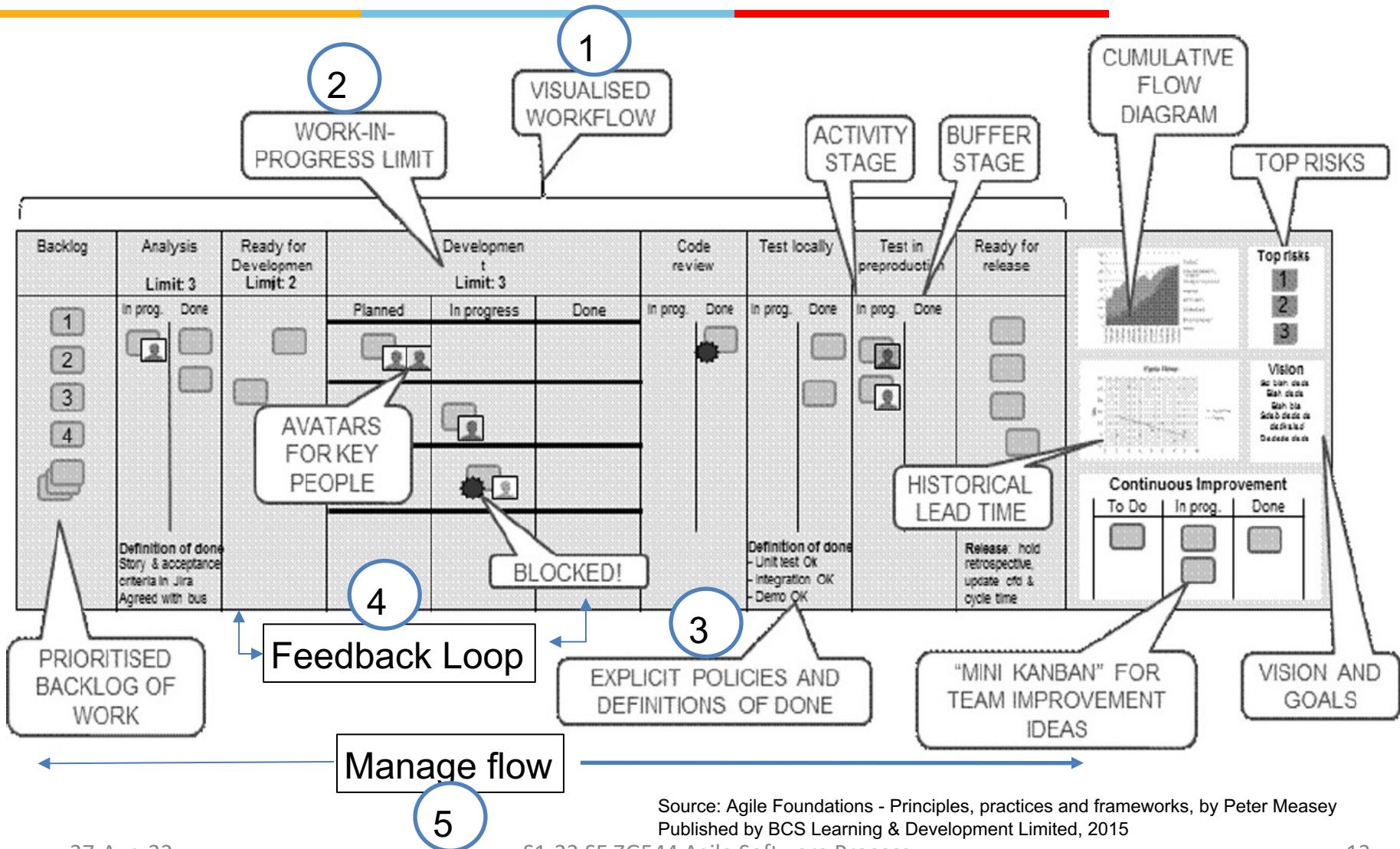
Current State: $3.5/(12.5)*100 = 28\%$; Future state: $3/(8)*100 = 37.5\%$

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

Kanban

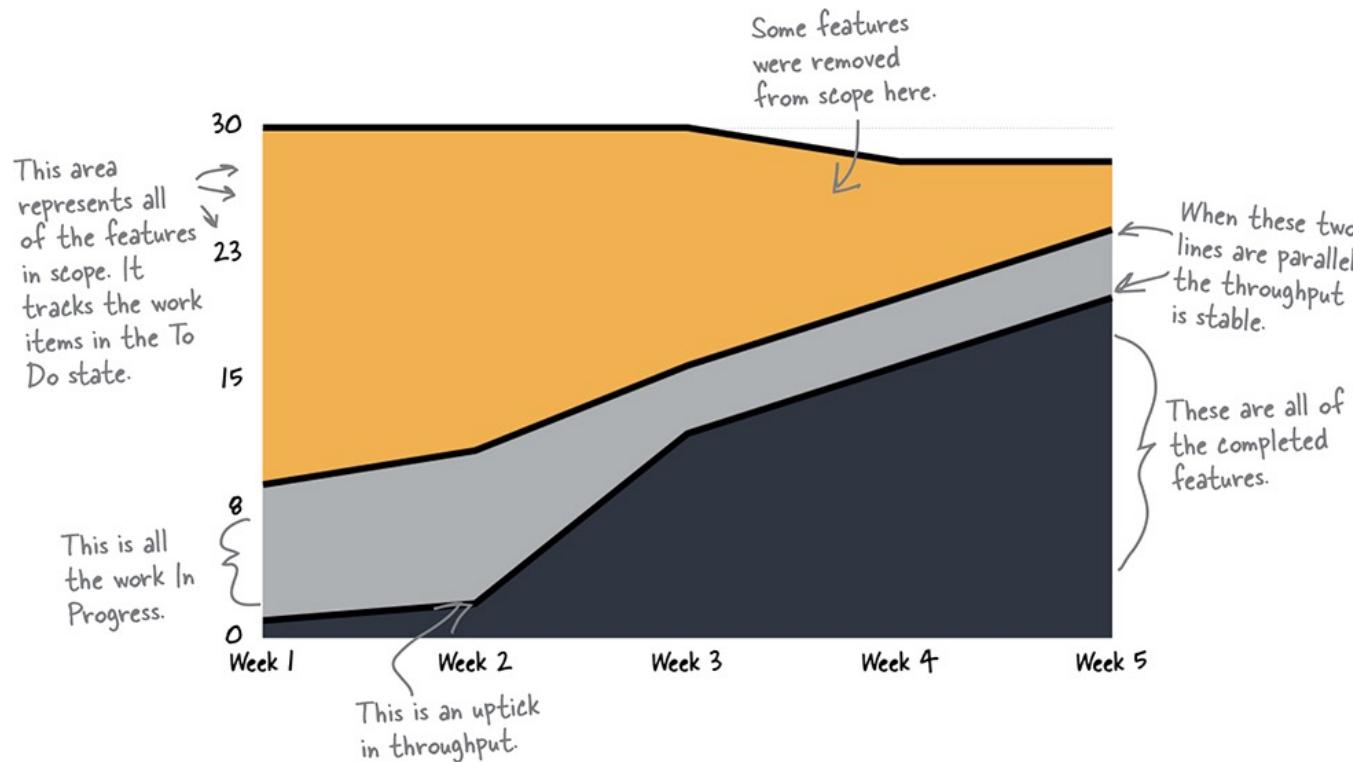
- Kanban is a popular framework used to implement agile and DevOps software development. It requires real-time communication of capacity and full transparency of work
- Kanban is not an Agile software development method (or process) or a software engineering methodology
- Kanban is flow based methodology 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



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

- » Q4 <https://forms.gle/XwFUwDgDsGE4btNL8>
- » Q11 <https://forms.gle/dRMWUqDiEHk8dyiJA>
- » Q12 <https://forms.gle/NKWokLkTZxmemrNB8>
- » Q13 <https://forms.gle/GtKZ38Pdyzxz7wkDA>



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 Life Cycle Process

Life Cycle and Process

PRE-GAME		DEVELOPMENT		RELEASE
PLANNING	STAGING			
<p>Purpose:</p> <ul style="list-style-type: none"> - establish the vision, set expectations, and secure funding <p>Activities:</p> <ul style="list-style-type: none"> - write vision, budget, initial Product Backlog and estimate items - exploratory design and prototypes 	<p>Purpose:</p> <ul style="list-style-type: none"> - identify more requirements and prioritize enough for first iteration <p>Activities:</p> <ul style="list-style-type: none"> - planning - exploratory design and prototypes 	<p>Purpose:</p> <ul style="list-style-type: none"> - implement a product or system ready for release in a series of 30-day iterations (Sprints) <p>Activities:</p> <ul style="list-style-type: none"> - Sprint planning meeting each iteration, defining the Sprint Backlog and estimates - daily Scrum meetings - Sprint Review 		<p>Purpose:</p> <ul style="list-style-type: none"> - operational and functional deployment <p>Activities:</p> <ul style="list-style-type: none"> - documentation - training - marketing & sales - ...

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

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



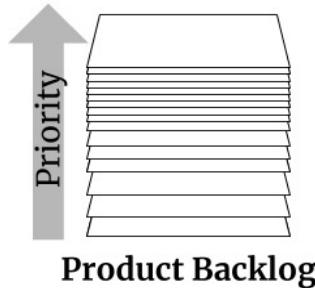
Scrum Master

- 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

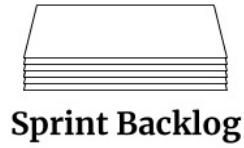
Ref: The Agile Developer's Handbook, by Paul Flewelling, Published by Packt Publishing, 2018

Scrum Artifacts

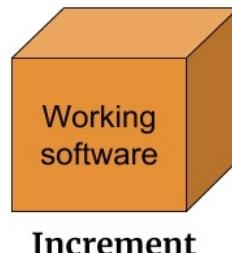
Scrum Artifacts



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

innovate

achieve

lead

Planning

Daily Scrum

Daily Scrum

Daily Scrum

Development

Daily Scrum

Daily Scrum

Daily Scrum

Sprint Review

Retrospective

30 days
2 WKS. TO 4WKS.

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.

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?

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

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.

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

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



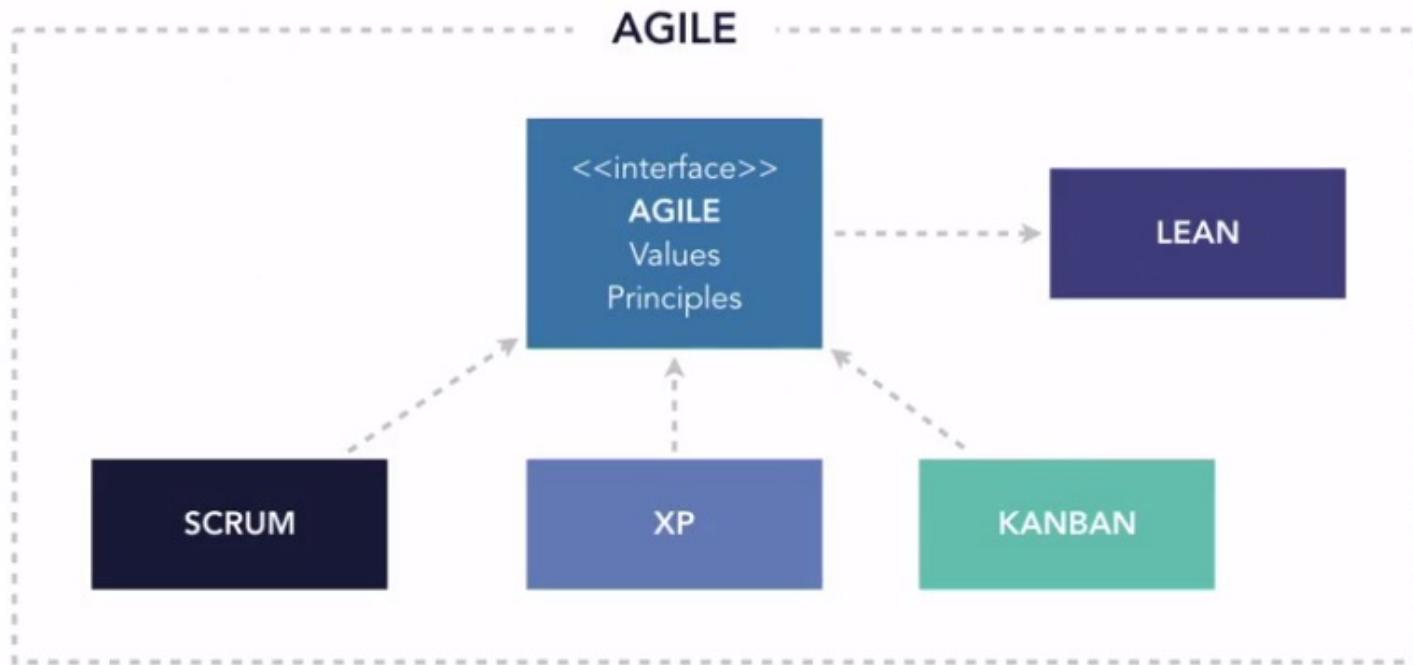
XP- Extreme Programming

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



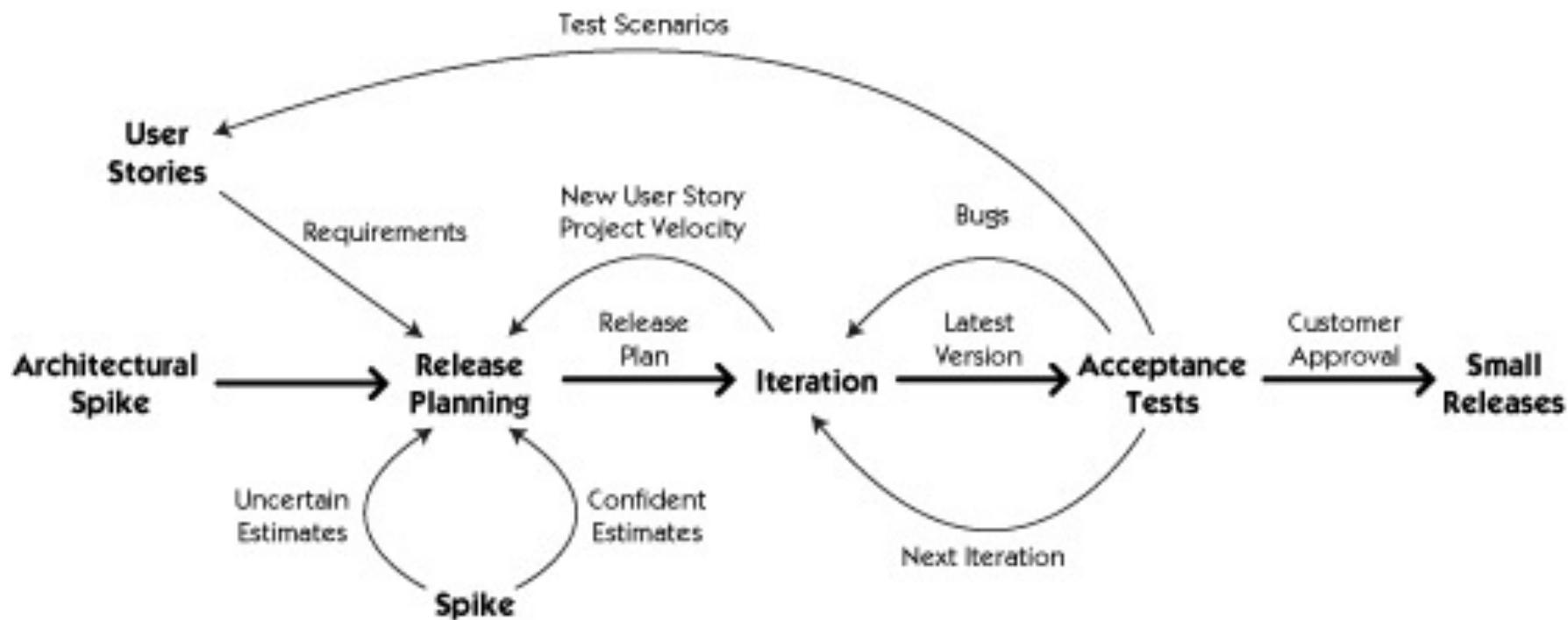
Source :LinkedIn Learning/XP overview

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](https://www.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 ...

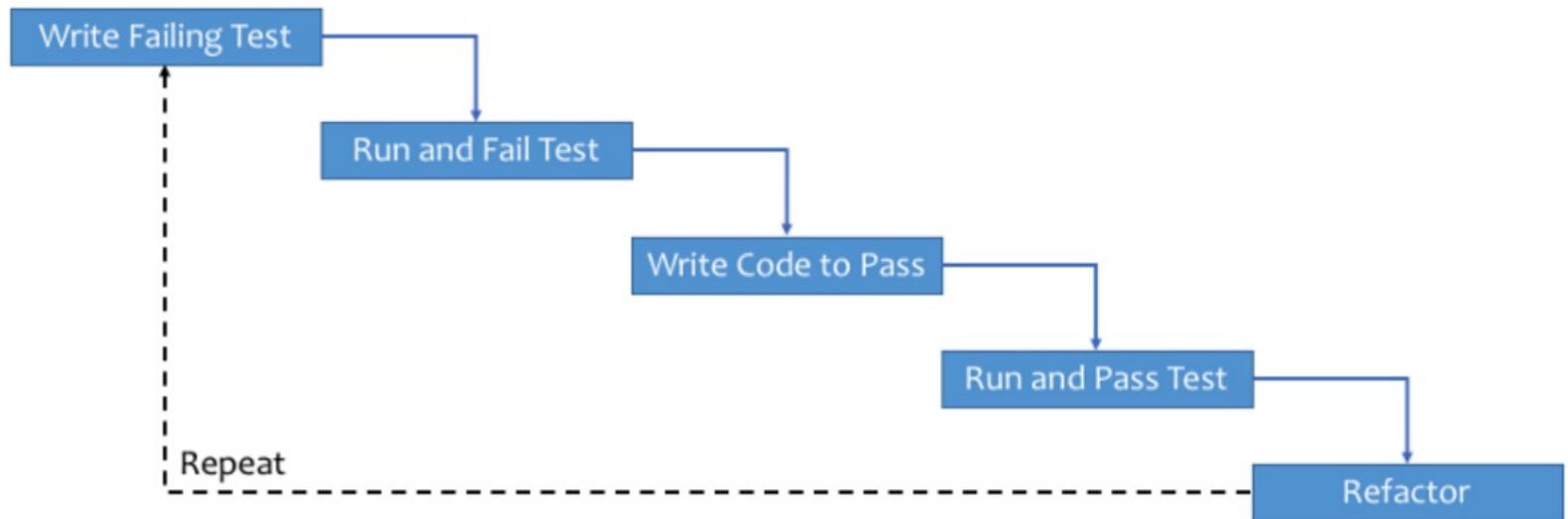
- **Forty-hour week (Energized work)**
 - Teams aim for sustainable phase
 - XP does not forbid overtime, but it has a clear rule – *You can't work a second week of overtime.*
- **On-site customer**
 - A real customer should sit with the team.
 - This person will be someone who will use the system, who 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.



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.





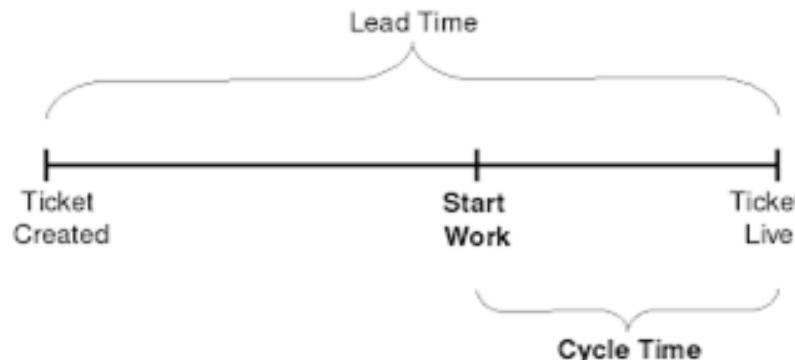
Lean Software Development

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.



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 defer 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.



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 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.

Kanban Core Practices

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

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.

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.

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

How to use Kanban to improve your process

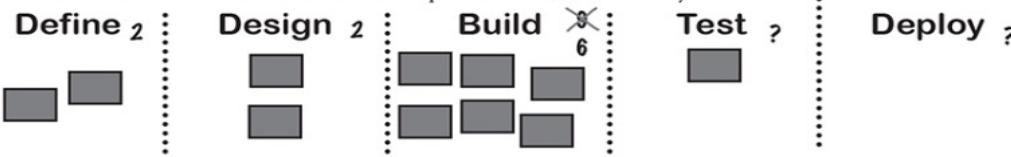
A value stream map is a great way to create this picture! These are the same boxes that you'd see at the top of the map.



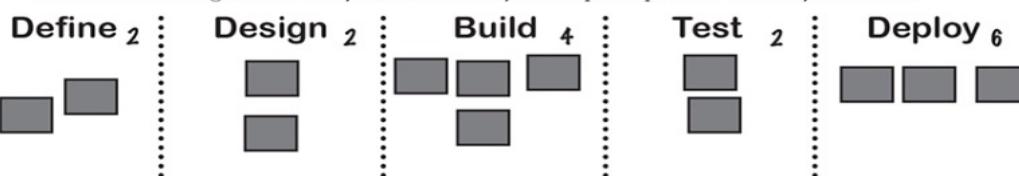
- 1 **Visualize Workflow:** create a picture of the process you're using today.



- 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.

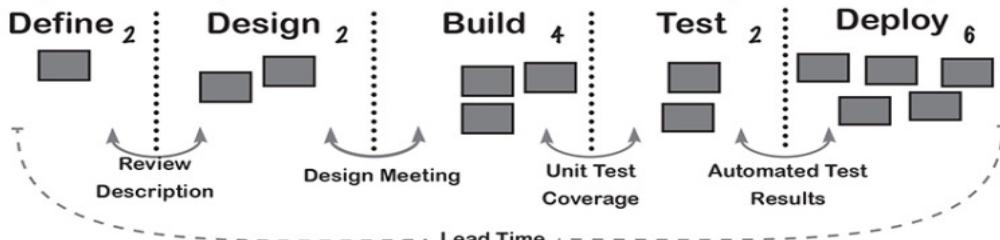


- 3 **Manage Flow:** measure the lead time and see which WIP limits give you the shortest time to delivering features to your clients. Try to keep the pace of delivery constant.



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

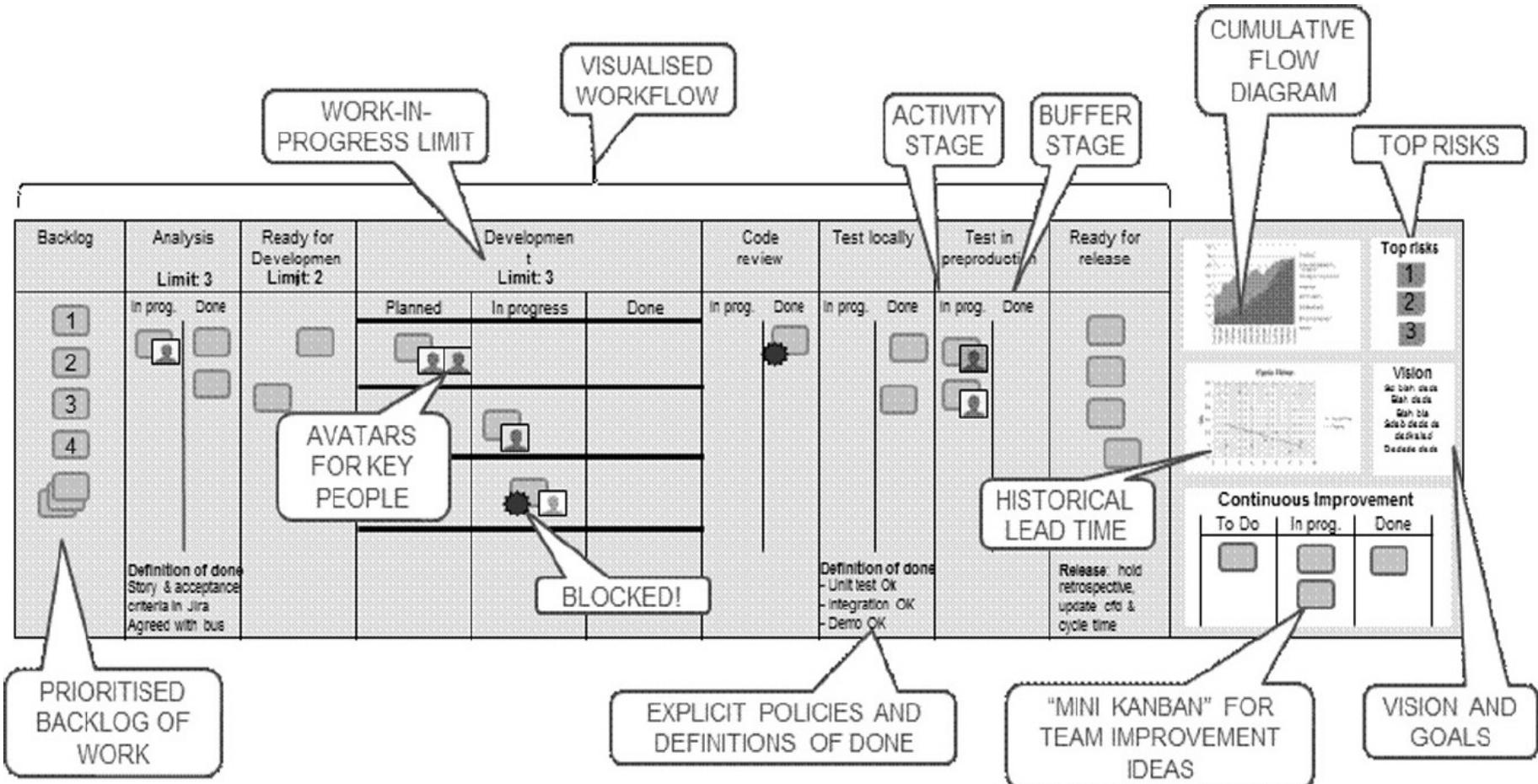
- 5 **Implement Feedback Loops:** for each step in the process create a check to make sure the process is working. Measure lead and cycle time to make sure the process isn't slowing down



- 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

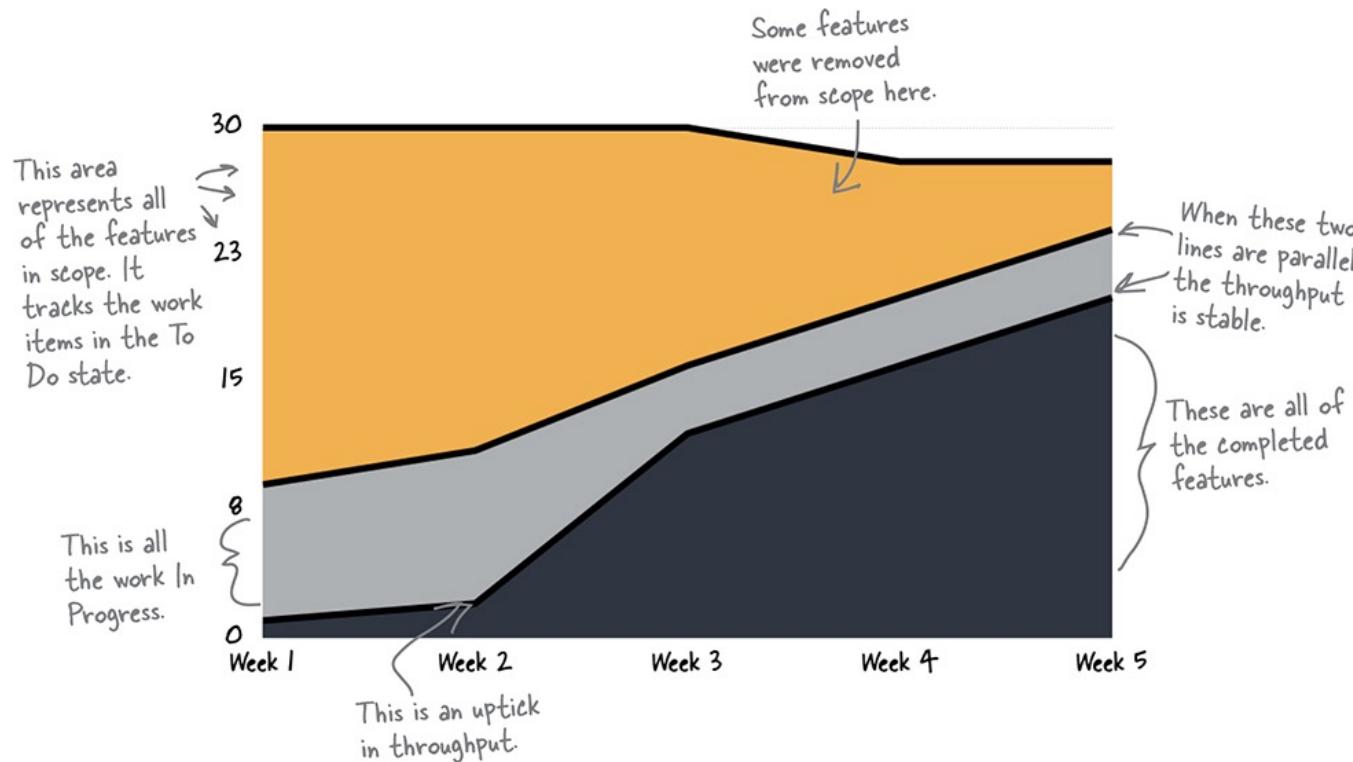
Typical Kanban board



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

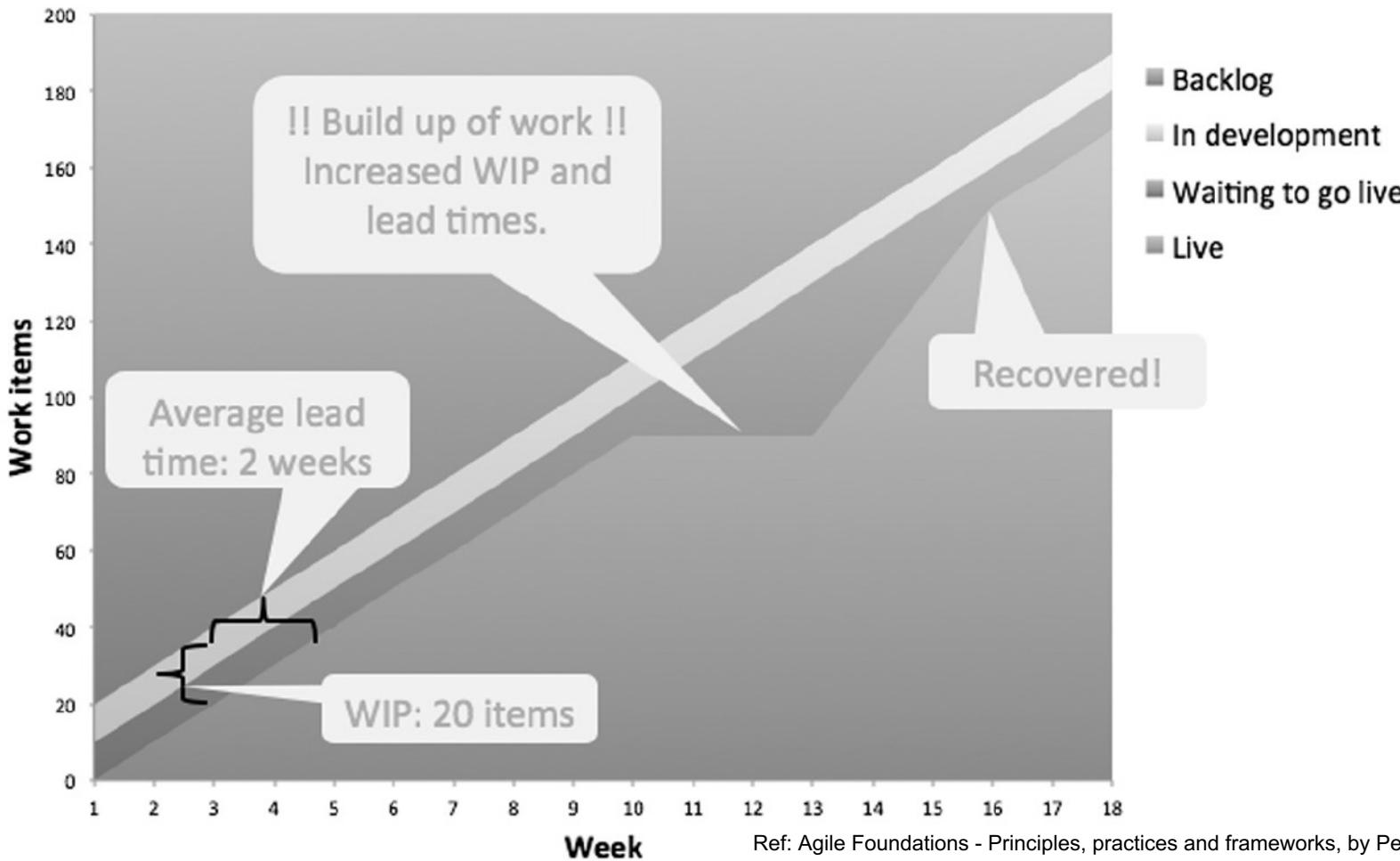
Cumulative flow diagram (Example-1)

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



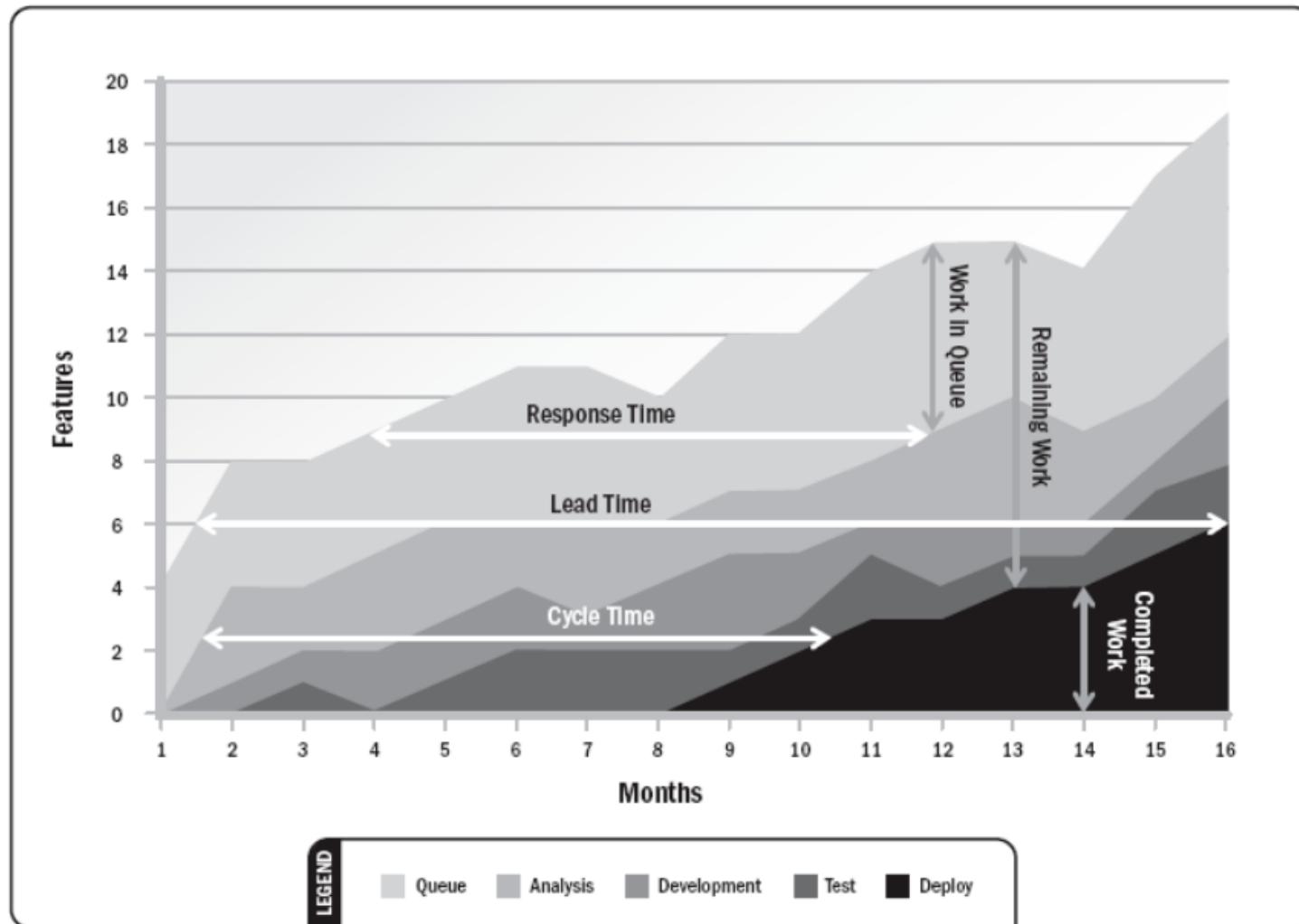
Ref: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

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)





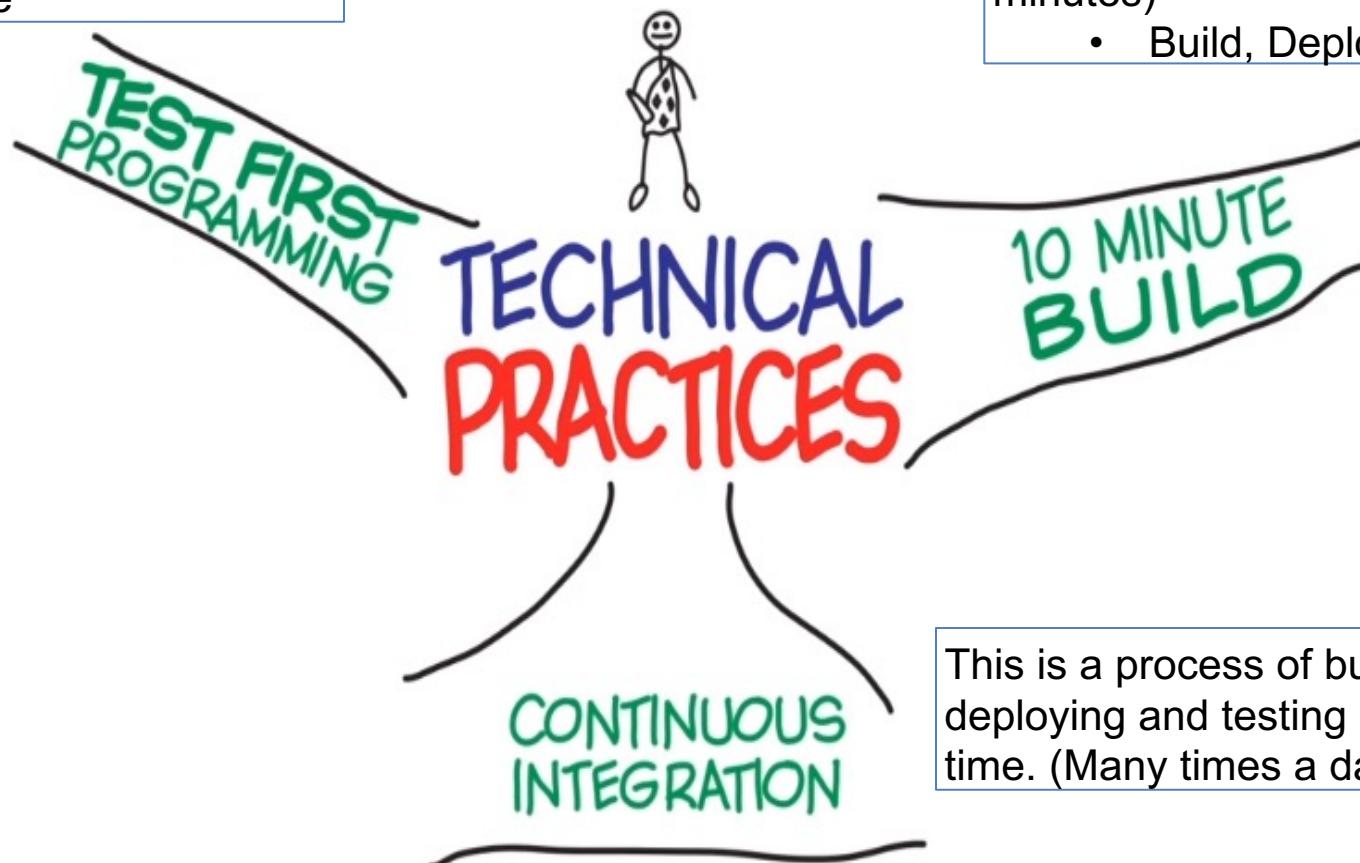
Module-4 XP-Technical Practices

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)

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



Thank you



BITS Pilani
Pilani Campus

BITS Pilani presentation

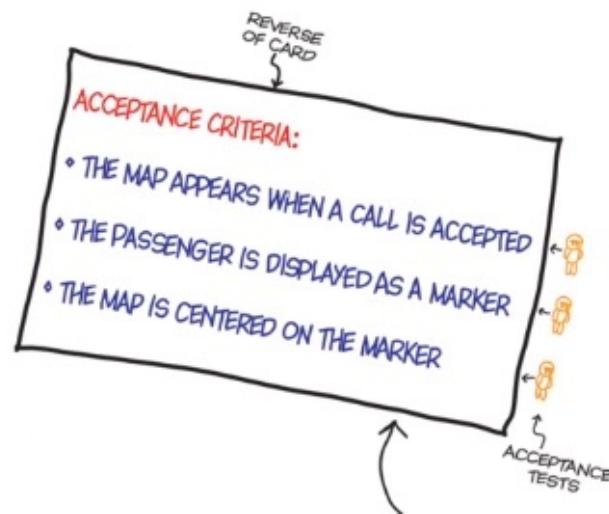
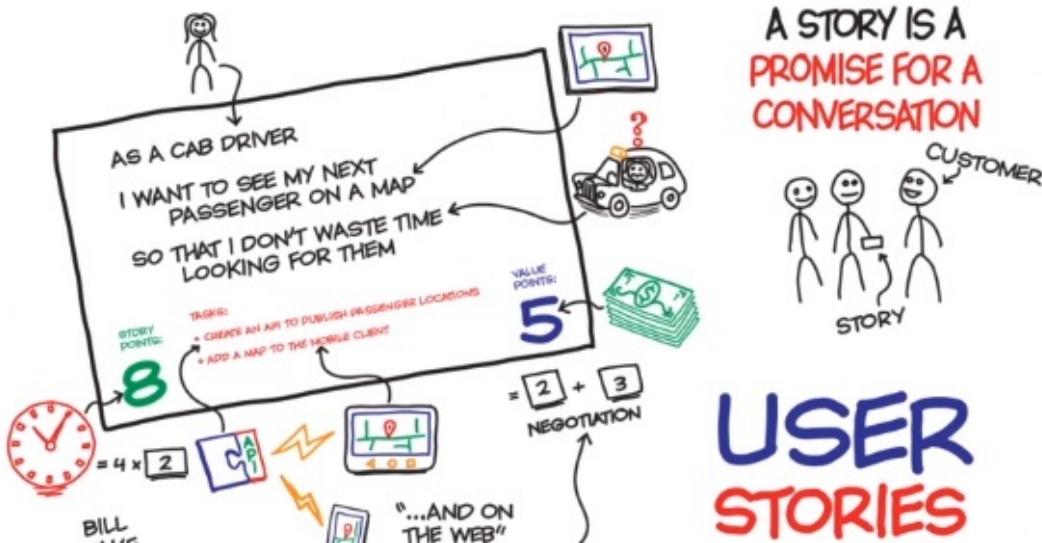
K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



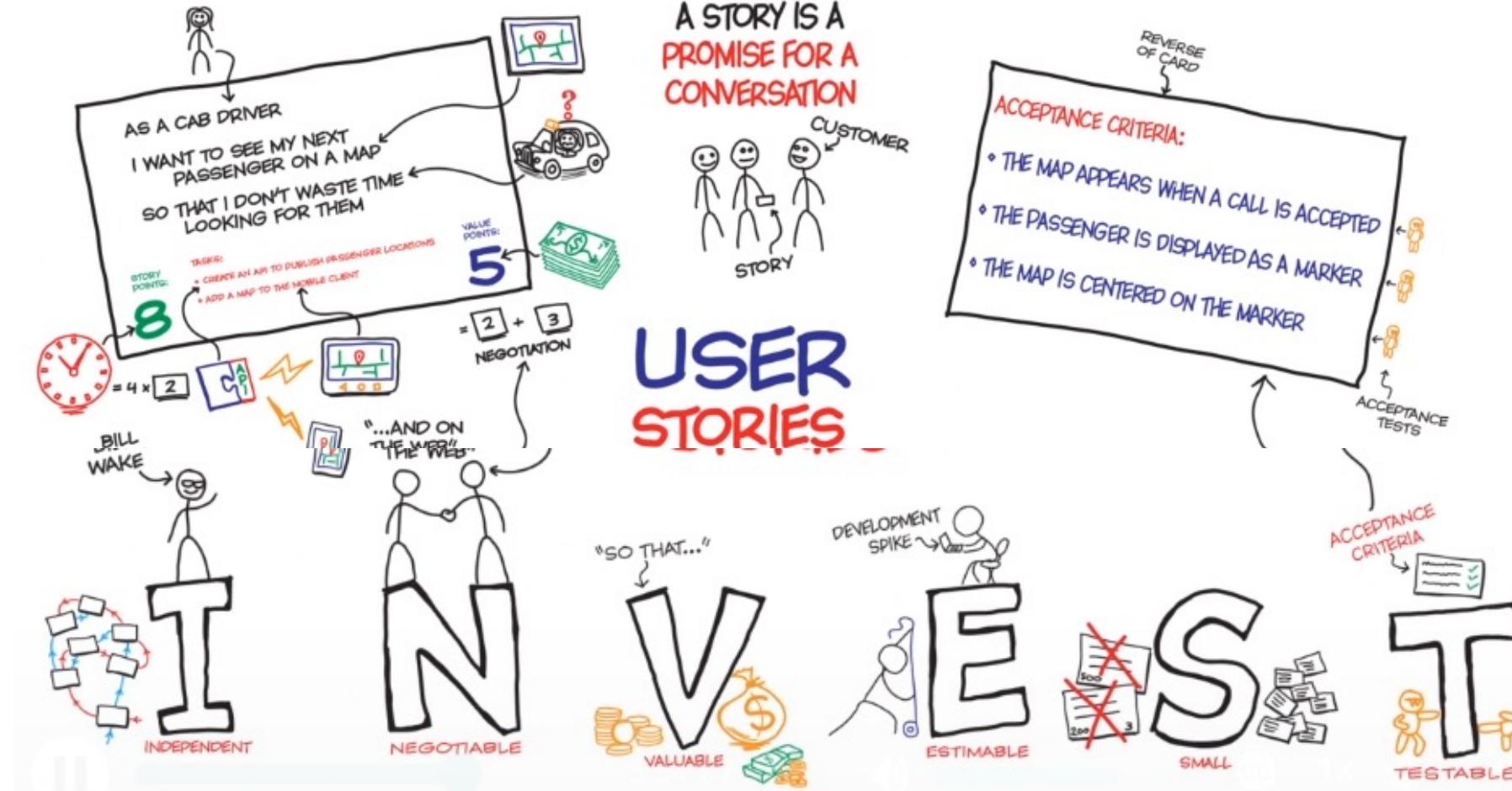
Module-5 Agile Requirements & Agile Estimation

Agile Requirements-1

(Effective User Stories - CCC)

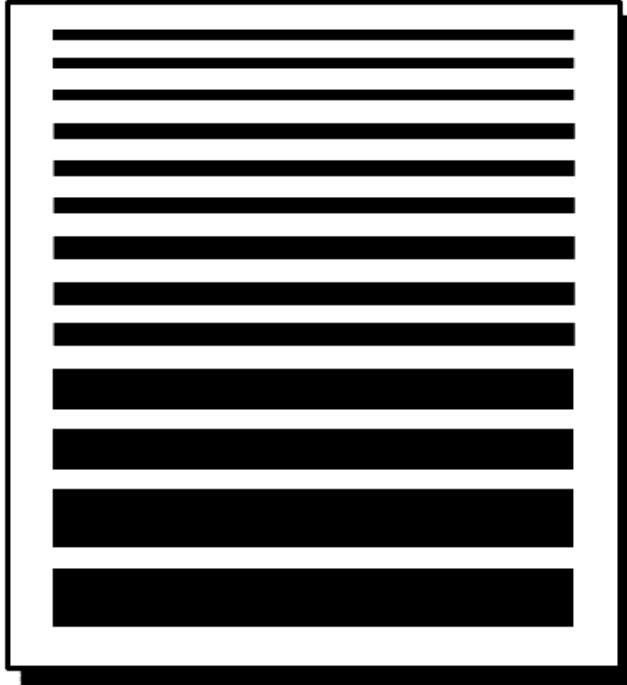


Effective User Stories (CCC & INVEST Guide)



Product Backlog

High Priority



Fine-grained, detailed items ready to be worked on in the next sprint

D E E P
Detailed Emergent Estimatable Prioritized

Large, coarse-grained items

- Themes, Epic

Q&A (Apply INVEST Test)



(Real Estate Project - House Construction)

1. As a home owner, I want my house to have walls so that the house will be wind-proof.
2. As a home owner I want the house to have a roof so that it will be rain proof.
3. As a property developer I want to build a mansion because mansions sell well in this area.
4. As a builder I want to use cement from Acme Corporation because it's run by my sister.
5. As a property developer I want the house to look great so that it will sell

<https://forms.gle/bAjVy4iSD7QEhLGK9>

Q&A (Apply INVEST Test)-Answer

innovate

achieve

lead



Approaches to User Stories Prioritization



- Customer Valued Prioritization
 - Kano Analysis (Satisfied, Dissatisfied, Exciters)
 - MoSCoW Technique (Must have, Should have, Could have, Wont have)
- Relative Prioritization
 - Business Value Vs Risk Vs Effort
 - Relative Weighting Prioritization (Value/Cost)
- Story Mapping



Agile Estimation – Story Point Estimation

Agile Estimation

Absolute Estimation

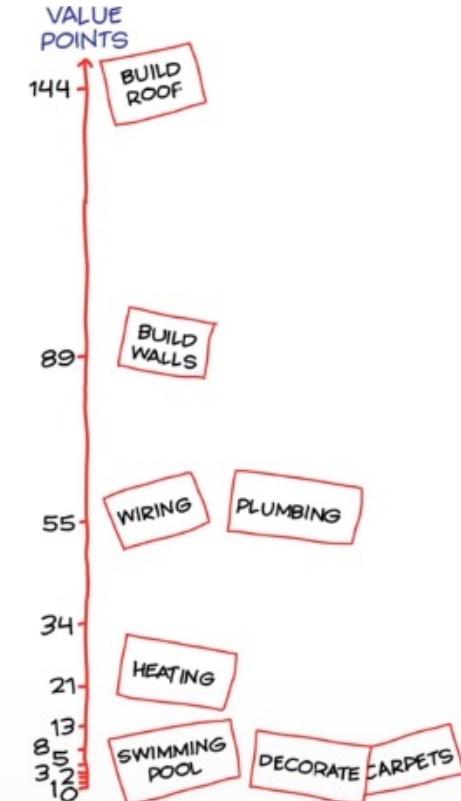
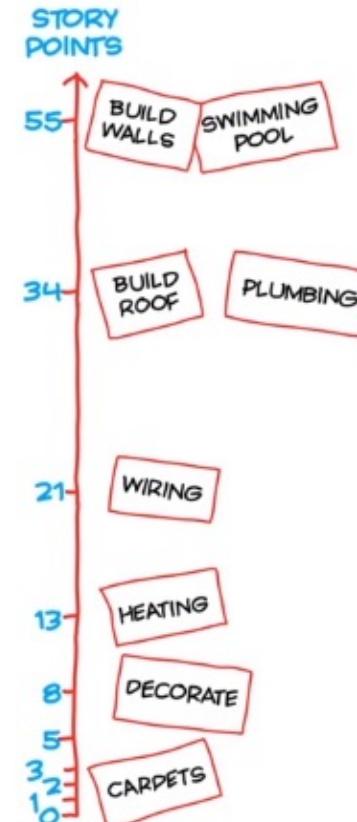
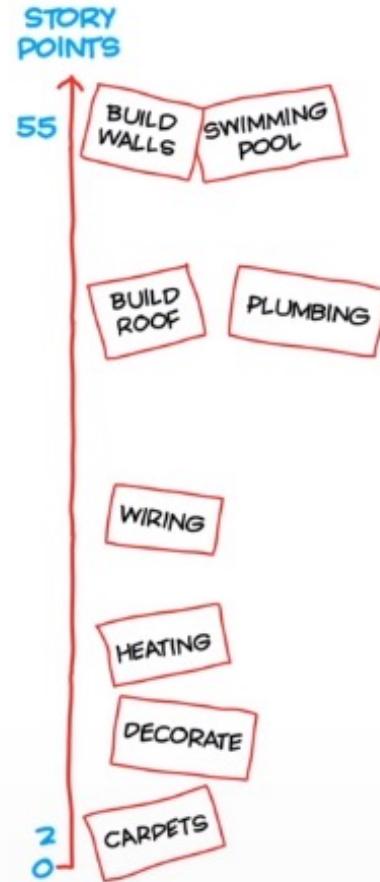
- ✓ Relative Estimation
- ✓ Fibonacci Sequence (1,2,3,5,8,13,21,34,55,
- ✓ Story Point
- ✓ Value Point
- ✓ Velocity
- ✓ Average Velocity
- ✓ BFTB (Bank For The Buck)= Value Point /StoryPoint

Agile Estimation - User Stories



Example-Housing Project

User Stories
Build Walls
Carpets
Decorate
Build Roof
Wiring
Plumbing
Swimming Pool
Pool
Heating



✓ Fibonacci Sequence (1,2,3,5,8,13,21,34,55,)

Agile Estimation

1. Story Point Estimation

	STORY VALUE BFTB		
	34	144	4.2
BUILD ROOF	34	144	4.2
WIRING	21	55	2.6
BUILD WALLS	55	89	
HEATING	13	21	1.6
PLUMBING	34	55	
CARPETS	2	2	1
DECORATE	8	2	0.25
SWIMMING POOL	55	1	0.01

This story needs to go first

ITERATION 1
VELOCITY: 55
VALUE: 89

BUILD WALLS

Cost: \$20000,
Cost/Value= 20000/89 = \$225

ITERATION 2
VELOCITY: 55
VALUE: 199

BUILD ROOF
WIRING

Cost: 199*225=\$44000

ITERATION 3
VELOCITY: 47
VALUE: 76

HEATING
PLUMBING

Cost: 76*225
=\$17000

ITERATION 4
VELOCITY: 10
VALUE: 4

CARPETS
DECORATE

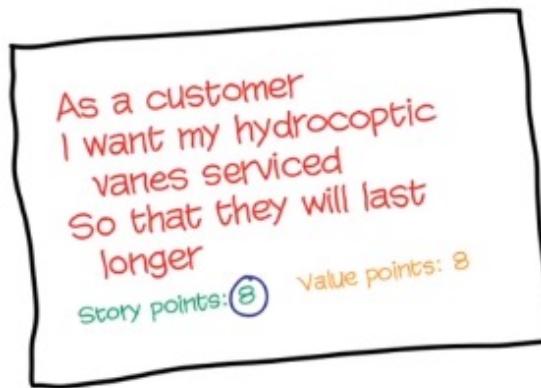
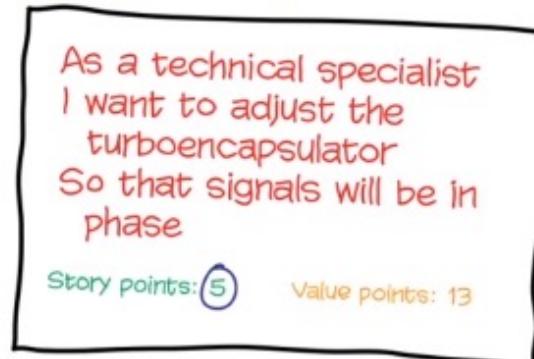
Cost: 4*225=900

Iteration-5
Swimming Pool

Estimation Exercise

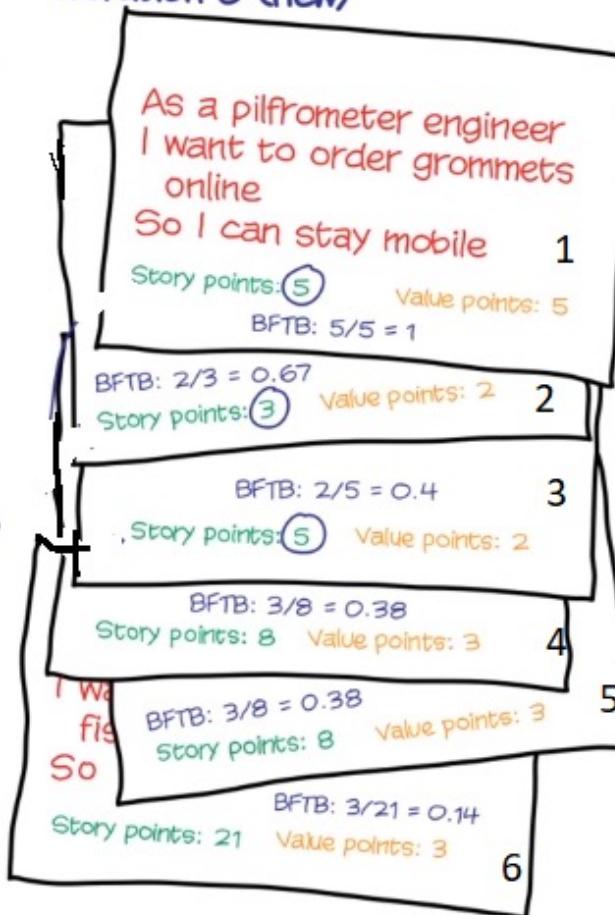
(Iteration Length = 2 weeks (10 Days, 8hrs Per day)

Iteration 7 (complete)



1. Iteration 7 velocity?
2. How long is a story point?
3. Which stories in the next iteration?
4. How long is the rest of the backlog?

Iteration 8 (new)



<https://forms.gle/Zkg8aYpUHdoGa6UV7>

Estimation Exercise

(Iteration Length = 2 weeks (10 Days, 8hrs Per day)

Iteration 7 (complete)

As a technical specialist
I want to adjust the
turboencapsulator
So that signals will be in
phase

Story points: 5 Value points: 13

As a customer
I want my hydrooptic
vanes serviced
So that they will last
longer

Story points: 8 Value points: 8

1. Iteration 7 velocity?

$8+5 = 13$ Story Points

2. How long is a story
point?

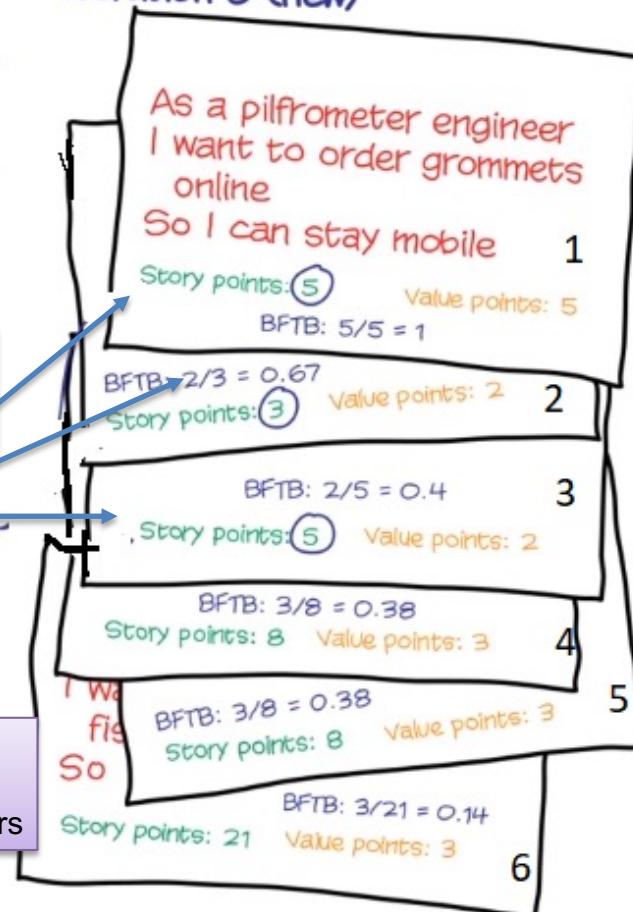
$80 \text{ hrs} = 13 \text{ Story Points}$
 $1 \text{ Story Point} = 80/13$
 $= 6.15 \text{ hrs}$

3. Which stories in the
next iteration?

4. How long is the rest
of the backlog?

Total SP= $5+3+5+8+21$
 $=50$ Points
Time = $50*6.5 \text{ hrs} = 307.5 \text{ hrs}$

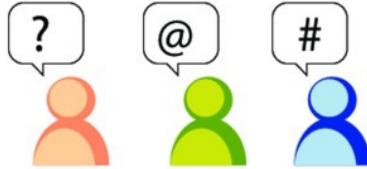
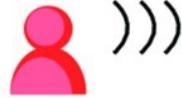
Iteration 8 (new)



Planning Poker

How User Stories are estimated by the team?

1. Customer reads story.



Development team asks questions

2. Team estimates.
This includes testing.



Discussion ...

3. Team discusses.



4. Team estimates again.
Repeat until consensus reached.



Estimator	Round 1	Round 2
Team member-1	3	5
Team member-2	8	5
Team member-3	2	5
Team member-4	5	8

Ref: The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010
Image source: <https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110>



Module – Agile Requirements & Agile Estimation – Additional Notes

Requirements Gathering in Waterfall Method



- In Waterfall model, We tend to describe upfront how the entire product/system will work and document them.
 - PRD or SRS or CRS
- The problem with gathering requirements as documentation isn't one of volume—**it's one of communication**. It's just really easy to misinterpret what someone wrote.
- Other Issues:
 - Lengthy Process (1 to 3 months), Sometime Project wont get started
 - Requirement change is hard, especially late in the cycle
 - Bad guesses and wrong assumptions and so on

Requirements Gathering in Agile



- In Agile Projects, **User Stories** are the main way to track the information or requirements, of the project.
- User Stories tell us about:
 - What customer the wants the team to do?
 - How valuable the work is?
 - How long it is likely to take completed?
- **User stories are fundamental unit** of product development in Agile environments
- User stories describe single feature **which enable rapid iteration.**

Why Write User Stories?

- User stories also enable **empathic design and development** as they are written from the perspective of the end user
- A well-written user story will communicate both how a feature will work and how it will benefit the end-user
- User stories ensure that teams are building features to meet a user goal or need instead of “building stuff to build stuff”

Another User Story format and Examples



Writing User Stories

The user story is written in the following format:

"As a [user], I want to do [x] so that I can accomplish [y]."

For example, "As a Gmail user, I want to be able to attach a photo to an email so that I can share it as part of my message."

Who?

What?

Why?

Note: If the user story involves a frontend (user-facing) design component, the design files should be included with the user story

Ref: Writing User Stories By Ryan Harper O'Reilly Media -Video

User Story Examples

A user story for returning images in Google Image search.

“As a Google Images user, when I search for an image I want to see images that match my query so that I can find the image for which I’m looking.”

- Note that, user story does not focus on how the images will be returned or displayed, but rather on end user’s goal and needs.

Writing Acceptance Criteria for User Stories



- The second part of the user story , **the acceptance criteria**, explains how the feature will work.
- The acceptance criteria consists of series of **boolean statements (true or false)**, such as “ When [x] happens, [y] should happen

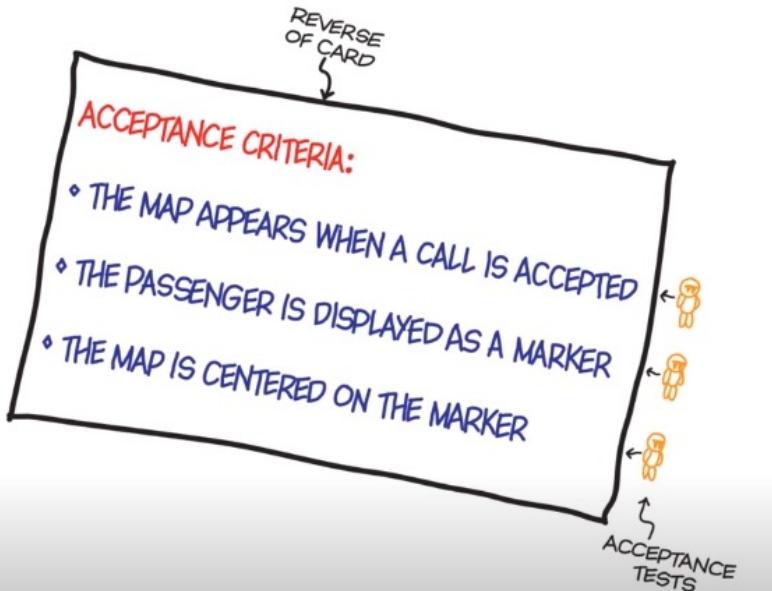
Acceptance Criteria/Conditions of Satisfaction



- Creating clear acceptance criteria reduces ambiguity for the development team and allows a feature to be easily tested.
- Acceptance criteria can also serve as a form of documentation once a feature has gone live, providing a written record of how a feature is intended to work.

Examples Acceptance Criteria

Back of the User Story card



For example for an Gmail user,

“ When the user saves an email that has not been sent, it should be stored in the user’s Drafts folder”

Outcome: Email stored in Drafts (Yes) or Not (No)

Acceptance Criteria for Google Image search

“As a Google Images user, when I search for an image I want to see images that match my query so that I can find the image for which I’m looking.”

Some possible acceptance criteria:

“When the user inputs a query, such as ‘cat’, the image results should be returned in order of relevance.”

“The image results should be returned in rows.”

“When the user clicks/taps on an image, a detail view of that image should appear between that image’s row and the row below.”

1. Relevance: The image most related to user query appears first, the images least related to user query appears last
2. Rows: Layout function
3. Tap on Image: how user will interact with the image

Elements of Good User Stories



- Bill Wake came up with the **INVEST** acronym for good user story.
- Good user stories also have the following characteristics:
 - **I**ndependent
 - **N**egotiable
 - **V**aluable
 - **E**stimatable
 - **S**mall
 - **T**estable

The 3 Cs- Story Process

- In the book *Extreme Programming Installed*, Ron Jeffries et al. (Addison-Wesley Longman Publishing) **describe the story process** best:

• Card:

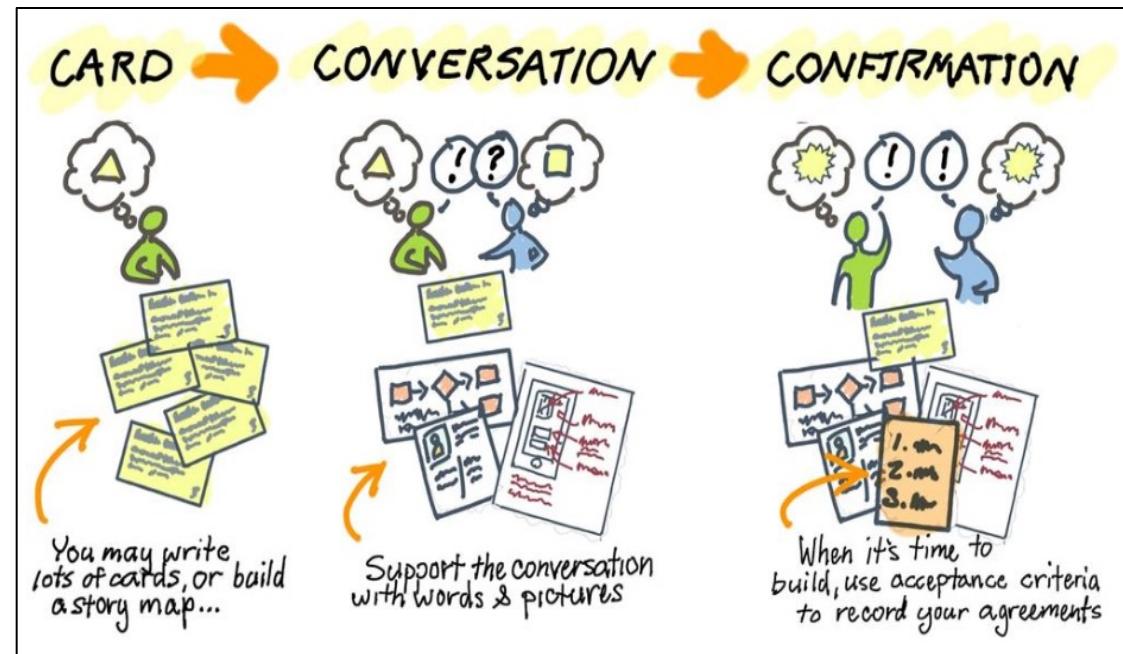
- Write what you'd like to see in the software on a bunch of index cards.

• Conversation

- Get together and have a rich conversation about what software to build.

• Confirmation

- Together agree on how you'll confirm that the software is done.



Difference Between User Story, Bugs, Constraints



Features vs. Bugs

Unlike features, bugs (problems with how a live feature is working) are not written using the user story format.

Instead, bugs are documented with a descriptive title and clear steps for how to reproduce the issue.

For example, if tapping a pause button on a video player in Mobile Safari wasn't working, the bug could be written as follows:

Title: Tapping Pause on Video Player Doesn't Work

1. In Mobile Safari on iOS 10.3 / iPhone 6S, go to myvideoplayer.com/videoexample.
2. The video will play automatically on mute.
3. Tap the pause button. The pause button does not become a play button, and the video is not paused.

Constraints

Story: Website must be super fast

Story: Design should look really good
– Constraint Card

- **Stories like these, we call constraints.**
- But they are important because they describe characteristics our customers would like to see in their software.
- For example, The Website must be super fast can be written like this.

All web pages must load in less than 2 sec

A constraint card

How to take care of Frontend Design?

- If a user story includes a new design, the design files illustrating that design (including any interactions) are included with user story.
 - Design tools: Adobe Photoshop, Sketch and Invision
 - Design files: Wireframes, Mockup, Prototypes

<https://justcoded.com/blog/wireframe-mockup-and-prototype-whats-the-difference/>

Story Grooming Meeting

- Story grooming meetings give the engineering team a chance to review user stories **before they are scheduled for a development sprint.**
- Story grooming meetings are **critical for securing the development team's buy-in.**
- The team is given a chance to ask the questions that would normally arise during sprint planning:
 - What should we do if the user enters invalid data here?
 - Are all users allowed to access this part of the system?
 - What happens if...?
- The team provides feedback on the feasibility, viability, and size of each feature and may provide alternate solutions/ or identify previously unforeseen prerequisites or roadblocks for the user needs identified in the user stories.



Agile Estimation

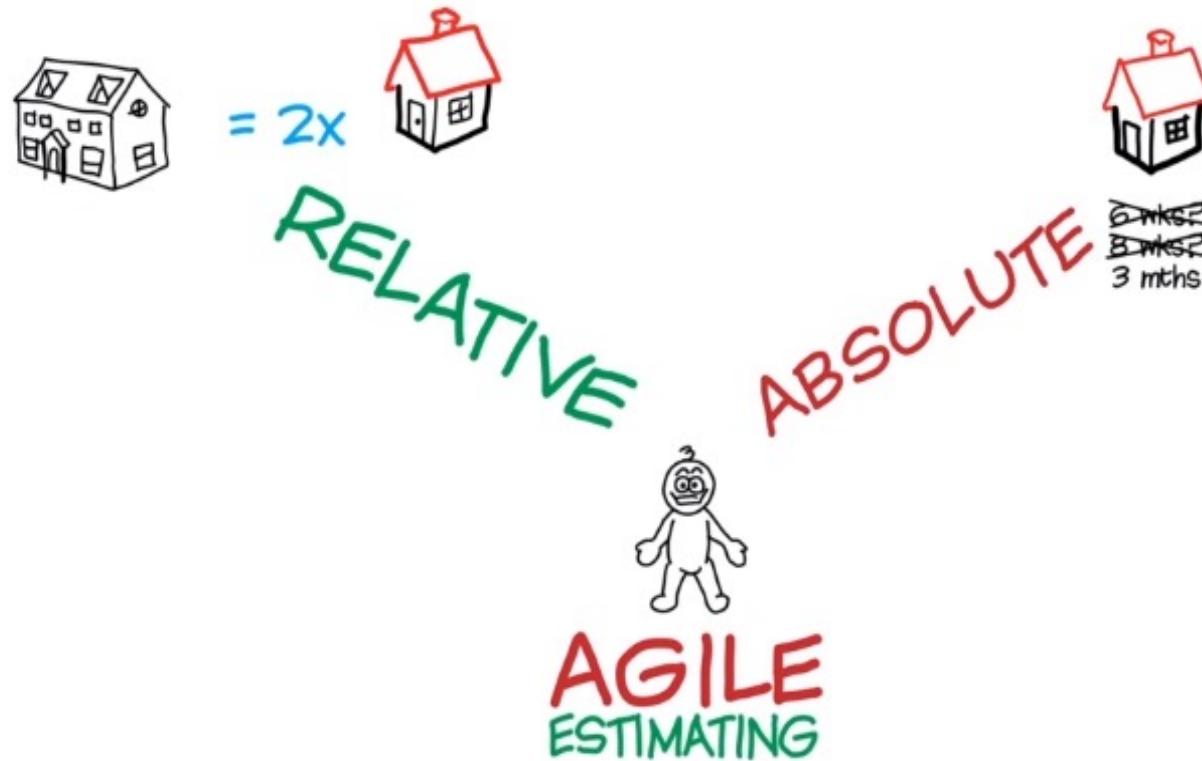
Absolute Estimation

- We estimate our work in hours, days, and weeks.
- We use all the knowledge and experience at hand to make a guess about the amount of time it is going to take.
- Estimation is **approximate and not accurate**
- Absolute estimation:
 - Estimating in absolute values (Examples, days, weeks, months or KMs, Miles)
 - **Absolute values are not easier to judge**
 - **People are not good at absolute estimation**

Relative Estimation

- Agile team use relative estimation.
- Relative estimating compares what you don't know against what you do know.
 - For example, You might not be able to guess how much a truck weighs, but if you saw the truck, you can probably guess how many cars equal a truck.
 - A “customer search” story, as it probably involves double the effort to implement than a simple “Login user” story.
- Relative estimation is easier to judge than absolute values.
 - This means judging how big or complex tasks are with respect to other tasks
- This estimation is not designed to be precise.
 - But that doesn't mean it's useless. Instead it gives you a starting point, a way to start the discussion on what it takes to deliver your stories.

Absolute vs Relative Estimation



As an analogy, it is much easier to say that Delhi to Bangalore is twice the distance of Mumbai to Bangalore than saying that the distance from Delhi to Bangalore is 2061 kms.

Why Agile team use Relative Estimation?

1. Relative estimation takes away from the false comfort of precision.
 - The team is accepting the fact that the estimates will be imprecise.
 - That way we can start talking about what it takes to deliver this story instead spending too much time on estimates.
2. Agile uses relative estimating is that it keeps the team from **confusing estimates from commitments**.
 - An estimate is the useful information you might give a co-worker. A commitment is something that you usually give to a supervisor. An estimate is a best guess. A commitment is often a worst case scenario. That's why for Agile planning, you want estimates and not commitments.
3. Relative sizing across stories tends to be much more accurate over a larger sample, than trying to estimate each individual story for the effort (in hours) involved.

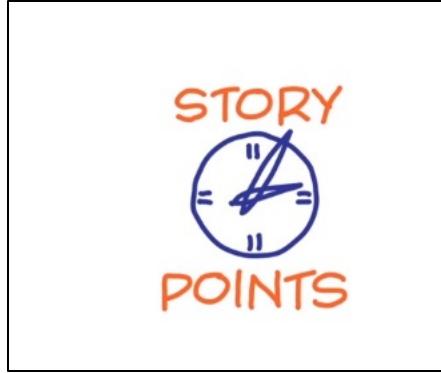


Story Point Estimation

Story Point for Estimation

- In Agile, we use relative estimation
- We do this by comparing the time to take one story vs time to take another story without using absolute estimates
- We do this by using Story points.
- We will have an exponential number sequence.
 - Something like 1,2,3,5,8, 13 These are the points for each of the stories.
- When we estimate with story points, we assign a point value to each item.
 - **The raw values we assign are unimportant.** What matters are the ***relative values***. A story that is assigned a 2 should be twice as much as a story that is assigned a 1. A 2 point story is 2/3 of 3 point story.
 - Instead of assigning 1, 2 and 3, that team could instead have assigned 100, 200 and 300. Or 1 million, 2 million and 3 million. It is the ***ratios that matter***, not the actual numbers.

What does a Story Point represent ?



- **Represents the amount of effort or fixed period of time** required to implement a user story. (**Size**)
- Story Point is not an estimate of the amount of time it takes to implement a Story.
- Some argue that it is a **measure of complexity**, but that is only true if the complexity or risk involved in implementing a user story translates into the effort involved in implementing it.

Fibonacci series as Story points

- The most common way is to estimate a user story is to use the **Fibonacci series** (1, 2, 3, 5, 8, 13, 21, 34, 55..... with each number the sum of the preceding numbers.
- Why Fibonacci?
 - It's because numbers that are too close to one another are impossible to distinguish as estimates.
- In Fibonacci series, after the 2 (which is 100% bigger than one), each number is about **60% larger** than the preceding value.

Predictability of User Stories Estimation

- Small stories tend to result in a more accurate and reliable estimates.
- Small stories reduces variability and improves predictability.
- So, a 13 or 20-point story is likely much less predictable than several 2, 3, or 5-point stories.
- Relative story point estimates using the Fibonacci sequence are, by design, increasingly **less accurate for larger estimates** – like the “cone of uncertainty”

Velocity

- **Velocity** = Number of story points the team can deliver in an iteration/Sprint (OR)
- **Calculating Velocity:**

$$1. \text{Average} = (16+15+17+20) / 4 = 17 \text{ Story points}$$

Sprints	Number of Story points Delivered
Sprint-1	16
Sprint-2	15
Sprint-3	17
Sprint-4	20

Iterations Completed	Low Multiplier	High Multiplier
1	0.6	1.60
2	0.8	1.25
3	0.85	1.15
4 or more	0.90	1.10

2. Give it as a range

- Velocity is a rolling average. That means that the velocity may increase or decrease depending on what happens with the team.
- After some iterations the velocity will become stable.

Examples from Software Development- Story Points are relative

1 –QUICK TO DELIVER AND MINIMAL COMPLEXITY. AN HOUR

Example: add field to a form

2 –QUICK TO DELIVER AND SOME COMPLEXITY. MULTIPLE HOURS

Example: Add parameter to form, validation, storage

3 –MODERATE TIME TO DELIVER, MODERATE COMPLEXITY, POSSIBLE
UNKNOWNs

Example: Migrate somewhat complex static CSS into a CSS pre-processor

5 –LONGER TIME TO DELIVER, HIGH COMPLEXITY, LIKELY UNKNOWNs

Example: Integrate with third-party API for pushing/pulling data, and link to user profiles in
platform

8 –LONG TIME TO DELIVER, HIGH COMPLEXITY, CRITICAL UNKNOWNs

Example: Overhaul the layout/HTML/CSS/JS of a web application

13 –LONG TIME TO DELIVERY, HIGH COMPLEXITY, MANY CRITICAL UNKNOWNs

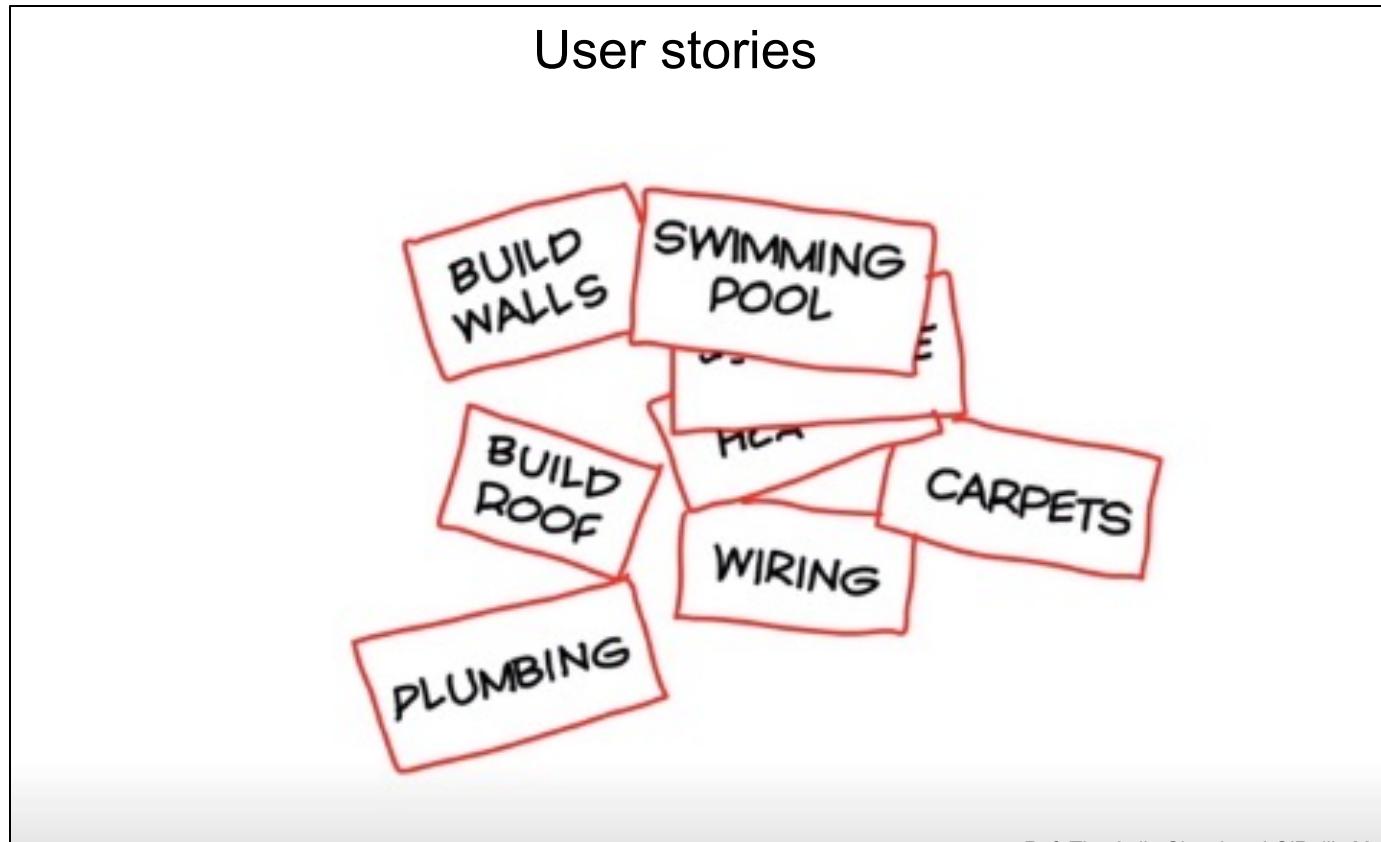
Example: Migrate application from an outdated data store to new DB technology and ORM

How User Stories are estimated by the team?

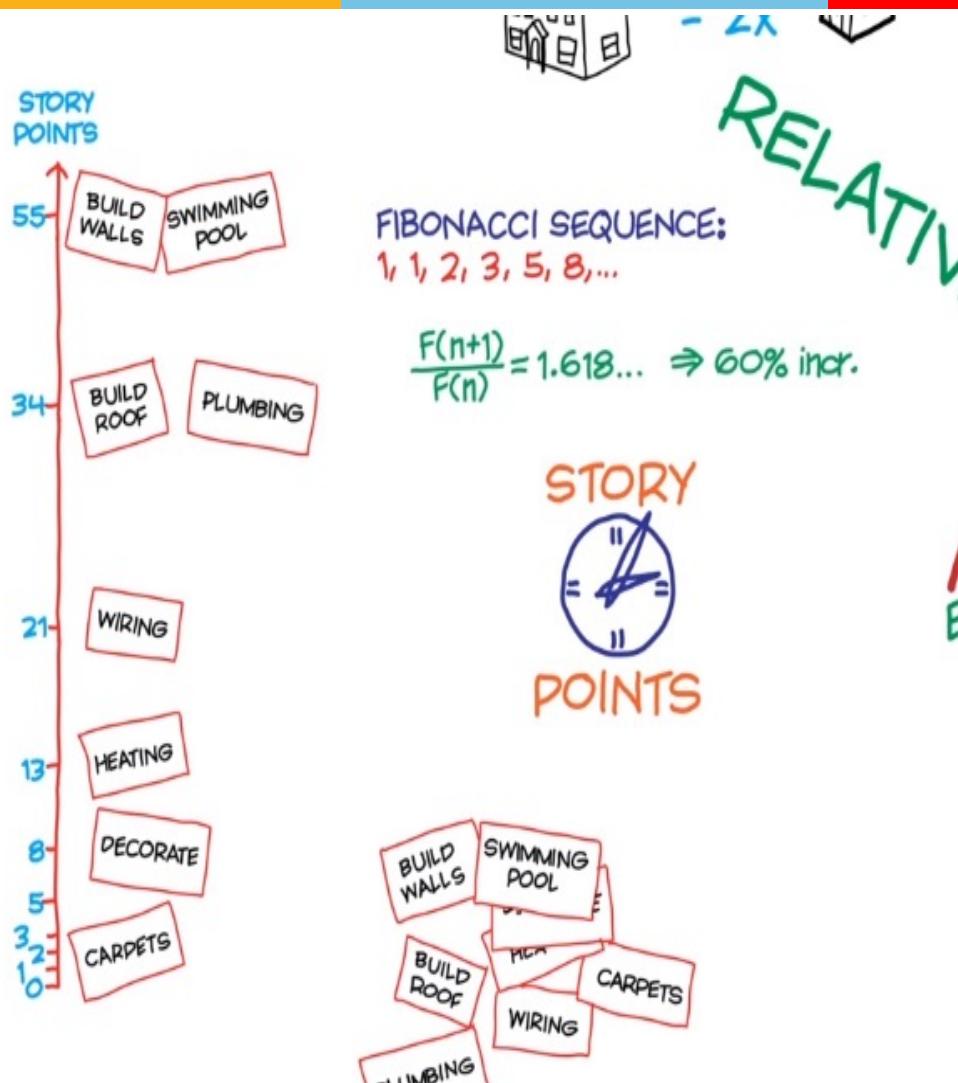
- One method is to play **Planning poker** game.
 - Planning poker helps give everyone a voice.
 - Combining of individual estimates through group discussion leads to better estimates
 - Combat Groupthink-meaning, the way that people tend to agree with the most popular idea.
- Planning poker is a game where the development team estimates stories individually first (using a deck of cards with numbers like **1, 2, 3, 5, 13** ...on them) and then compares the results collectively together after.
- If everyone's estimate is more or less the same, the estimate is kept. If there are differences, however, the team discusses them and estimates again until consensus is reached.

Story Point Estimation – An Example

Project : Build a House, Suppose we have the following bunch of user stories to be estimated. How do we start?



Story Point Estimation – Example



Steps:

1. The team may start with the smallest - Carpets Story , assign 2 points.
2. Next, for example, we may discuss the Build wall story. We agree on, it is 30 times larger than Carpet story. Hence, we assign 55 points in Fibonacci scale.
3. Then relatively, assign story points to other stories, Decorate, Heating etc....

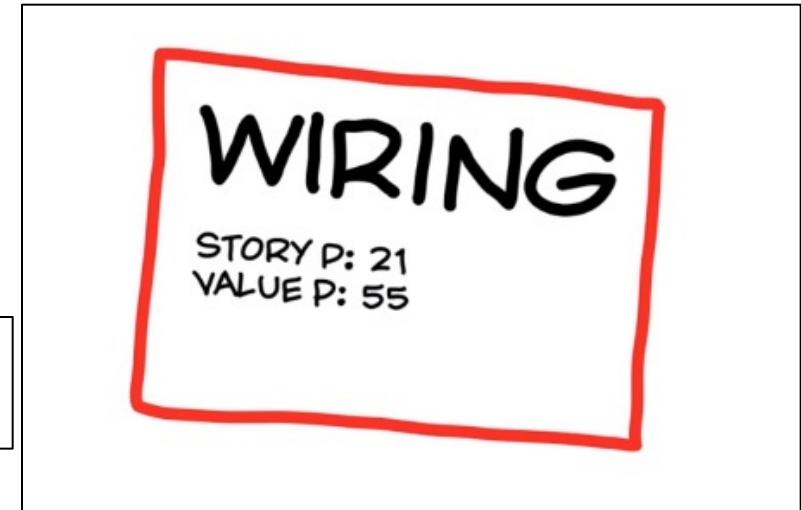
At this point, We do not know the effort of each story

Value Point

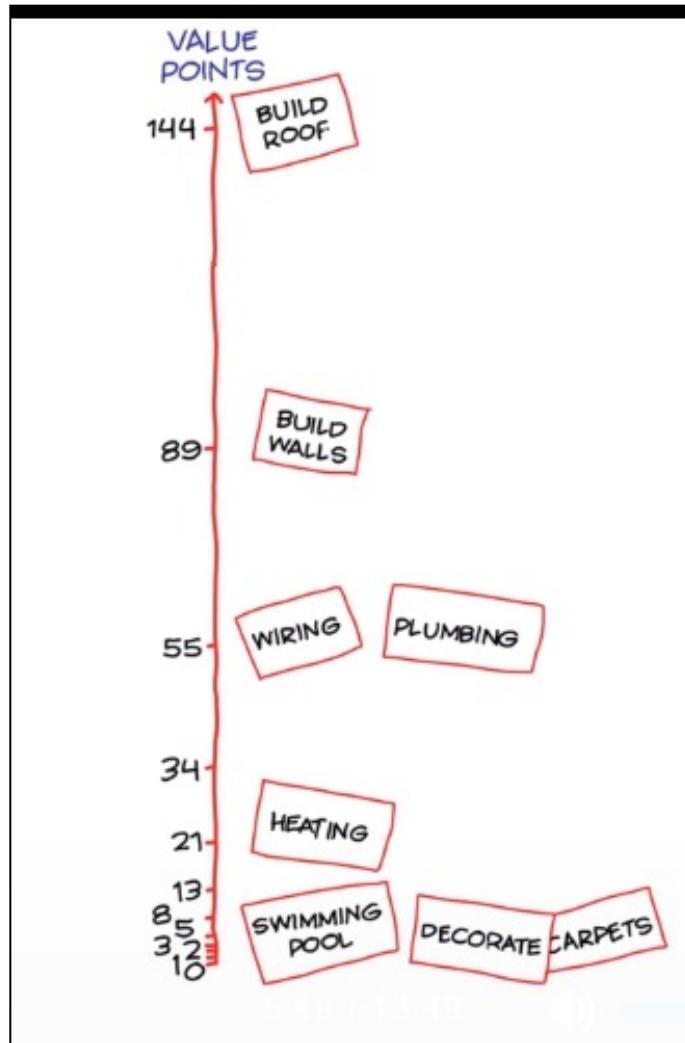
- A user Story has two estimates.
 - Story point – estimate of time
 - Value Point – estimate of value
- Developers , the people who do the work, estimate User Stories in Story points,
- Customer / Product owner estimates User Stories in Value Point, in the same way.

Story Card
Story Point : Amount of time/effort
Value Point : Worth to Customer

Agile is about delivering value early.



Value Point Estimation – use the previous example

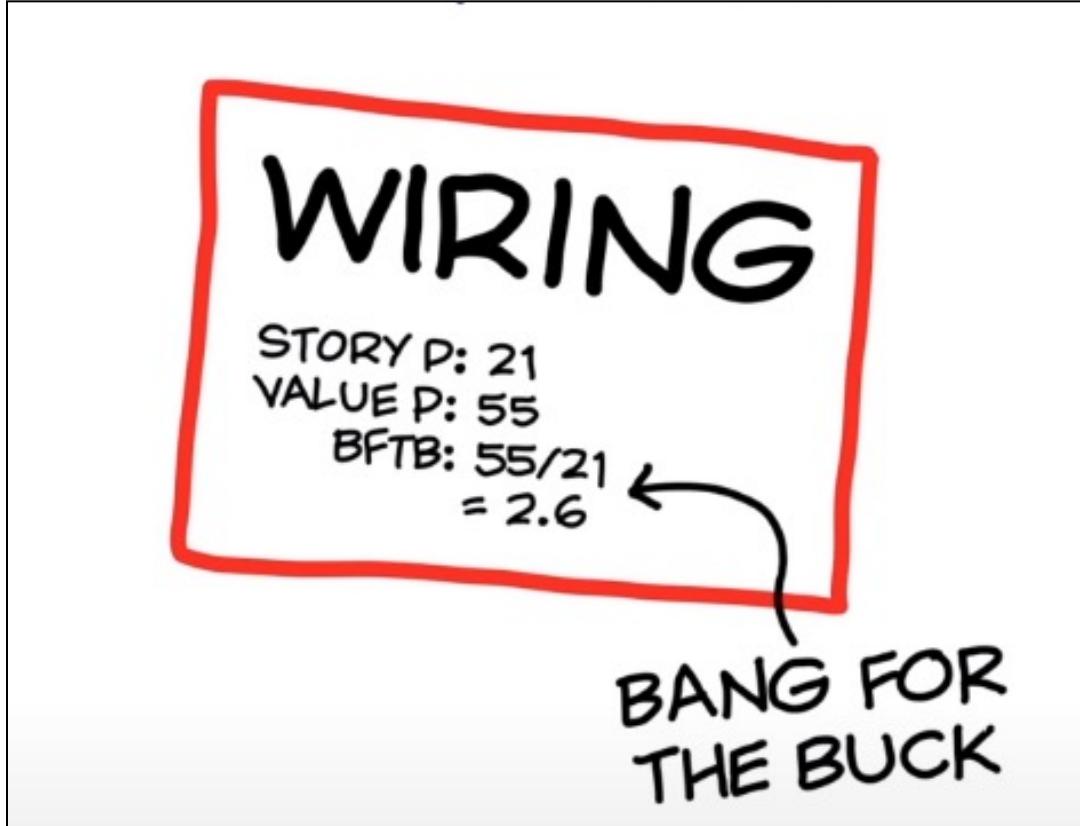


Highest Valued Stories at the top of Product backlog

Relatively Valued Stories

Lowest Valued Stories at the bottom

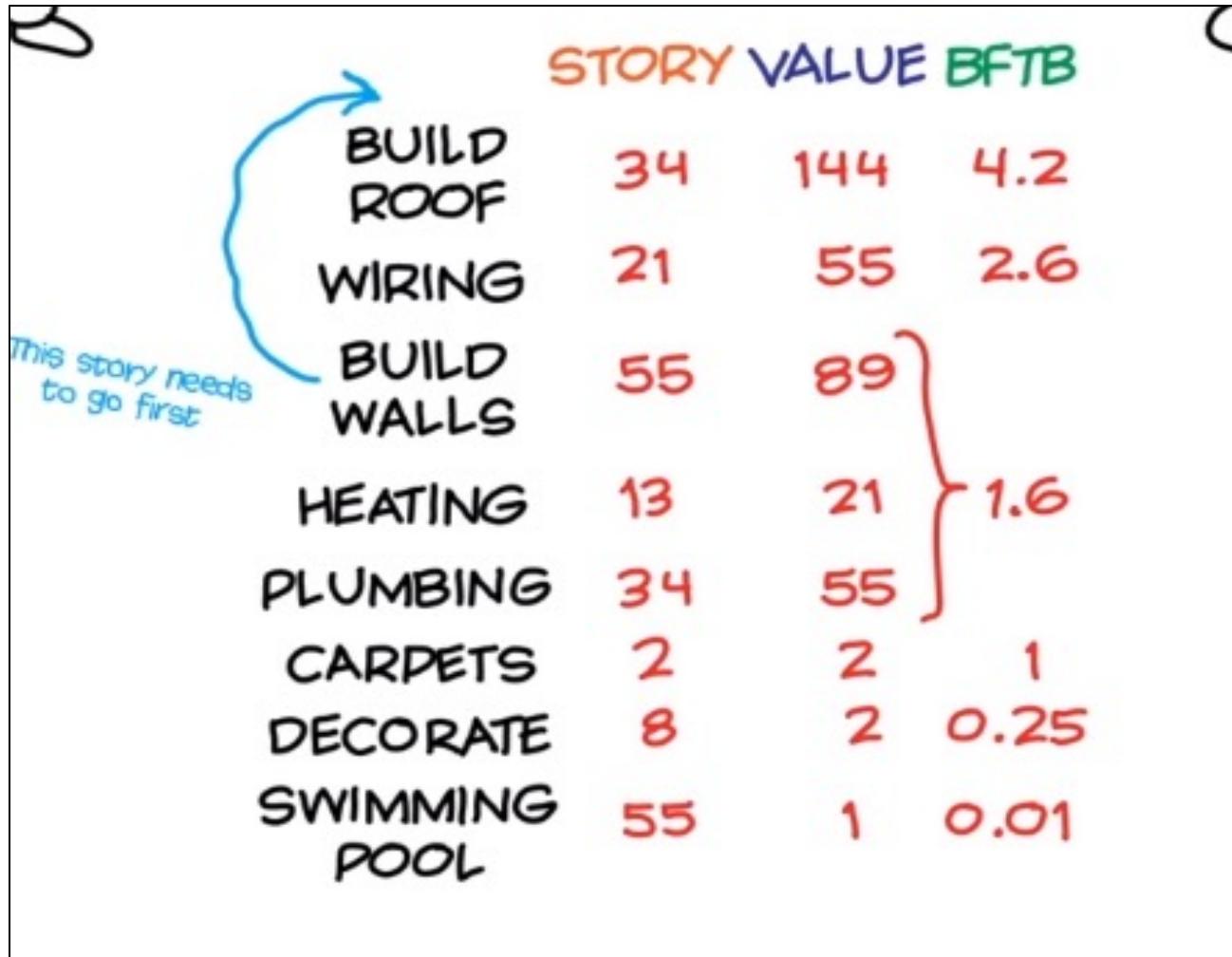
Bang For the Buck (BFTB) (OR) Priority



BFTB = Value Point divided by Story Point for each story

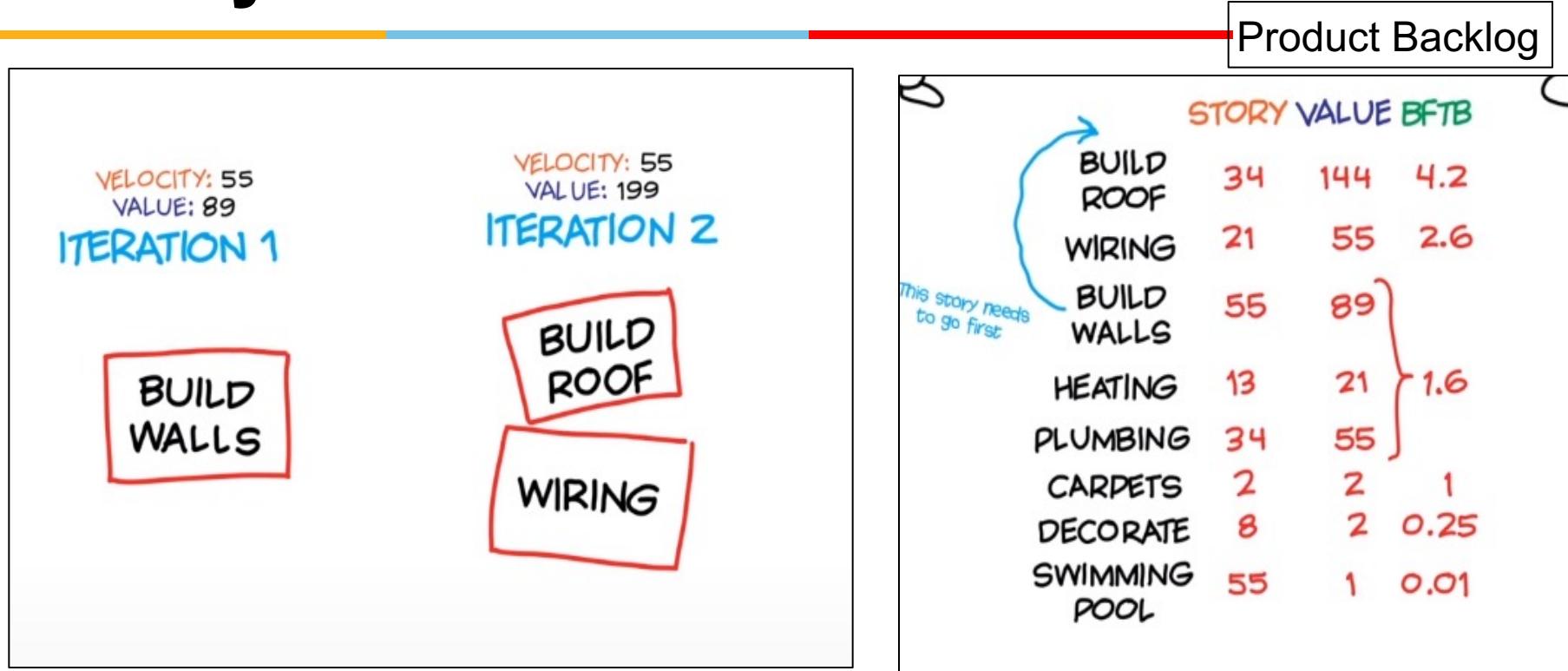
How stories are prioritized for each Iteration – by BFTB

Product Backlog



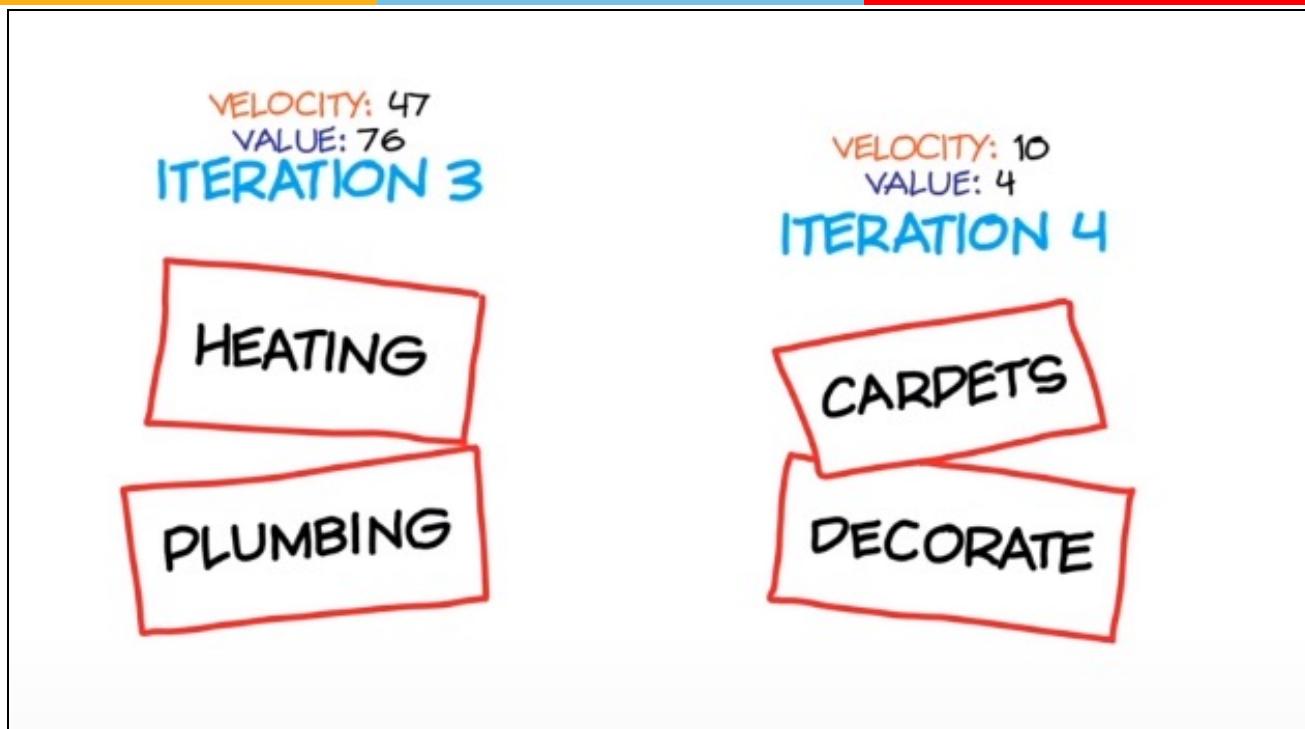
STORY VALUE BFTB			
BUILD ROOF	34	144	4.2
WIRING	21	55	2.6
BUILD WALLS	55	89	1.6
HEATING	13	21	
PLUMBING	34	55	0.25
CARPETS	2	2	
DECORATE	8	2	
SWIMMING POOL	55	1	0.01

Highest value delivered in early Iterations



Suppose, Cost of this iteration-1 is 20000\$;
= $20000/89 \sim 225$ Per Value Point.
For Iteration-2 = $225*199 \sim \$44,000$ (Highest value delivered)

Value delivered decreases as iteration progress



Iteration-3 value = $76 * \$225 = \$17,100$; Iteration-3 = $4 * 225 = 900$

- **This is an example**, but In reality, the value will decrease after many iterations, then customer can take a call to continue the project or not.
- Over the period of time, after some iterations the velocity will become stable and value delivered will decrease.

Estimation Exercise (Assume, 2 week Iteration)

Iteration 7 (complete)

As a technical specialist
I want to adjust the
turboencapsulator
So that signals will be in
phase

Story points: 5

Value points: 13

As a customer
I want my hydrooptic
vanes serviced
So that they will last
longer

Story points: 8

Value points: 8

1. Iteration 7 velocity?

$8 + 5 = 13$ story points

2. How long is a story
point?

$80 \text{ hrs} = 13 \text{ story pts}$

$1 \text{ story pt} = (80/13) \text{ hrs}$
 $= 6.15 \text{ hrs}$

3. Which stories in the
next iteration?

4. How long is the rest
of the backlog?
Total sp = $5+3+5+8+8+21$
 $= 50$

Time = $50 \times 6.15 \text{ hrs}$
 $= 307.5 \text{ hrs}$

Iteration 8 (new)

As a pilfrometer engineer
I want to order grommets
online
So I can stay mobile

Story points: 5

Value points: 5

BFTB: $5/5 = 1$

BFTB: $2/3 = 0.67$

Story points: 3

Value points: 2

BFTB: $2/5 = 0.4$

Story points: 5

Value points: 2

BFTB: $3/8 = 0.38$

Story points: 8

Value points: 3

I want
fish
So
BFTB: $3/8 = 0.38$
Story points: 8
Value points: 3

BFTB: $3/21 = 0.14$
Story points: 21
Value points: 3

Story Points – Real Examples



Pointing User Stories

Pointing Rubric at iHeartMedia
(two week development sprints)

- 1: Text Change
- 2: Text Change + Small Functionality Change
- 3: One Day of Work for One Developer
- 5: One Week of Work for One Developer
- 8: Two Weeks of Work for One Developer
- 13: Two Weeks of Work for Two Developers

Pointing Rubric at Condé Nast Entertainment
(one week development sprints)

- 1: Text Change
- 2: Text Change + Small Functionality Change
- 3: One Day of Work for One Developer
- 5: One Week of Work for One Developer
- 8: One Week of Work for Two Developers
- 13: Must Be Broken Down Into Smaller Stories



Other Estimation Techniques

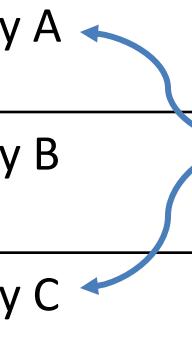
Estimate by Analogy

- Comparing a user story to others
 - “This story is like that story, so its estimate is what that story’s estimate was.”
- Don’t use a single gold standard
 - Triangulate instead
 - Compare the story being estimated to multiple other stories

Triangulation

- Confirm estimates by comparing the story to multiple other stories.
- Group like-sized stories on table or whiteboard

3 points	Story A		
2 points	Story B	Story E	Story F
1 point	Story C	Story D	



Ideal Time

- How long something would take:
 - If it's all one person worked on
 - Had no interruptions
 - And everything you need is available.
- The ideal time of a football game is 90 minutes
 - Four 15-minute quarters
 - The elapsed time is much longer (3+ hours)
- It's easier to estimate in ideal time.
- It's too hard to estimate directly in elapsed time.
 - Need to consider all the factors that affect elapsed time at the same time you're estimating

Story Points Vs Ideal Time

- Story points help drive cross-functional behavior
 - Story point estimates do not decay
 - Story points are a pure measure of size
 - Estimating in story points is typically faster
-
- My ideal days cannot be added to your ideal days
 - Ideal days are easier to explain outside the team
 - Ideal days are easier to estimate at first

T –Shirt Sizing, Disaggregation

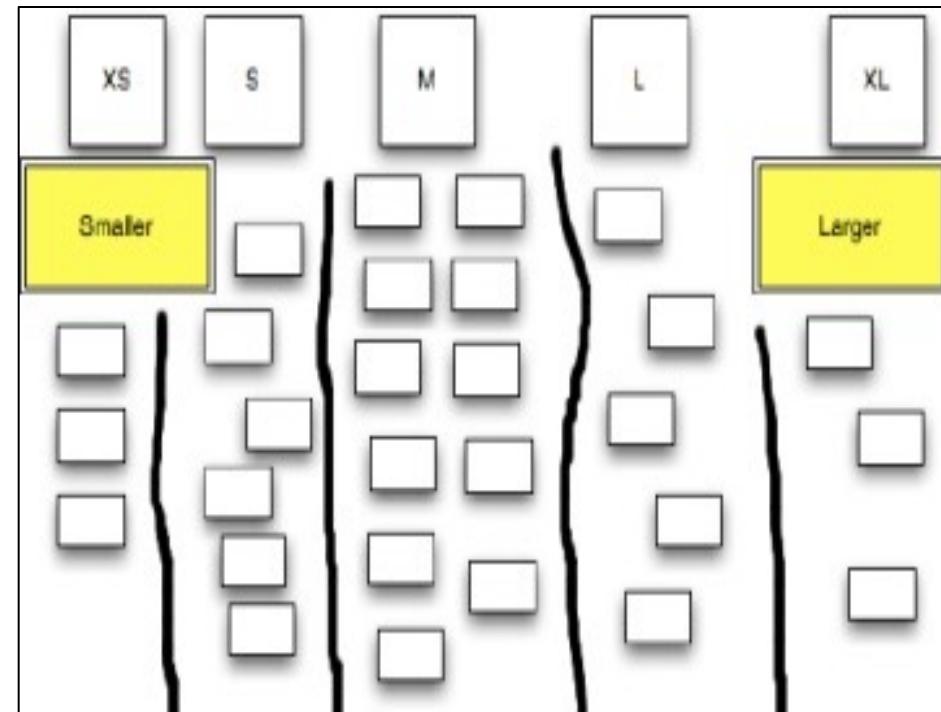
- **Level of Effort (LOE) or T-Shirt Sizing**
 - T-shirt size,” “level of effort” (LOE), or “small, medium, large.” (Easy, but lack precision, inability to add up several stories into a meaningful measure.)

Extra small 1 point	Small 2 points	Medium 3 points	Large 5 points	Extra Large 8 points	Extra Extra Large 13 points
-------------------------------	--------------------------	---------------------------	--------------------------	--------------------------------	---------------------------------------

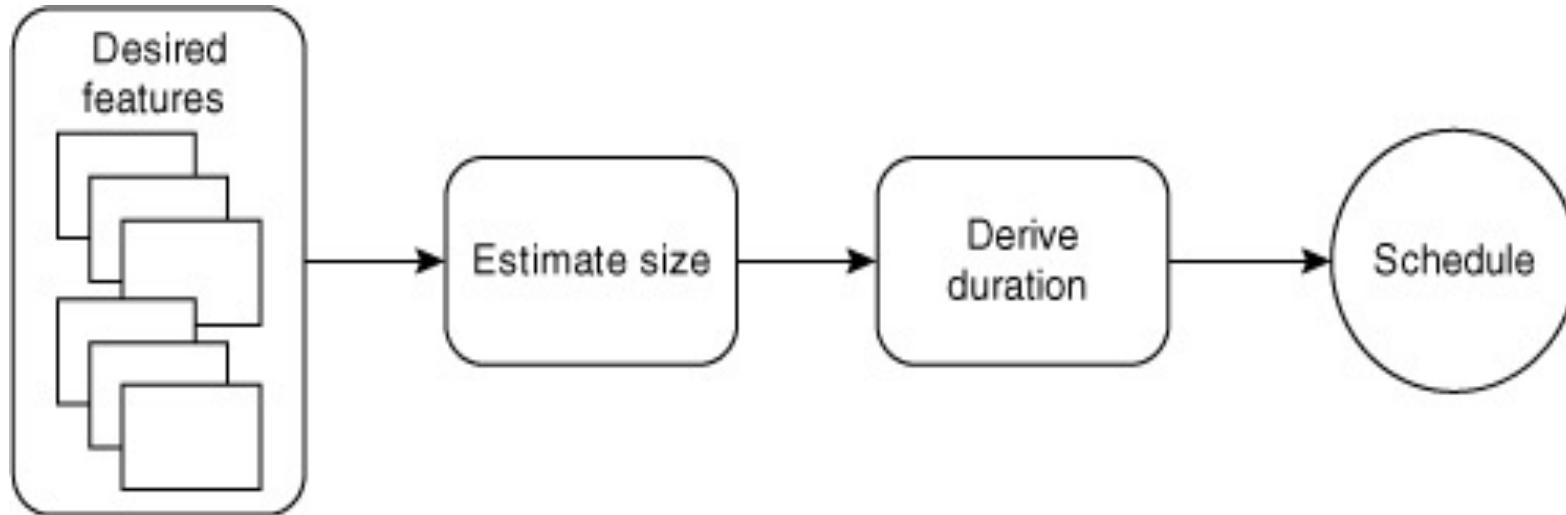
- **Disaggregation**
 - Breaking a big story into smaller stories ,we know how long the smaller stories take, So, disaggregating to something we know lets us estimate something bigger we don't know

Affinity Grouping

- Team members simply group items together that are like-sized, resulting in configuration similar to the one in figure.



Estimating the duration of a project begins with estimating its size.



- Sum the story-point estimates for all desired features we come up with a total size estimate for the project.
- If we know the team's velocity we can divide size by velocity to arrive at an estimated number of iterations.
- We can turn this duration into a schedule by mapping it onto a calendar.

Source: Agile Estimating and Planning by Mike Cohn
Published by Addison-Wesley Professional, 2005

Re-estimating

- Remembering that story points and ideal days are estimates of the size of a feature helps you know when to re-estimate.
- You should re-estimate only when your opinion of the relative size of one or more stories has changed.
- Do not re-estimate solely because progress is not coming as rapidly as you'd expected.
- Let velocity, the great equalizer, take care of most estimation inaccuracies.

Source: Agile Estimating and Planning by Mike Cohn
Published by Addison-Wesley Professional, 2005

Thank you



BITS Pilani
Pilani Campus

BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



Module-6 Agile Planning & Release Planning

Agile Planning



Diversity Latest Magazine Popular Topics Podcasts Video Store The Big Ic

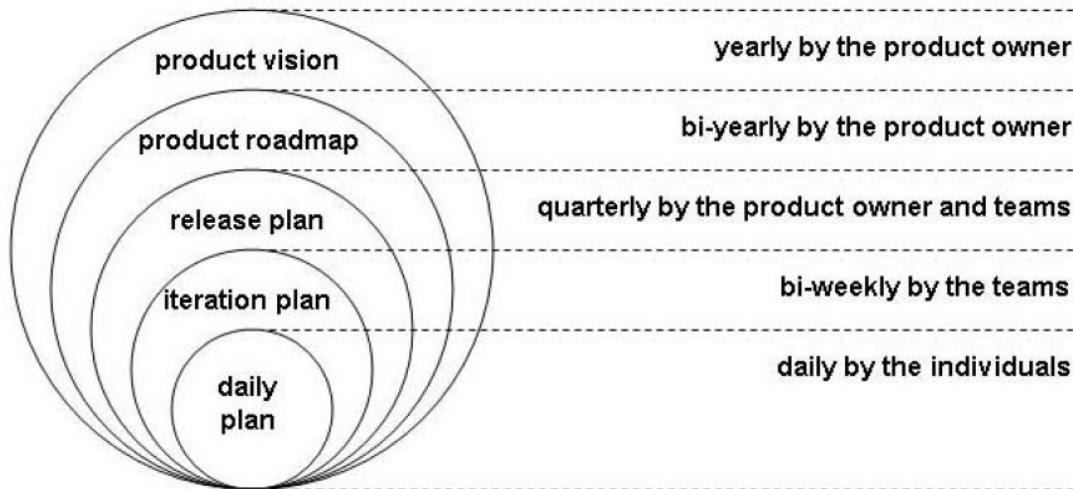
MANAGING ORGANIZATIONS

Bring Agile Planning to the Whole Organization

by Jeff Gothelf

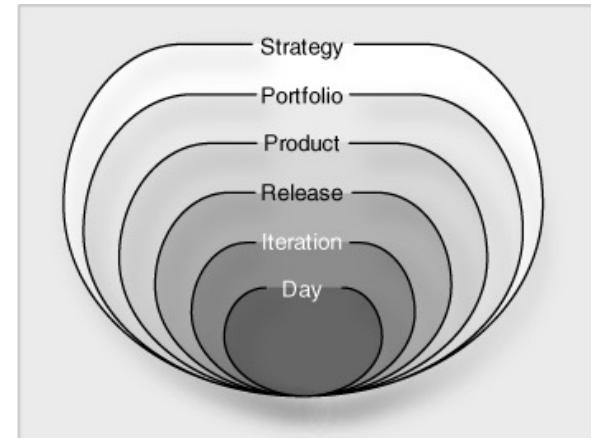
<https://hbr.org/webinar/2015/05/bring-agile-planning-to-the-whole-organization>

Agile Planning



Many Products or Services Organization

Single Product Organization



An Enterprise Agile Framework

Ref: 5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up by Hubert Smits

Release Planning

Inputs:

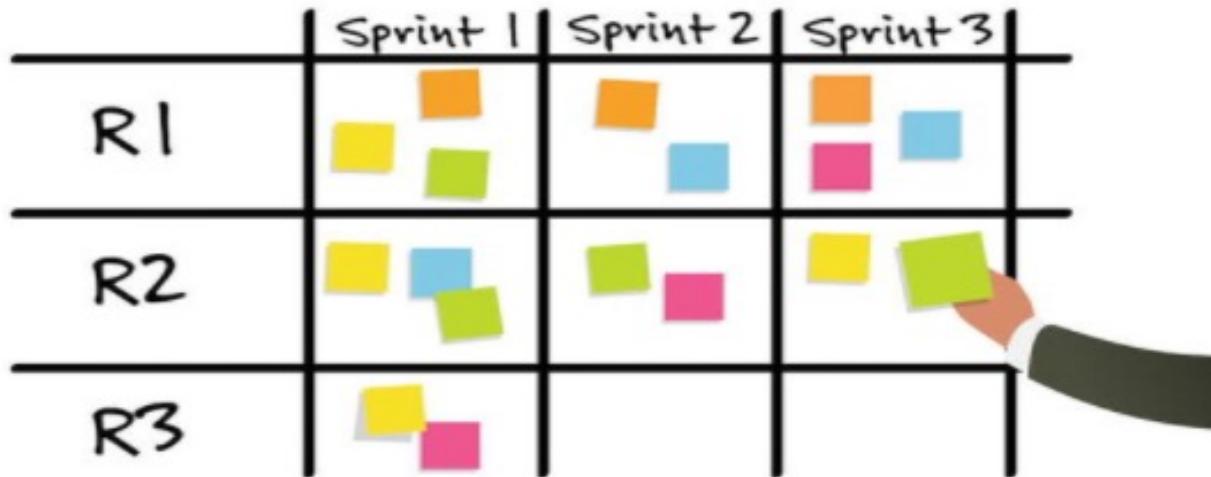
Product Vision

Product Road Map

Product Backlog

Release Backlog, Velocity, Iteration length, Trade off-matrix (Scope, Cost, Schedule)

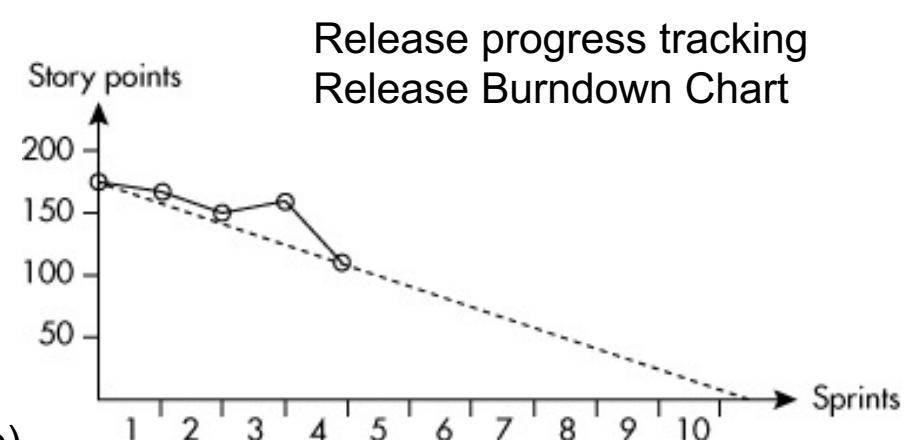
Release Planning



Example: Release planning

Inputs:

- Release backlog - 50 User stories
 - (200 Story points)
- Velocity = 20 Story points
- Iteration Length - 2 weeks
- Budget = \$200,000
- Cost of each Iteration = \$20000
- Trade of Matrix :
- Schedule (Fixed), Cost(Fixed), Scope(Flexible)



Outputs:

- Total number of Iterations required = 10 Iterations ($200/20$)
- If you are planning for 2 releases
- Number of iterations per release = 5 Iterations
- Duration of each release = $5*2 = 10$ weeks
- Suppose, Iteration cost = \$25000; only 8 iterations is possible.
- 160 points can be delivered (Scope may have to be reduced)

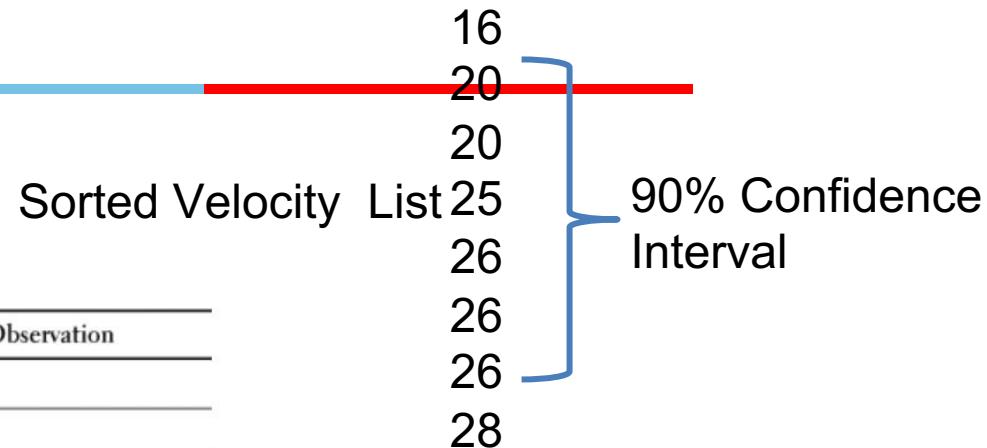
Estimating Velocity

- Use historical values.
- Run an iteration
- Make a forecast.
- You should consider expressing the estimate as a range.
 - Example: If your team velocity is 20 story points - You have a very limited chance of being correct in future. Instead give a range 15-24 story points

Iterations Completed	Low Multiplier	High Multiplier
1	0.6	1.60
2	0.8	1.25
3	0.85	1.15
4 or more	0.90	1.10

High-confidence forecast- Example

- Velocity of completed **8 sprints**
- : 20, 25, 28, 26, 16, 20, 26, 26



Number of Velocity Observations	<i>n</i> th Velocity Observation
5	1
8	2
11	3
13	4
16	5
18	6
21	7
23	8
26	9

- 20 → Lower confidence, certainly we will do
23 → Mean Velocity – We will get here
26 → Upper confidence, Most we could expect

Use the *n*th Lowest and the *n*th Highest Observation of a Sorted List of Velocities to Find a 90% Confidence Interval

Source: Succeeding with Agile SW development by Mike Cohen

Creating a Release Plan Exercise



- The backlog for this release has 140 story points, with a start date of D0 and a sprint length of 2 weeks. Range of estimated velocities: Low = 18; High = 20
 - The average velocity of the first two sprints was measured to be 15 Story Points.
1. Calculate the maximum and minimum schedules, as well as the points that can be completed per sprint, by maintaining the same velocity range.
 2. What is the maximum and minimum timeline and number of points that can be completed if the budget is \$140000 and the cost of a sprint is \$20000?

Creating a Release Plan Exercise



1. Calculate the maximum and minimum schedules, as well as the points that can be completed per sprint, by maintaining the same velocity range.

- Velocity High =20; Number of Iteration Required = $140/20 = 7$
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number of iteration required to complete the 110 points = $110/20 = 5.5 \sim 6$ Sprints
- Sprint 1-2 = 15 points; Sprint 3-7 = 20; Total number of points that can be delivered = 130

- Velocity low = 18; Number of Iteration Required = $140/18 \sim 8$ Sprints
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number of iteration required to complete the 110 points = $110/18 = 6.1 \sim 7$ Sprints
- Sprint 1-2 = 15 points; Sprint 3-7 = 18; Total number of points that can be delivered = 120

Creating a Release Plan Exercise



2. What is the maximum and minimum timeline and number of points that can be completed if the budget is \$140000 and the cost of a sprint is \$20000?

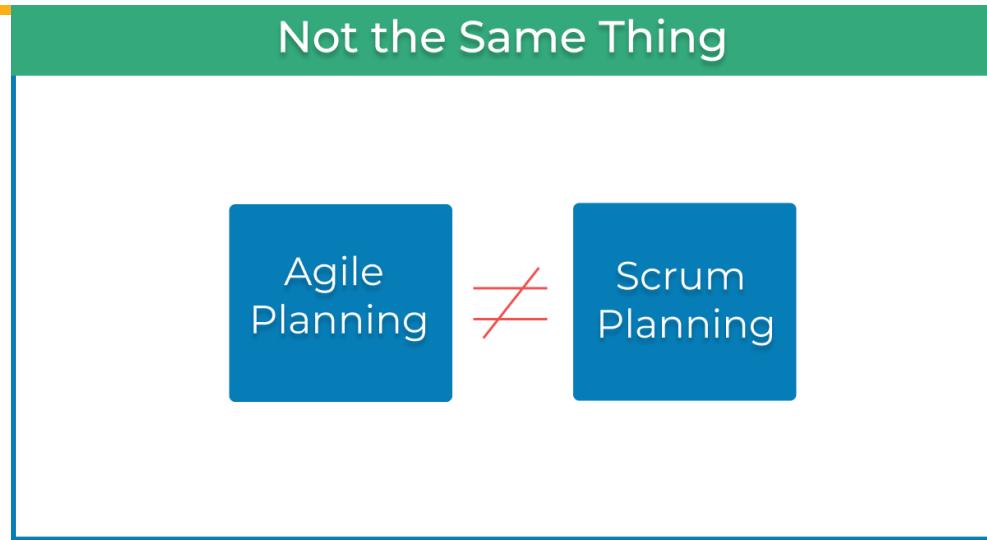
- Available budget is \$140000, Each Iteration cost = \$20000; Only 7 Iterations is possible.
- Max and Min Schedule is same = D0 +14 Weeks
- Velocity High =20; Number of Iteration Required = $140/20 = 7$
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number of iteration required to complete the 110 points = $110/20 \sim 6$
- Sprint 1-2 = 15 points; Sprint 3-7 = 20 points ;
- Total number points that can be delivered = 130 points

- Velocity low = 18; Number of Iteration Required = $140/18 \sim 8$ Sprints
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number of iteration required to complete the 110 points = $110/18 \sim 7$
- Max Schedule = D0+ 14 weeks



Module-6 Agile Planning & Release Planning – Additional Notes

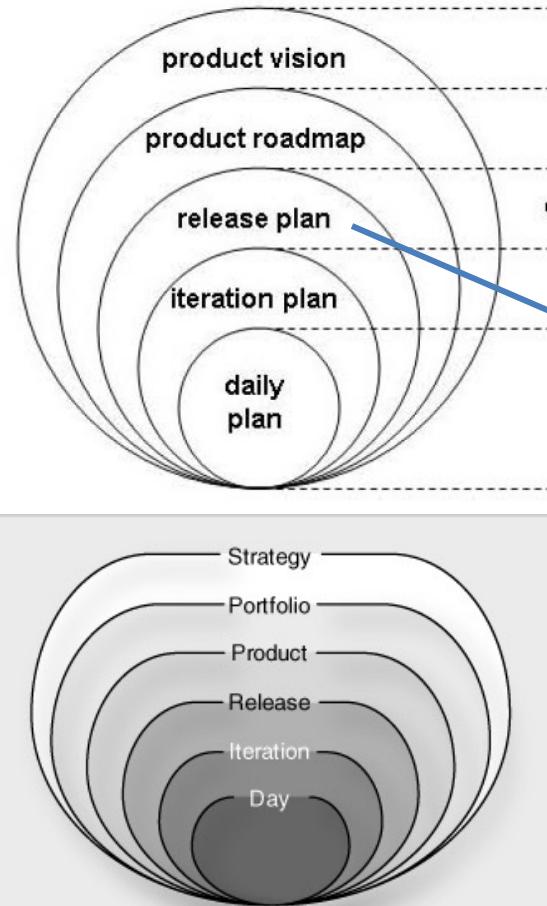
Agile Planning



- Agile thinking applied across various industries and not just software.
- It is more important to know how to apply generic techniques and practices on the global company level, irrespective of the type of business.

Source : <https://kanbanize.com/agile/project-management/planning>

5-Levels of Agile Planning (Product Planning)



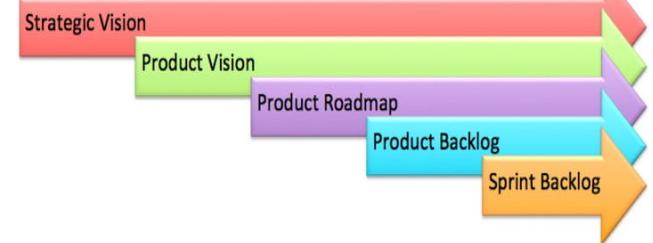
Ref: 5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up by Hubert Smits

10/9/22

SE ZG544 S1-22 Agile SW Process

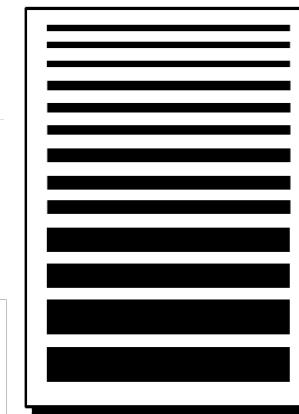
14

Flexibility to accommodate change
Decreases



DEEP:
Detailed,
Emergent,
Estimable
Prioritized

Low Priority



Fine-grained, detailed items ready
to be worked on in the next sprint

Large, coarse-grained items

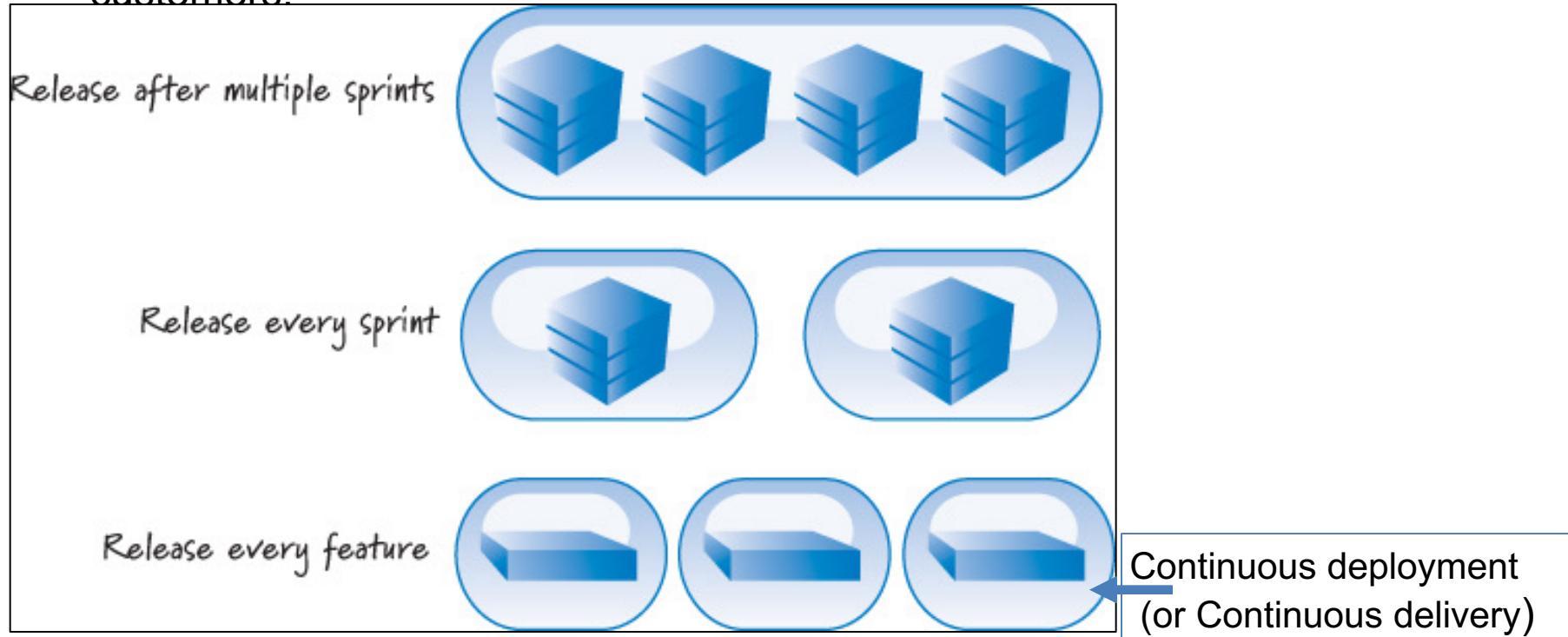
5-Levels of Agile Planning (Product Planning)



- Each of the five levels of planning addresses the fundamental planning principles: priorities, estimates and commitments.
- 5 Levels of Agile Planning is aimed to avoid big upfront design
- Most agile teams are concerned only with the three innermost levels (Day, Iteration, Release) of the **planning onion**.
- Involve stakeholders in planning, Review the plans frequently

Patterns of Release Planning/Different release cadences

- Many organizations have its own cadence regarding release of products to its customers.



Whichever release cadence being followed, Most organizations find some amount of longer-term, higher-level planning to be useful. We refer to this type of planning as release planning

Agile Release Planning

- Release planning is an important task for product people working with agile teams:
 - It ensures that the product is moving in the right direction and it connects Product strategy and tactics.
- Release as a version of a product:
 - For example, Mac OS X Catalina and Windows 10.
 - Releases come in two flavors: major releases, like iOS 13, and minor releases, such as iOS 13.3.
- Release planning is the process of determining the desired outcome of one or more major releases and maximizing the chances of achieving it.

Source: <https://www.romanpichler.com/blog/release-planning-advice/>

Agile Release Planning ...

- Agile release planning provides a high-level summary timeline of the release schedule (typically 3 to 6 months).
- Agile release planning also determines the number of iterations or sprints in the release.
- Allows the product owner and team to decide how much needs to be developed and how long it will take to have a releasable product based on business goals, dependencies, and impediments.

Source: <https://www.romanpichler.com/blog/release-planning-advice/>

Make Release Planning Collaborative

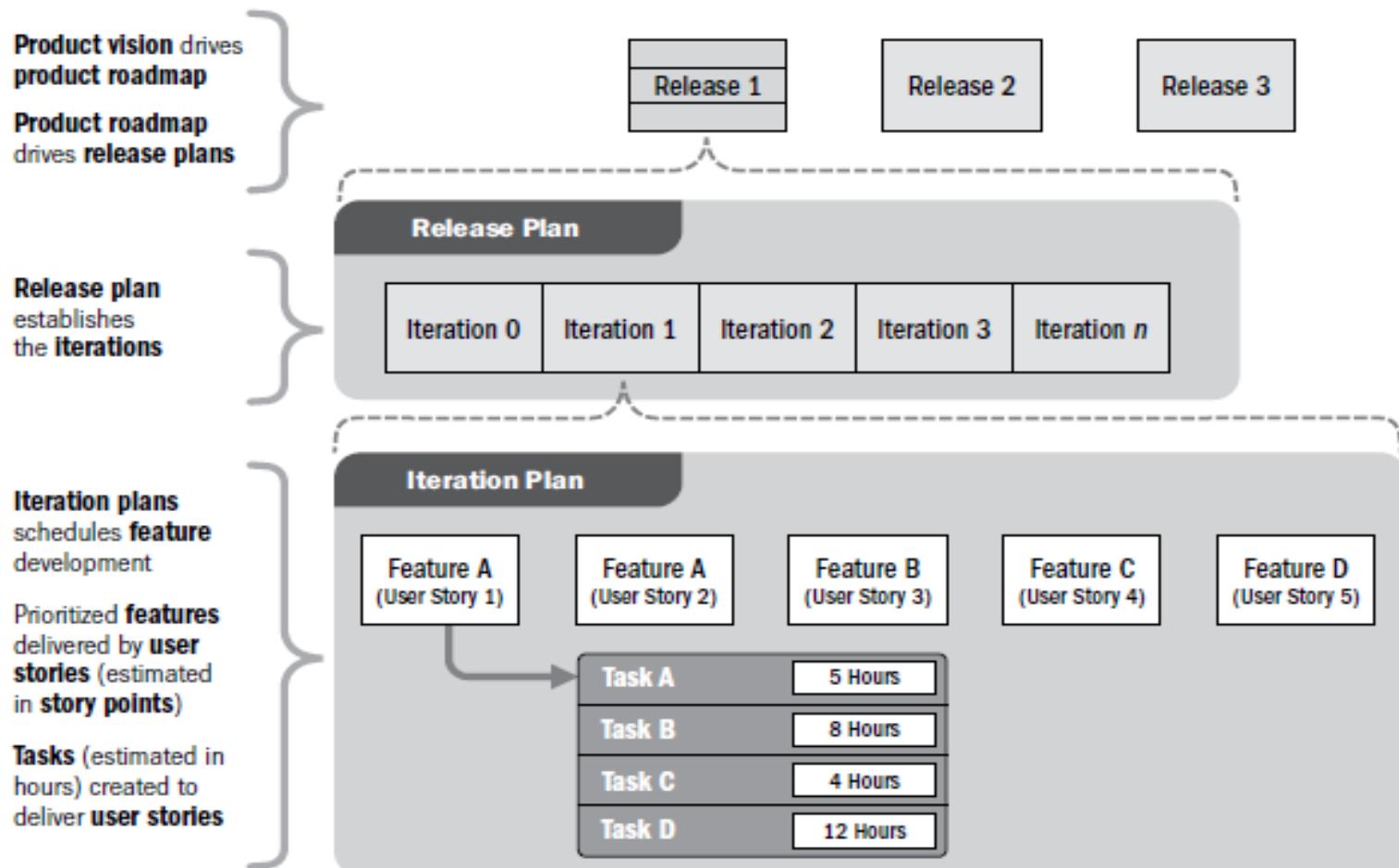
- Release planning is best done as a collaborative effort by involving the stakeholders and the development team



- Schedule regular roadmapping sessions.
- Possibly as part of your strategy review process and invite key stakeholders and development team members.
- Discuss Release Progress
- Invite Stakeholders to Sprint Review meetings.

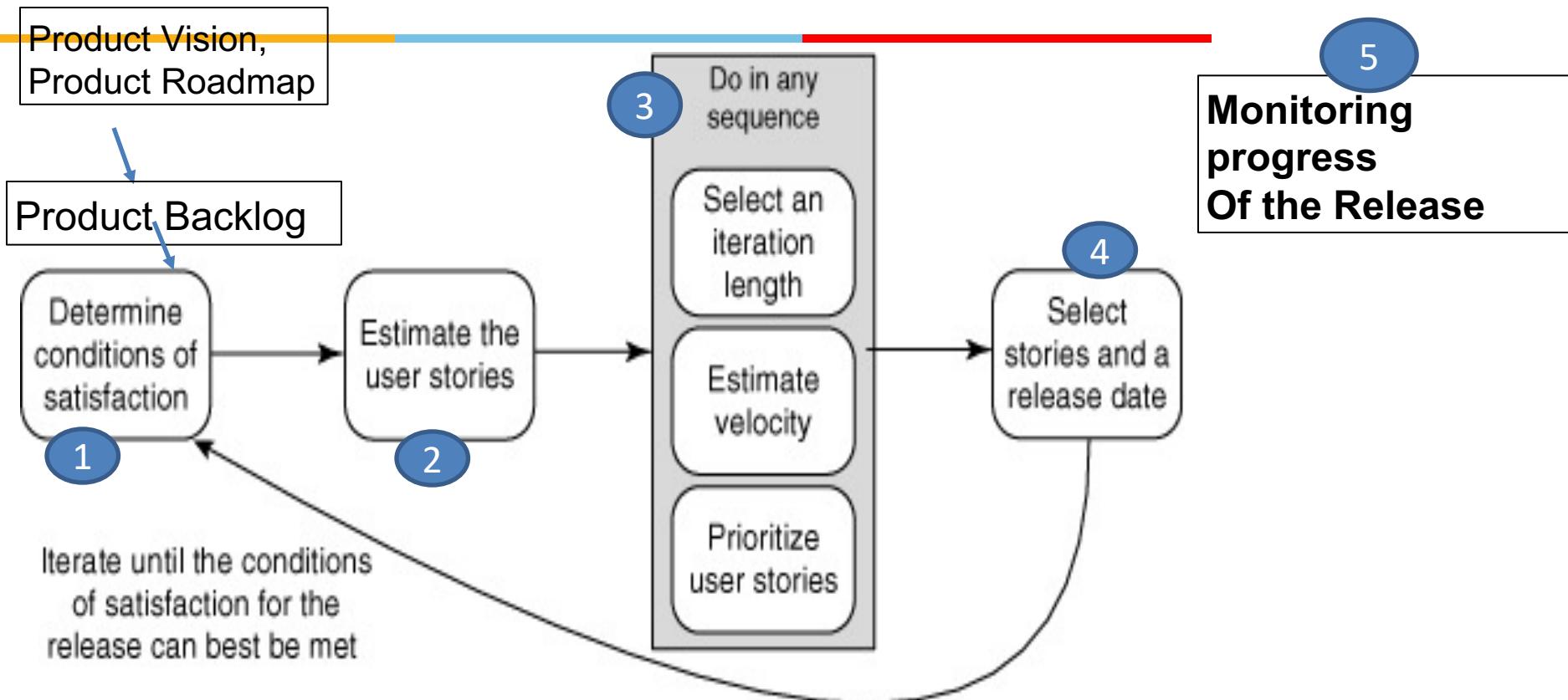
Source: <https://www.romanpichler.com/blog/release-planning-advice/>

Relationship between product vision, product roadmap, release planning, and iteration planning.



Source: PMI.ORG

The steps in planning a release.



- 1 Use Trade-off Matrix (Fixed, Flexible, Accept), Date, Scope, Cost - Fix important factor

Given a fixed schedule, we will choose a level of resources and adjust the features set as necessary.

Development Constraint Combinations



Project Type	Scope	Date	Cost
Fixed Everything (Not Recommended)	Fixed	Fixed	Fixed
Fixed Scope and Date (Not Recommended)	Fixed	Fixed	Flexible/Accept
Fixed Scope	Fixed	Flexible	Fixed/Flexible
Fixed Date	Flexible	Fixed	Fixed

Condition of satisfaction

- **Establishing clear, specific, and measurable goals.** Call these goals product or release goals - captured in product roadmap.
- **Prioritize the Success Factors for Releases:**
 - But in reality, unforeseen things do happen. The development progress may not be as fast as anticipated, for instance, or one of the technologies may not work as expected.
 - Use Trade-off Matrix (Fixed, Flexible, Accept)
 - Date, Scope, Cost - Fix important factor
- **Quality:** Quality should be fixed and not be compromised. Otherwise, responding to user feedback and changing market conditions and quickly adapting your product will be hard, if not impossible.

Estimate User Stories

- It is not necessary to estimate everything that a product owner may ever want.
- It is necessary only to have an estimate for each new feature that has some reasonable possibility of being selected for inclusion in the upcoming release.
- Often, a product owner will have a wish list that extends two, three, or more releases into the future. It is not necessary to have estimates on the more distant work.

Factors in Select an Iteration Length

- The length of the release being worked on
- The amount of uncertainty
- The ease of getting feedback
- How long priorities can remain unchanged
- Willingness to go without outside feedback
- The overhead of iterating
- How soon a feeling of urgency is established
- Make a Decision and stick to the Rhythm
- 2 weeks sprint is ideal.

The Overall Length of the Release

- Short projects benefit from short iterations.

The length of a project's iterations determines:

1. How often the software can be shown and progress measured?
2. How often the product owner and team can refine their course, because priorities and plans are adjusted between iterations.
 - Opportunities to gather end-of-iteration feedback
 - General rule of thumb: Aim for five to six feedback opportunities per release.
 - Example: 3 months release
 - Iteration length : 2 weeks – 5 times feedback for course corrections-ok
 - 4 weeks iteration provides only two times feedback- not ok

The Amount of Uncertainty

Uncertainty comes in multiple forms.

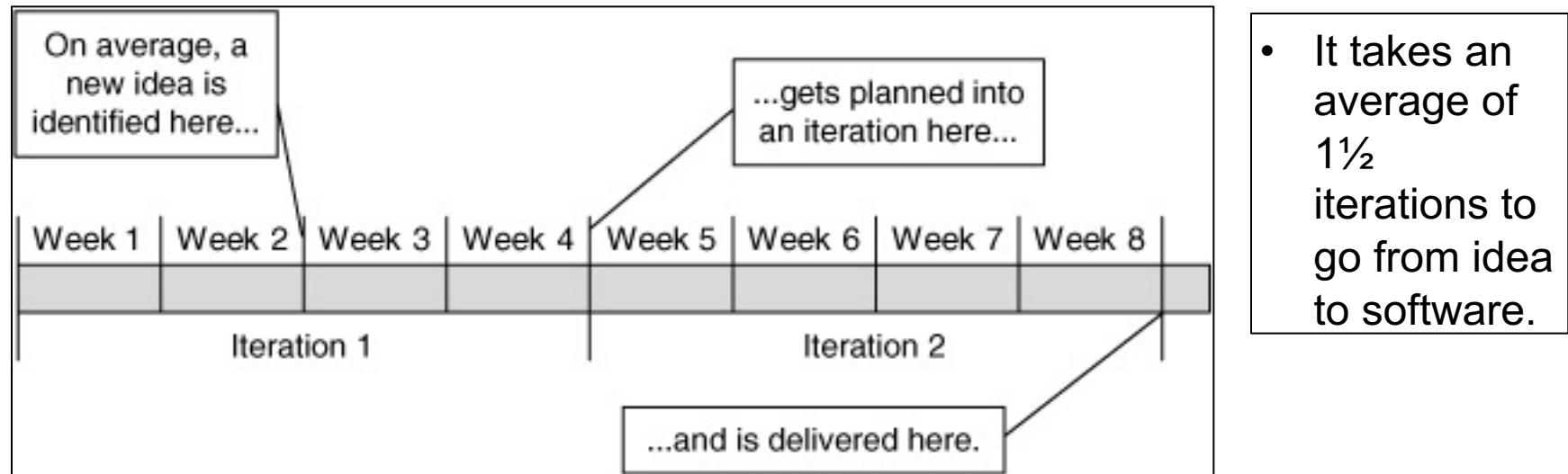
- User need
- Technical aspects
- Team Velocity
- The more uncertainty of any type there is, the shorter the iterations should be.
 - Shorter iterations allow more frequent opportunities for the team to measure its progress through its velocity and more opportunities to get feedback from stakeholders, customers, and users.

The ease of getting feedback

- Choose your iteration length to maximize the value of the feedback that can be received from those inside and outside the organization.

How Long Priorities Can Remain Unchanged

- Once a development team commits to completing a specific set of features in an iteration, it is important that that the **product owner not change priorities** during the iteration, also protect the team from others to change the priorities.



Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 20

Willingness to Go without Outside Feedback

- Even with a well-intentioned and highly communicative team, it is possible that the results of an iteration could be found worthless when shown to the broader organization or external users at the conclusion of the iteration.
 - This may happen if the developers misunderstand the product owner (and don't communicate often enough during the iteration).
 - It could also happen if the product owner misunderstands the needs of the market or users.
- Less often a team receives outside feedback, the more likely we are to go astray and the greater the loss will be when that happens.

The Overhead of Iterating

- There are costs associated with each iteration.
- For example, each iteration must be fully regression tested:
 - If this is costly (usually in terms of time), the team may prefer longer, four-week iterations.
 - Naturally, one of the goals of a successful agile team is to reduce (or nearly eliminate) the overhead associated with each iteration.
 - But especially during a team's early iterations, this cost can be significant and will influence the decision about the best iteration length.

How Soon a Feeling of Urgency Is Established

- “As long as the end date of a project is far in the future, we don’t feel any pressure and work leisurely. When the pressure of the finish date becomes tangible, we start working harder.” - Niels Malotaux (2004).
- Even with four-week iterations , it is sufficiently far away that many teams will feel tangibly less stress during their first week than during the fourth and final week of an iteration.
- The point is not to put the team under more pressure but distribute it more evenly across a suitably long iteration.

Stick with It to Achieve a Steady Rhythm



- Whatever duration you choose, you are better off choosing a duration and sticking with it rather than changing it frequently.
- Teams fall into a natural rhythm when using an unchanging iteration duration.
- A regular iteration rhythm acts like a heartbeat for the project.
- “Rhythm is a significant factor that helps achieve a sustained pace”

Making a Decision

- One of the main goals in selecting an iteration length is finding one that encourages everyone to work at a consistent pace throughout the iteration.
- If the duration is too long, there is a natural tendency to relax a bit at the start of the iteration, which leads to panic and longer hours at the end of the iteration. Strive to find an iteration duration that smooths out these variations.
- Two-week iterations to be ideal.
- **Mike Cohen** suggests:
 - To follow a macro-cycle of six two-week iterations followed by a one-week iteration. “ $6 \times 2 + 1$.”
 - During the one-week iteration, however, the team chooses its own work.

Estimating Velocity

- **It is better to be roughly right than precisely wrong.”—John Maynard Keynes.**
- One of the challenges of planning a release is estimating the velocity of the team. You have the following three options:
 - Use historical values.
 - Run an iteration
 - Make a forecast.
- You should consider expressing the estimate as a range.
 - You could create a range by simply adding and subtracting a few points to the average or by looking at the team's best and worst iterations over the past two or three months.
 - Example: If your team velocity is 20 story points - You have a very limited chance of being correct in future. Instead give a range 15-24 story points.

Using Historical values

- Use historical values only when very little has changed between the old project and team and the new project and team.
- Before using them, ask yourself questions like these:
 - Is the technology the same?
 - Is the domain the same?
 - Is the team the same?
 - Is the product owner the same?
 - Are the tools the same?
 - Is the working environment the same?
 - Were the estimates made by the same people?
 - The answer to each question is often yes when the team is moving onto a new release of a product they just worked on. In that case, using the team's historical values is entirely appropriate. Even though velocity in a situation like this is relatively stable, you should still consider expressing it as a range.

Run an Iteration

- An ideal way to forecast velocity is to run an iteration (or two or three) and then estimate velocity from the observed velocity during the one to three iterations.
- Because the best way to predict velocity is to observe velocity, this should always be your default approach.
- Multipliers for Estimating Velocity Based on Number of Iterations Completed

Iterations Completed	Low Multiplier	High Multiplier
1	0.6	1.60
2	0.8	1.25
3	0.85	1.15
4 or more	0.90	1.10

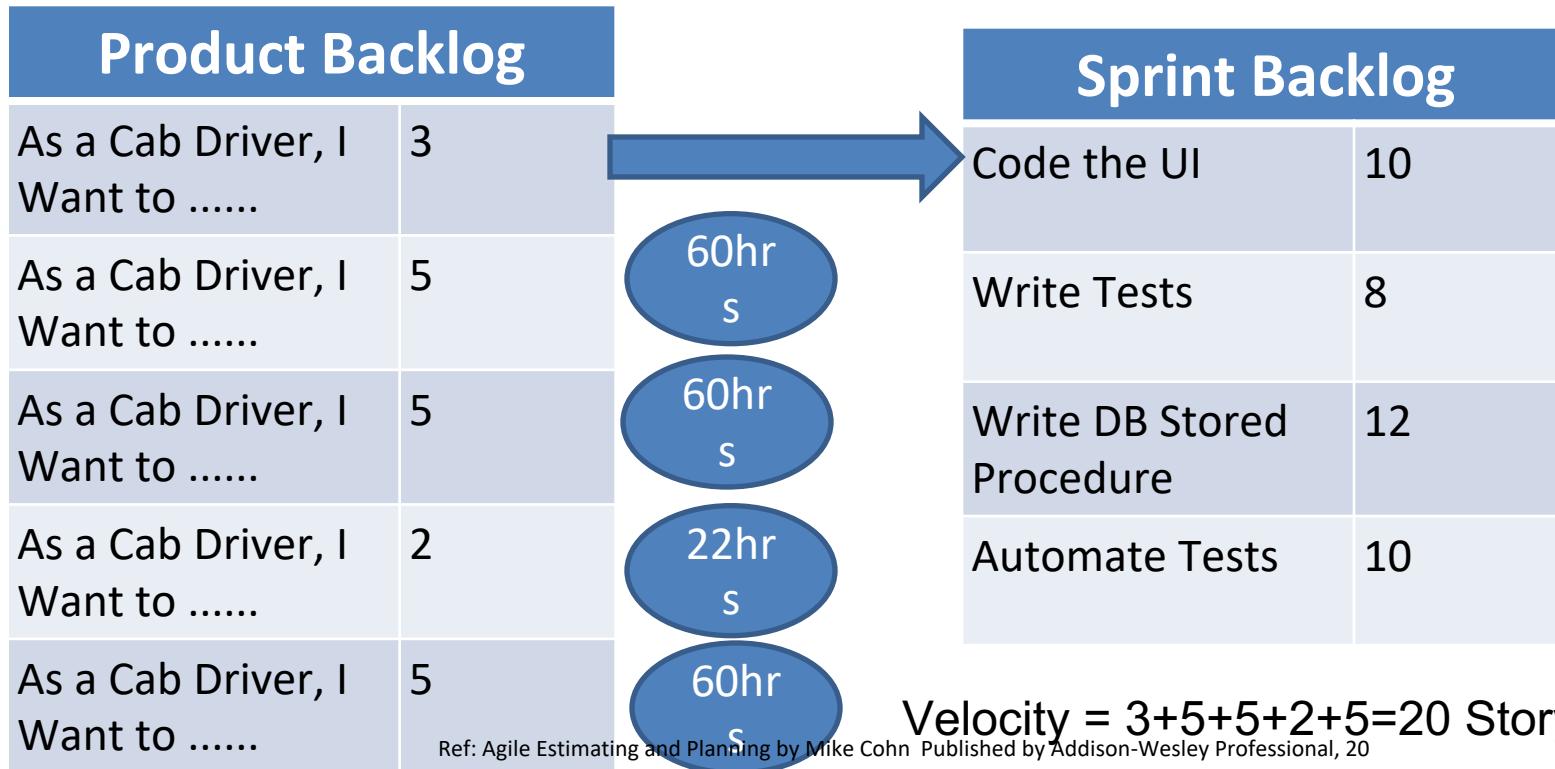
Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 20

Forecasting Velocity

1. Estimate the number of hours that each person will be available to work on the project each day.
2. Determine the total number of hours that will be spent on the project during the iteration.
3. Arbitrarily and somewhat randomly select stories, and expand them into their constituent tasks.
 - Repeat until you have identified enough tasks to fill the number of hours in the iteration.
 - 4. Convert the velocity determined in the preceding step into a range.

Example

Number of Team members	Available hours per day/team member	Total Available hrs	Team Capacity for 10 days iteration
4	6 hrs.	$4*6= 24$	$10*24=240$



High-confidence forecast- Example



- Suppose we want to create high confidence forecast(90%) for the next release.
- As soon as the team has run five or more sprints, we can create a high-confidence forecast
- Suppose, Velocity of completed 8 sprints:20, 25, 28, 26, 16, 20, 26, 26.
- Sorted list:16, 20, 20, 25, 26, 26, 26, 28

Use the nth Lowest and the nth Highest Observation of a Sorted

List of Velocities to Find a 90% Confidence Interval

Number of Velocity Observations	<i>n</i> th Velocity Observation
5	1
8	2
11	3
13	4
16	5
18	6
21	7
23	8
26	9

- Velocity of completed **8 sprints**
- :20, 25, 28, 26, 16, 20, 26, 26

Sorted Velocity List

16

20

20

25

26

26

26

26

28

90% Confidence Interval

- 20 → Lower confidence, certainly we will do
- 23 → Mean Velocity – We will get here
- 26 → Upper confidence, Most we could expect

Creation a Release Plan

- Total story points of Release backlog **divided by** Mean velocity or Velocity range.
- This will give us a provisional number of sprints required for the release
- **Example:**
 - Total Story points = 200; Mean velocity = 20; Number Sprints required = 10
 - We then map the identified number of sprints onto the calendar and consider the factors that are likely to influence the velocity and that are not accounted for in the velocity forecast.
 - These can include holidays, vacations, training and development, sickness statistics, and planned changes to the project organization, such as modifying the team composition. We adjust the forecasted velocity of each sprint accordingly.

Sample Release Plan

Sprint	1	2	3	4	5	6	7	8
Velocity forecast	N/A	12-32	18-28	21-28	11-18	16-23	21-28	21-28
Actual velocity	20	25	28					
Dependencies			Imaging library					
Releases				Alpha: Calls, basic text messages	Holidays	Beta: Conference calls, picture messages		V1.0
				Current sprint				

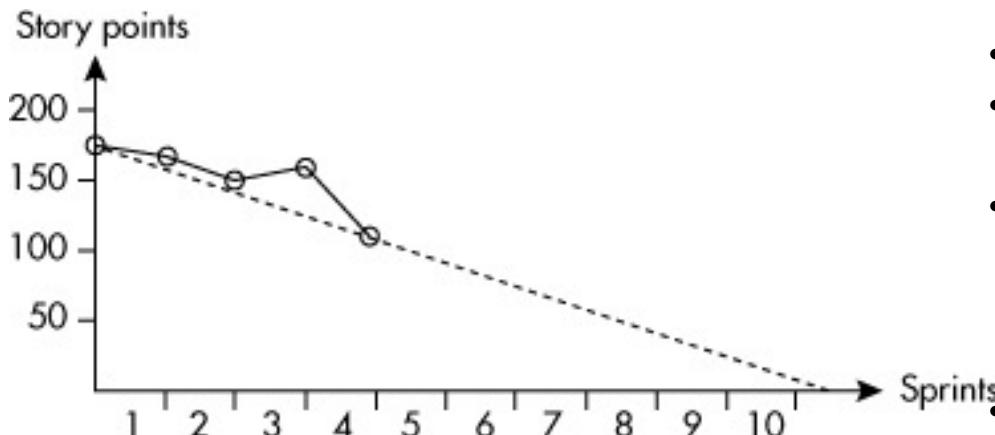
Iterations Completed	Low Multiplier	High Multiplier
1	0.6	1.60
2	0.8	1.25
3	0.85	1.15
4 or more	0.90	1.10

1. Actual velocity for the first three sprints of 20, 25, and 28.
2. The average (mean) velocity per sprint, then, is 24 points.
3. The Scrum team has forecasted a velocity of 21 to 28 points for the fourth, seventh, and eighth sprints using the multipliers Table(below).
4. The release plan also anticipates a velocity drop in sprints five and six, when several team members will take time off and then return to work.

Source: Agile Estimation and Planning by Mike Cohen

Tracking the Progress of the Release

– Release Burndown chart



- The solid line is the actual burndown- Indicate the progress
- Slow start. Might be - impediments and risks materializing, team-building dynamics, or technology issues.
- Third sprint, the remaining effort even increased. - caused by the team reestimating backlog items or discovering new requirements
- The fourth sprint saw a steep burndown; the project progressed fast.

- X-Axis - Number of sprints as the unit
- Y- Axis - Number of story points estimated
- The first data point is the estimated effort of the entire product backlog before any development has taken place.
- To arrive at our next data point, we determine the remaining effort in the product backlog at the end of the first sprint.
- Then we draw a line through the two points. This line is called the burndown(.... Line)
- It shows the rate at which the effort in the product backlog is consumed.
- If we extend the burndown line to the x-axis, we can forecast when the project is likely to finish—assuming effort and velocity stay stable.

Product Visioning - Level 1

- A product vision describes the **future state of a product** (Big Picture) that a company or team desires to achieve. You can also define that future state as: **a goal**.
- **Aligns:**
 - Product strategy, product development roadmap, backlog & planning, execution & product launch
 - There is/can be a difference between a product and company vision.
- **What information does a product vision contain?**
 - Focused on Customers (B2C or B2B)- How it will benefit the company and the customer.(What?)
 - It's looking into the future and outlining a clear state of the product/goal that the company and team(s) want to achieve. This goal should be underlined with the motivation behind it (Why?, not How?)
 - The art of defining a great product vision that people want to follow is to make it catchy.

Source:<https://www.christianstrunk.com/blog/product-vision>

A. How to define a product vision?



1. Defining key product information.

- Have some valid data in the product discovery process to find answers to open questions.
- Gaining a clear picture of your customer, your market, the problems you want to solve, and your business goals
- According to Roman Pichler's product vision board, it's important to answer 4 key questions:
- What's the target group?, What are the customer needs?, What is and will be the product and its USP(s)?, What are the business goals?

2. Phrasing the product vision in one inspiring sentence.

- Examples: Google's company vision statement is: *“to provide access to the world's information in one click.”* because that's Google's core business.
- Card reader Makers: *“We believe in a world where small businesses can offer a super fast and safe payment experience to their customers, for minimal costs with no administrative efforts.”*

3. Why is having a product vision important? – Gives direction to Teams

- Who owns? What is the Process to create product vision?

B. How to define a product vision?- Classical format.

- **Create an elevator statement** or a Product vision box/Product Vision Board . (Non technical)
- A format popularized by Geoffrey Moore's classic *Crossing the Chasm*

For **(target customer)** who **(statement of need or opportunity)**, the **(product name)** is a **(product category)** that **(key benefit, reason to buy)**.

Unlike **(primary competitive alternative)**, our product **(statement of primary differentiation)**.

- Here's an example of a product vision statement for Microsoft Surface:
 - For the business user who needs to be productive in the office and on the go, the Surface is a convertible tablet that is easy to carry and gives you full computing productivity no matter where you are.
Unlike laptops, Surface serves your on-the-go needs without having to carry an extra device.
- Any further planning (Design) at this stage may divert our attention from future vision of the product.

Source:280 Group LLC

Product Roadmap – Level 2 Planning



- A product vision is a high-level aspirational projection of the future state of a product.
 - It must be impactful to generate sufficient interest among the innovators, early adopters, and early-stage investors.
- A product roadmap is essentially a timeline of feature rollout plans.- (Review the roadmap regularly)
 - It helps product managers prioritize R&D dollars to maximize chances of realizing the product's promised or anticipated ROI.
 - It allows the product team to focus on more value-creating features "here and now" versus hundreds of features that might have limited relative potential.
 - It helps customers know that their favorite features are planned somewhere down the road, and, if they so desire, the product team can expedite them.
 - It also allows customer feedback of what features are perceived as critical and what could be deferred to another time.
 - Helps the delivery team to see as whole, learn business priorities, Provide technical and estimates inputs to Product roadmap.

Type of Product Roadmaps

- Goal/Objectives driven roadmap
- Feature Driven
- Date/Time Driven

Goal Oriented Roadmap



THE GO PRODUCT ROADMAP



 DATE The release date or timeframe	Date or timeframe	Date or timeframe	Date or timeframe	Date or timeframe
 NAME The name of the new release	Name/version	Name/version	Name/version	Name/version
 GOAL The reason for creating the new release	Goal	Goal	Goal	Goal
 FEATURES The high-level features necessary to meet the goal	Features	Features	Features	Features
 METRICS The metrics to determine if the goal has been met	Metrics	Metrics	Metrics	Metrics

When will the release be available?

What is it called?

**Why is it developed?
Which benefit does it offer?**

What are the 3-5 key features?

How do we know that the goal is met?

A small cartoon character of a notepad with a face, arms, and legs, holding a pencil and looking towards the right.

Goal Oriented Roadmap – An Example



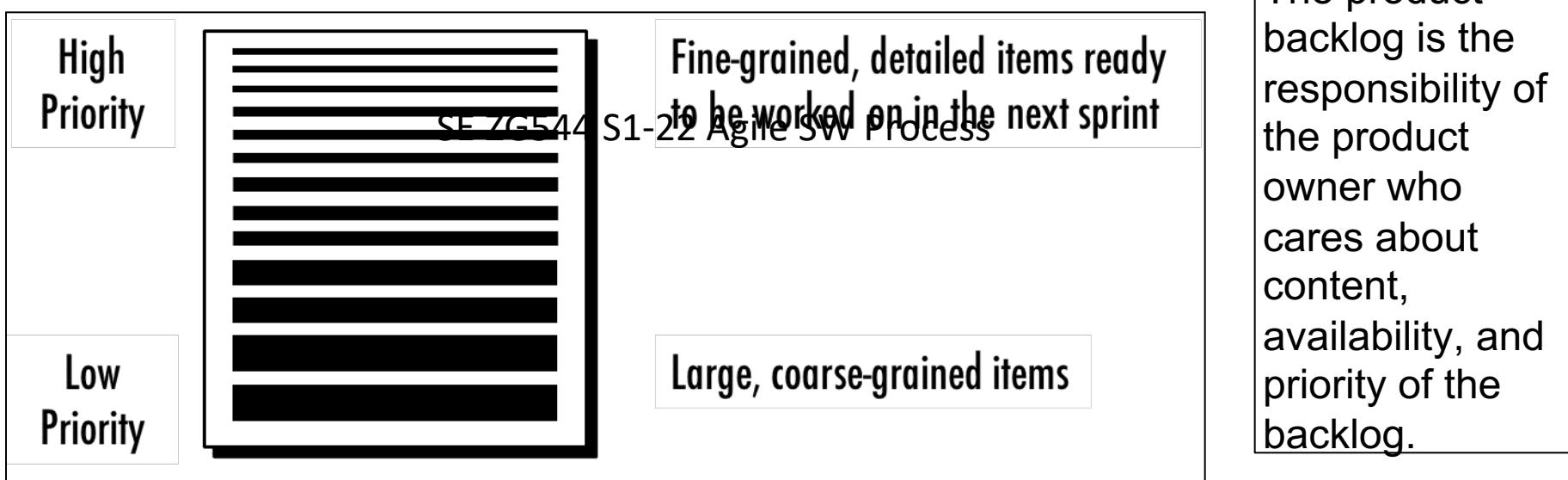
Develop a new dance game for girls aged eight to 12 years. The app should be fun and educational allowing the players to modify the characters, change the music, dance with remote players, and choreograph new dances.

	1 st quarter	2 nd quarter	3 rd quarter	4 th quarter
	Version 1	Version 2	Version 3	Version 4
	Acquisition: Free app, limited in-app purchases	Activation: Focus on in-app purchases	Retention	Acquisition: New segment
	<ul style="list-style-type: none">Basic game functionalityMultiplayerFB integration	<ul style="list-style-type: none">Purchase dance movesCreate new dances	<ul style="list-style-type: none">New characters and floorsEnhanced visual design	<ul style="list-style-type: none">Street dance elementsDance competition
	Downloads: top 10 dance app	Activations, downloads	Daily active players, session length	Downloads

Source: <https://www.romanpichler.com/blog/goal-oriented-agile-product-roadmap/>

Product Backlog - Level 3

- A product backlog is a prioritized list of work* for the development team that is derived from the roadmap and its requirements.
- The most important items are shown at the top of the product backlog so the team knows what to deliver first.



* List of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.

Source: <https://www.romanpichler.com/blog/goal-oriented-agile-product-roadmap/>

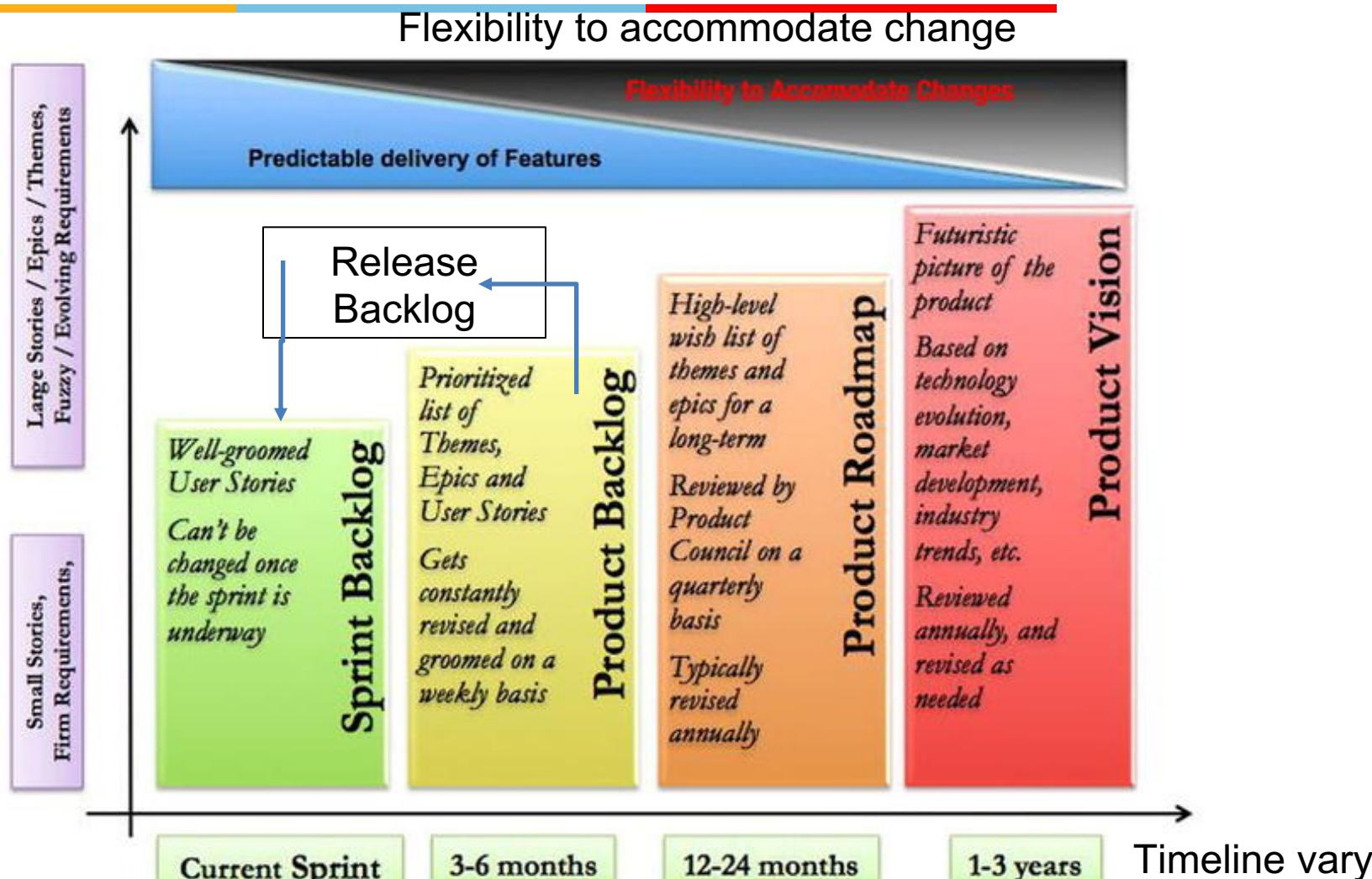
Characteristics of a Product Backlog



- There is an abbreviation that combines similar characteristics of good product backlogs. This is DEEP:
 - **Detailed appropriately**
 - higher-priority items are described in more detail than lower-priority ones
 - **Emergent**
 - It evolves and its contents change frequently. New items emerge based on customer and user feedback and are added to the backlog. Existing items are modified, reprioritized, refined, or removed on a regular basis.
 - **Estimated**
 - The product backlog items—certainly the ones participating in the next major release—should be estimated. The estimates are coarse-grained and often expressed in story points or ideal days.
 - **Prioritized**
 - All items in the product backlog are prioritized (or ordered)

Source: <https://www.romanpichler.com/blog/make-the-product-backlog-deep/>
Copyright © Pichler Consulting

Product runways represent a healthy trade-off between flexibility and predictability



Ref: Agile Product Development: How to Design Innovative Products That Create Customer Value by Tathagat Varma published by Apress, 2015

Project Trade-off Matrix

	Fixed	Flexible	Accept
Scope	X		
Schedule		X	
Cost			X

- The tradeoff matrix helps the development team, the product team, and the executive stakeholders manage change during a project.
- The trade-off matrix informs all participants that changes have consequences and acts as a basis for decision making.
- The trade-off matrix indicates relative importance of the three constraints (scope, schedule, cost) identified on the agile triangle (value, quality, constraints).
- The importance goes from Fixed, to Flexible, to Accept, the tolerance for variation increases.

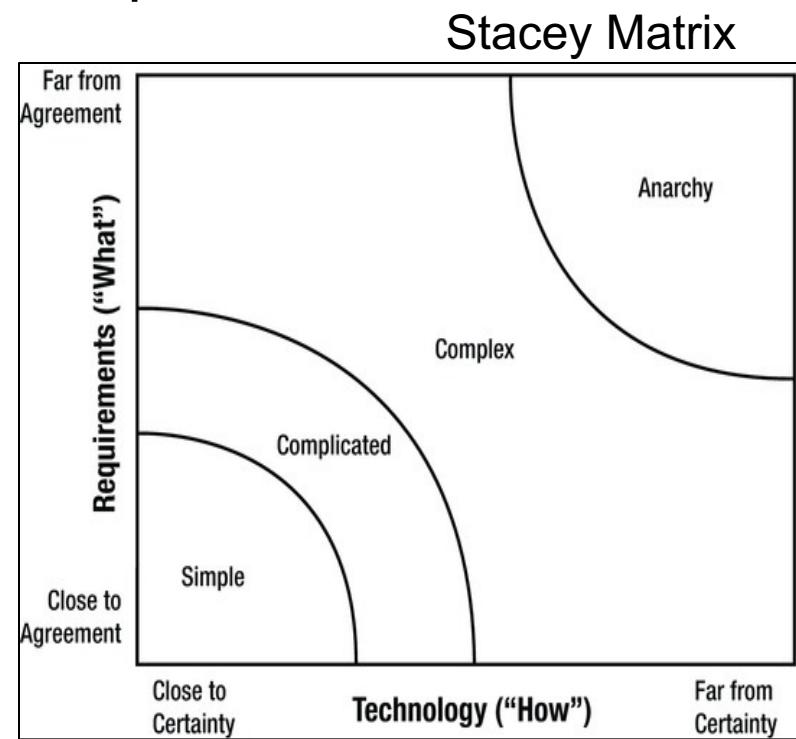
Ref: Agile Project Management: Creating Innovative Products, Second Edition by Jim Highsmith Published by Addison-Wesley Professional, 2009

Exploration Factor

- Articulating an exploration factor helps considerably in managing customer and executive expectations.

Product Technology Dimension					
Product Requirements Dimension	Bleeding Edge	Leading Edge	Familiar	Well-known	
Erratic	10	8	7	7	
Fluctuating	8	7	6	5	
Routine	7	6	4	3	
Stable	7	5	3	1	

Category	Requirements Variability
Erratic	25–50% or more
Fluctuating	15–25%
Routine	5–15%
Stable	<5%



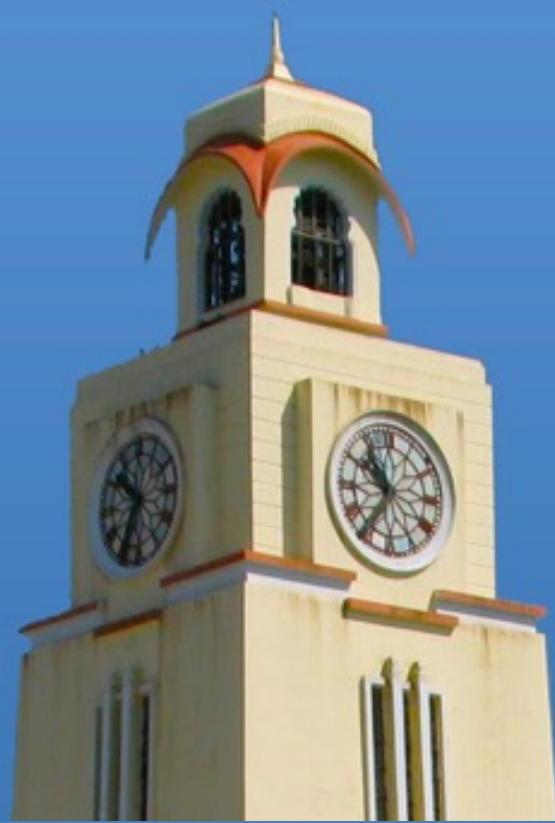
End



BITS Pilani
Pilani Campus

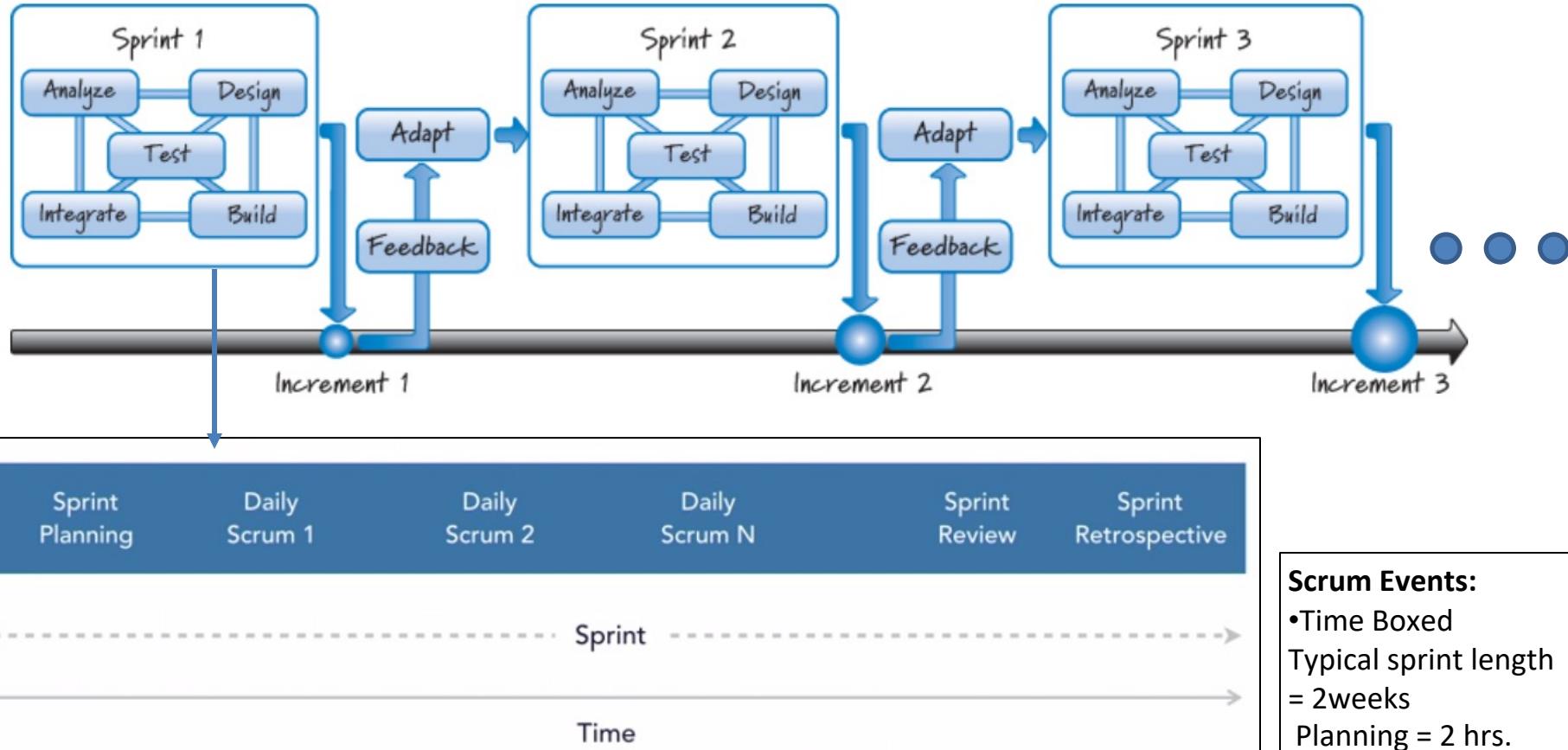
BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



SE CZ 544 , Agile Software Process Module – 7 Sprint Planning

Sprint Overview



Every Sprint is a learning Opportunity
Constructivism - Approach

Inputs for Iteration Planning

- Product backlog—with enough ready stories for about two iterations
- Quarterly (release) roadmap
- Definition of ready (DoR), where used; definition of done (DoD)
- Past performance (velocity), if available
- Availability of team members during the upcoming iteration (**Capacity Planning**)
- Process improvement tasks—added to the iteration backlog during the last iteration retrospective

Outputs of Iteration Planning

- Planning Part1
 - Sprint goal
 - Discuss Stories
 - Negotiate, Forecast/Identify stories
- Planning Part2
 - Identify developer tasks
 - Sign up for tasks
 - Estimate tasks
 - Negotiation
 - Commitment
- Task Board

Capacity = Velocity \pm Changed Circumstances
Capacity = Total Potential Person-Days \times Availability – Slack

Definition of ‘Done’

- **Completion Criteria** – Defined by Development Team
 - Code Complete and Checked in
 - Unit test done
 - Peer reviewed
 - QA Complete
 - Documentation done
- **Acceptance criteria** – Defined by Product owner
 - PO determines when the acceptance criteria s done
 - At this point, the story is considered accepted or “DONE”

Quiz

- » <https://forms.gle/UChV4y1fLnjfvBJ57>
- » <https://forms.gle/aG5AcsK5CcPrARAXA>
- » <https://forms.gle/aG5AcsK5CcPrARAXA>
- » <https://forms.gle/ZuQG7Fw6Sp8y6YsFA>

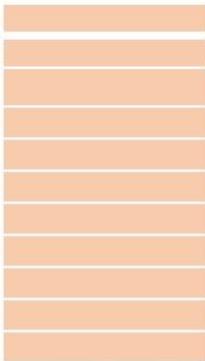


Sprint Planning – Additional Notes

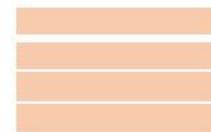
Sprint planning

- Sprint planning happens **once in each Sprint**
- Entire Scrum Team: Product Owner, Scrum Master & Development Team must be a part of the event
- Time box for the event is 8 hours (upper limit)
- Two agendas
 - **What is to be done** in this Sprint-Part1
 - Establish **how it will be done**-Part2

Product Backlog



Sprint Backlog



Product Owner discusses the product backlog with the development team



Product Owner

What is to be done?

- Sprint Goal/Objectives
- Responsibility: PO and Team

How it will be done?

- The team pulls subset of stories from Product backlog
- Spilt into Engineering tasks and estimates.
- Check against team velocity/capacity, Refine goal.
- The team Plan is not fixed, it evolves as more clarity increases
- Development team will come up with the plan

Sprint Planning Artifacts

- There are two defined artifacts that result from a sprint planning meeting:
 - A sprint goal
 - A sprint backlog
- A sprint goal is a short, one- or two-sentence, description of what the team plans to achieve during the sprint. It is written collaboratively by the team and the product owner.
- The following are example sprint goals on an eCommerce application:
 - Implement basic shopping cart functionality including add, remove, and update quantities.
 - Develop the checkout process: pay for an order, pick shipping, order gift wrapping, etc.

Examples of Sprint goals

- “Demonstrate the ability to send a text message through an integrated software, firmware, and hardware stack.”
- “Update mobile apps for faster convenient check-in for our valued customers”
- “Learn about the right user interaction” for the registration feature” (Learning goal, Risk Reduction)
- A non-optimal example of sprint goal is:
 - Implement all user stories to meet the definition of done, and fix all defects selected for this sprint. This sprint goal is too generic to be of any value in limiting the scope of work, and cannot inspire the team.

Sprint Planning : Part-1 – Defining the Sprint Goal - The Process



1. Product owner starts first with this kind of information



Bob
(product owner)

2. Product owner shows the Prioritized product backlog with this goal in mind – High value story at the top.

Business stakeholders want to increase membership. Our mobile apps are very basic and need to provide more functionality.



User story 12: As a potential club member, I should be able to sign up as a trial member and print temporary badge so I can try fitness center facilities.

User story 13: As an internet user, I should be able to view the calendar of activities at the fitness club so I can sign up for the activities.

Bug 8: Menu at the top of the website's home page seems to overlap with other visual controls on <X> tablet <Y> browser.

User story 17: As a fitness club member, I should be able to generate membership badge on my mobile phone so I can check in without my physical membership card.

User story 18: As a fitness club member, I should be able to book a tennis court or racquetball court from the mobile app.

User story 21: As an internet user, I should be able to view a list of sports facilities at the fitness club so I can make a decision on joining the club.

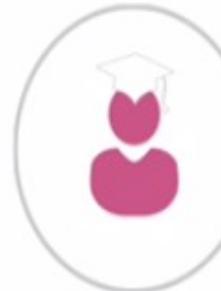
User story 21: As an internet user, I should be able to view a list of sports facilities at the fitness club so I can make a decision on joining the club.

Bug 3: Font size on the "Contact us" page looks different from the rest of the website.

User story 25: As an internet user, I should be able to view a list of fitness equipment types at the fitness club so I can make a decision on joining the club.



Development team



Ashley
(scrum master)

Sprint Planning – Part-2 – The How

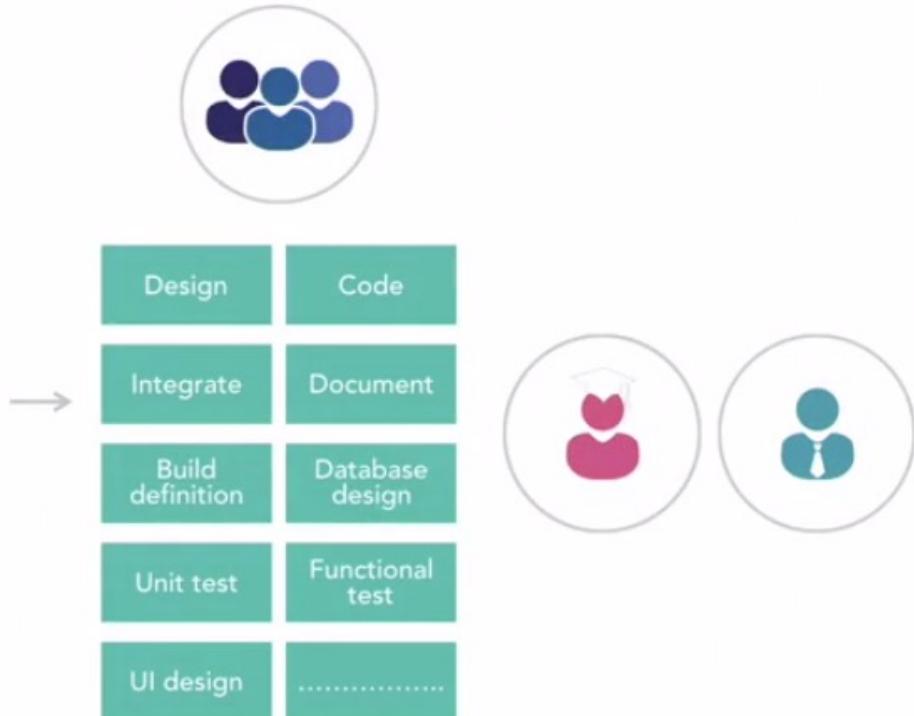


- Part-2 of the sprint planning is owned by the Development team.
- The Dev. Team focuses on detailed planning – splitting user stories into engineering/programming tasks.
- **Tasks are estimated in Ideal hours.**
- Product owner and Scrum master are available to facilitate and answer any questions.

Sprint Planning – Part-2 – Splitting into fine grain tasks



- User story 12:** As a potential club member, I should be able to sign up as a trial member and print temporary badge so I can try fitness center facilities.
- User story 13:** As an internet user, I should be able to view the calendar of activities at the fitness club so I can sign up for the activities.
- Bug 8:** Menu at the top of the website's home page seems to overlap with other visual controls on <X> tablet <Y> browser.
- User story 17:** As a fitness club member, I should be able to generate membership badge on my mobile phone so I can check in without my physical membership card.
- User story 18:** As a fitness club member, I should be able to book a tennis court or racquetball court from the mobile app.
- User story 21:** As an internet user, I should be able to view a list of sports facilities at the fitness club so I can make a decision on joining the club.
- User story 21:** As an internet user, I should be able to view a list of sports facilities at the fitness club so I can make a decision on joining the club.
- Bug 3:** Font size on the "Contact us" page looks different from the rest of the website.
- User story 25:** As an internet user, I should be able to view a list of fitness equipment types at the fitness club so I can make a decision on joining the club.



Note: This is not an elaborate plan for the entire sprint. This is just a collection of tasks for the next few days. More planning will be done by the development team as they finish tasks and learn more about the tasks at hand. - **em-pir-i-cism, Last responsible moment.**

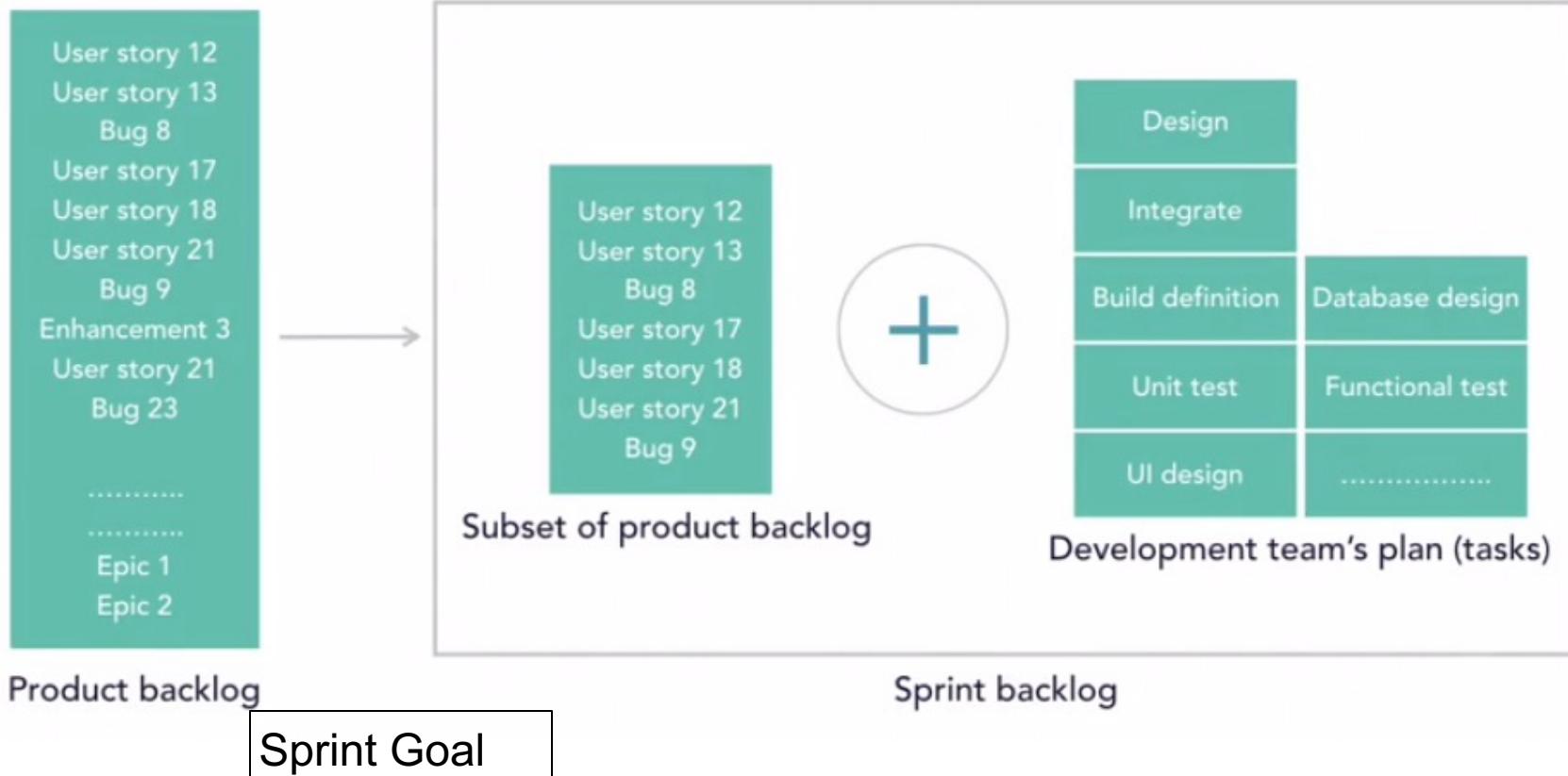
- Tasks are not assigned to the team by PO or SM. Pulled by dev. Team.

The Sprint Planning Ends with Sprint Backlog



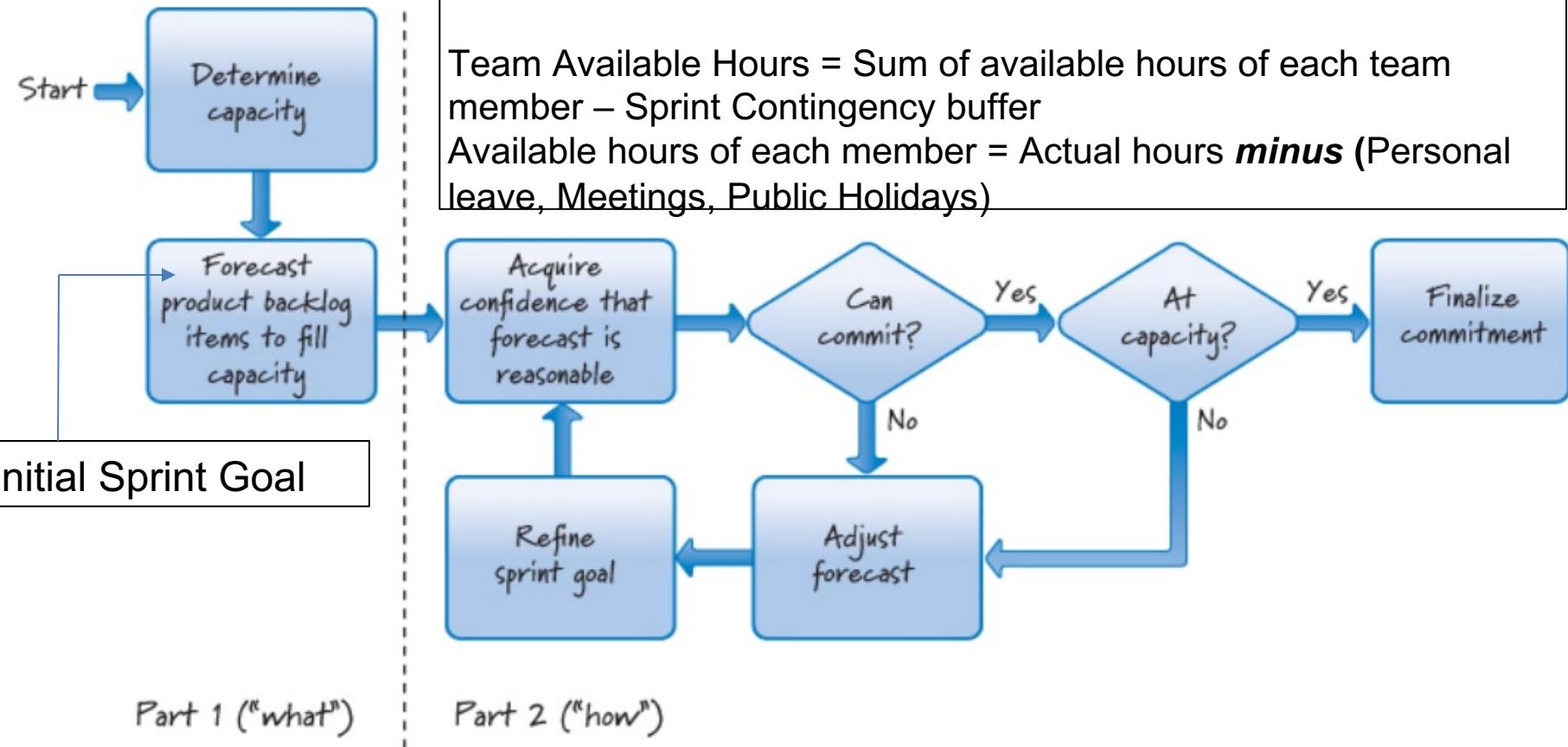
+ Team Commitment + Refined Sprint Goal

- Most Team break all user stories pulled for the sprint into tasks and estimate in ideal hours. Then compare with team capacity to establish confidence



Ref: LinkedIn Learning - Agile Software Development: Scrum for Developers with Shashi Shekhar.

The Process flow



Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

Potentially Shippable Product Increment (PSI)

- Each sprint must end with a potentially shippable increment.
- Product owner decides when to release the increment to user community (immediately or later).
- The **product increment** needs to be:
 - “Vertically sliced” portion of product that provides end-to-end functionality
 - Usable in production; provides business value
 - Good example: allows a user to search for a product by product name
(User interface to search -> Application layer components -> Database schema)
 - Bad examples: database schema, mocked user interface

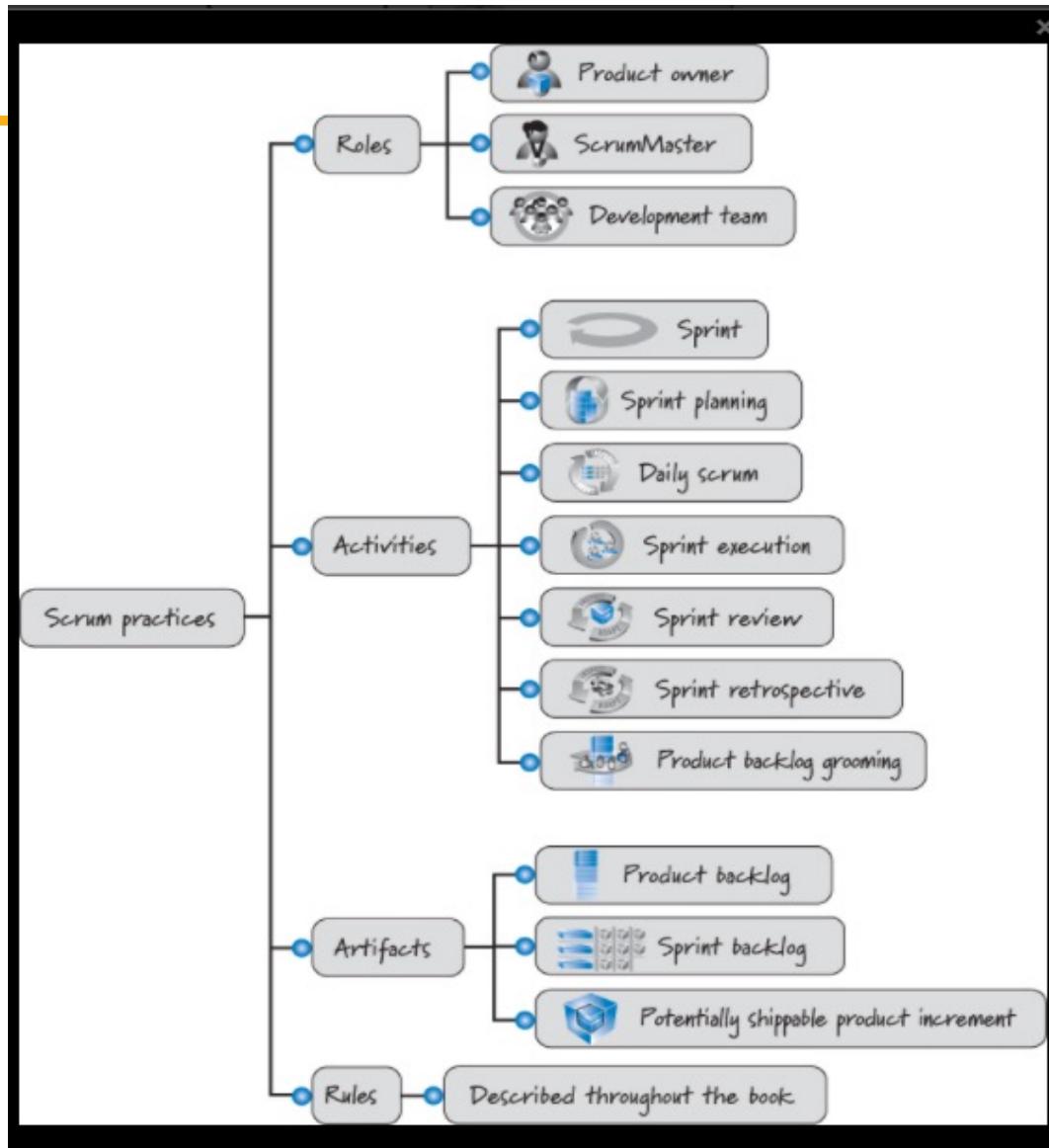
Sprint Practices

- **Sprint 0:**
 - Some team just focuses on planning/design and does not produce a working product increment. Sprint zero. - **Not a Good Practice**
 - Instead combine planning with some functionality delivery. – **Good practice.**
- **Hardening Sprint:**
 - Hardening sprints are designated for stabilizing products by fixing quality and performance issues.
 - The problem with this approach is that it encourages teams to produce unstable products at the end of sprints that do not provide expected business value and reduce transparency about a team's progress status, - **Not a Good practice**

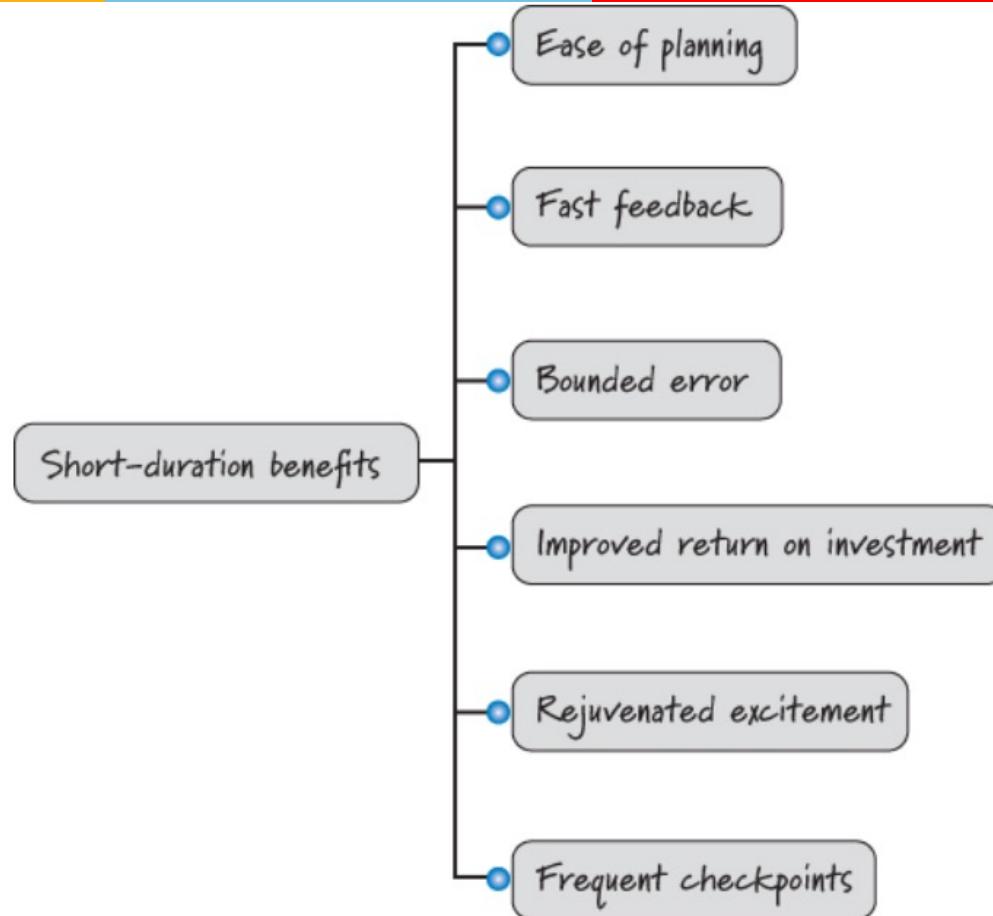
Spike - Story

- Spikes are research activities that are sometimes performed by Scrum teams.
- For example, evaluating a set of products/Technical solution to find the best solution for a specific business need.
- Spikes are allowed in Scrum but the golden rule is to combine spike activities with other development activities,
- Avoid sprints compromised entirely of spikes- **Good Practice**
- This is the era of **continuous delivery** where organizations release features multiple times during the day – This practice is acceptable within Sprint

Scrum Practices

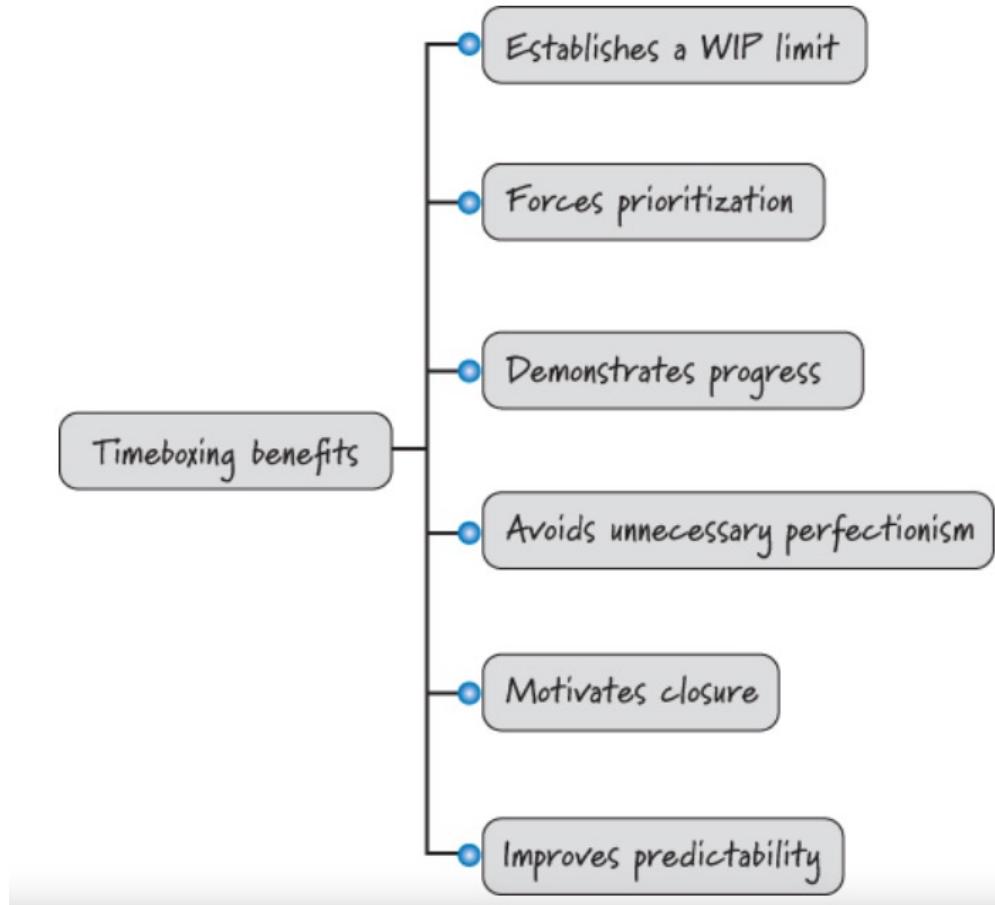


Short Sprints

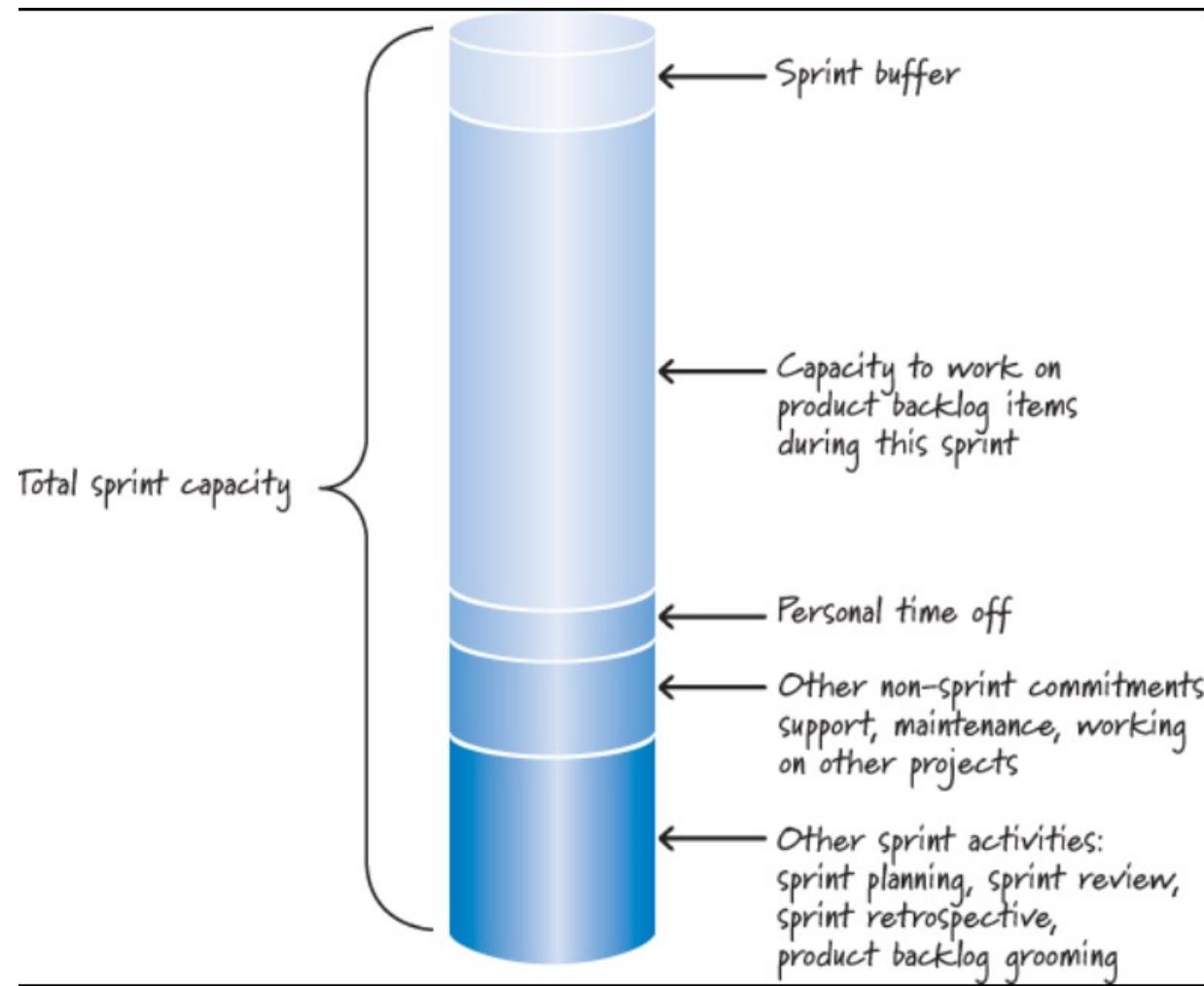


Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

Time Boxing



Development team capacity in a sprint



Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

Capacity in Story Points

- Initial estimate of its capacity/velocity for the upcoming sprint:
 - Start with the team's long-term average velocity.
 - Sometimes referred to as the “yesterday's weather” approach.
- Example:
 - Suppose Average Velocity = 40 Story points for 2 weeks sprint
 - Consider whether the upcoming sprint might differ from typical or previous sprints (it might not).
 - The result is a reasonable adjusted capacity (predicted velocity) for the upcoming sprint.
 - Adjust the velocity if the sprint is planned during long holidays (Year end holidays).

Capacity in Effort- Hours (Two Weeks Sprint) - Example

Team Members	Days Available (Less Personal Time)	Days for Other Scrum Activities	Hours per Day	Available Effort-Hours
1	10	2	4-7	32-56
2	8	2	5-6	30-36
3	8	2	4-6	24-36
4	9	2	2-3	14-21
5	10	2	5-6	40-48
				140-197

Caution: Taking 197 hours of work because it would leave no sprint buffer.

Better strategy: > 140 hrs. and < 197 hrs.

- If all team members are available full time and no personal holidays – Capacity = (Available Capacity) – (Total Days for Scrum activities) – Sprint buffer (Assume 5%)
$$= 400 - 80 - 20 = 300 \text{ hours} - \text{plan for 320-300 hours of capacity}$$

Selecting Product Backlog Items



- If we have a sprint goal, we would select product backlog items that align with that goal.
- If there is no formal sprint goal, our default is to select items from the top of the product backlog. We would start with the topmost item and then move to the next item and so forth.
- If the team were not able to commit to the next-highest-priority item (perhaps there is a skills capacity issue), it would select the next appropriate higher-priority backlog item that looks as if it can be completed within the constraints.
- Also, having a **good definition of ready** will prevent product backlog items from being selected that are poorly defined or have unfulfilled resource or dependency constraints that would prevent our finishing them in a sprint.
- The start-only-what-you-can-finish rule is based on the principles that we should limit WIP and that starting something and not finishing it generates a variety of forms of waste.

Examples of Product Backlog Items (PBI)

PBI Type	Example
Feature	As a customer service representative I want to create a ticket for a customer support issue so that I can record and manage a customer's request for support.
Change	As a customer service representative I want the default ordering of search results to be by last name instead of ticket number so that it's easier to find a support ticket.
Defect	Fix defect #256 in the defect-tracking system so that special characters in search terms won't make customer searches crash.
Technical improvement	Move to the latest version of the Oracle DBMS.
Knowledge acquisition	Create a prototype or proof of concept of two architectures and run three tests to determine which would be a better approach for our product.

Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

Acquiring Confidence

- Use predicted velocity to see if the commitment is realistic.
 - If predicted sprint velocity is 25 story points and our team has selected 45 story points' worth of work, the team should be concerned.
- The risk of using velocity as the sole means of establishing confidence is that even though the numbers look right, the commitment might still be unachievable.
 - However, until we dig a little deeper to the task level, we don't really know if the set of product backlog items that total 21 story points can actually be completed—there could be dependency issues, skills capacity issues, as well as a host of other issues that make it impractical for the team to get them all done.

Acquiring Confidence ...

- Most Scrum teams gain the necessary level of confidence by breaking the product backlog items down into the tasks that are required to complete them to the Scrum team's agreed-upon definition of done.
- These tasks can then be estimated (usually in effort-Ideal hours) and subtracted from the team's capacity.
- Breaking product backlog items into tasks is a form of design and just-in-time planning for how to get the items done.
- The result is a sprint backlog

Definition of Ready

- You can think of the definition of ready and the definition of done as two states of product backlog items during a sprint cycle.
 - State-1 : Before the start Sprint planning
 - State-2: After the item considered as done

Example:
Definition of
ready

Definition of Ready	
<input type="checkbox"/>	Business value is clearly articulated.
<input type="checkbox"/>	Details are sufficiently understood by the development team so it can make an informed decision as to whether it can complete the PBI.
<input type="checkbox"/>	Dependencies are identified and no external dependencies would block the PBI from being completed.
<input type="checkbox"/>	Team is staffed appropriately to complete the PBI.
<input type="checkbox"/>	The PBI is estimated and small enough to comfortably be completed in one sprint.
<input type="checkbox"/>	Acceptance criteria are clear and testable.
<input type="checkbox"/>	Performance criteria, if any, are defined and testable.
<input type="checkbox"/>	Scrum team understands how to demonstrate the PBI at the sprint review.

Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

Definition of Done

- An Example

Definition of Done	
<input type="checkbox"/>	Design reviewed
<input type="checkbox"/>	Code completed <ul style="list-style-type: none"> Code refactored Code in standard format Code is commented Code checked in Code inspected
<input type="checkbox"/>	End-user documentation updated
<input type="checkbox"/>	Tested <ul style="list-style-type: none"> Unit tested Integration tested Regression tested Platform tested Language tested
<input type="checkbox"/>	Zero known defects
<input type="checkbox"/>	Acceptance tested
<input type="checkbox"/>	Live on production servers

- Conceptually the definition of done is a checklist of the types of work that the team is expected to successfully complete before it can declare its work to be potentially shippable.
- Can be applied to product backlog item, Increment or a release.
- Obviously the specific items on the checklist will depend on a number of variables:
 - The nature of the product being built
 - The technologies being used to build it
 - The organization that is building it
 - The current impediments that affect what is possible
- Definition of Done Can Evolve Over Time

Definition of Done ...

- Most of the time, a bare-minimum definition of done should yield a complete slice of product functionality, one that has been designed, built, integrated, tested, and documented and would deliver validated customer value.
- To have a useful checklist, however, these larger-level work items need to be further refined.
 - For example, what does tested mean? Unit tested? Integration tested? System tested? Platform tested? Internationalization tested? You can probably think of many other forms of testing that are specific to your product. Are all of those types of testing included in the definition of done?
- Scrum teams need to have a robust definition of done, one that provides a high level of confidence that what they build is of high quality and can be shipped. Anything less robs the organization of the business opportunity of shipping at its discretion and can lead to the accrual of technical debt

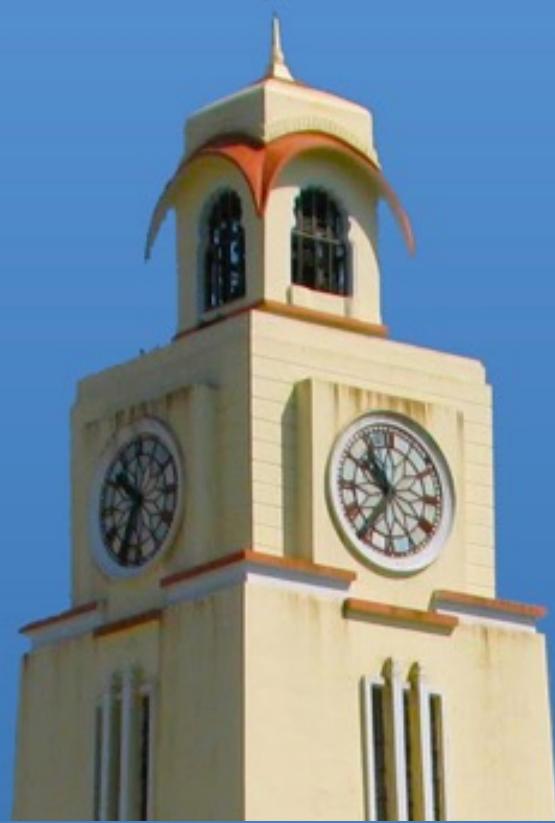
Thank you



BITS Pilani
Pilani Campus

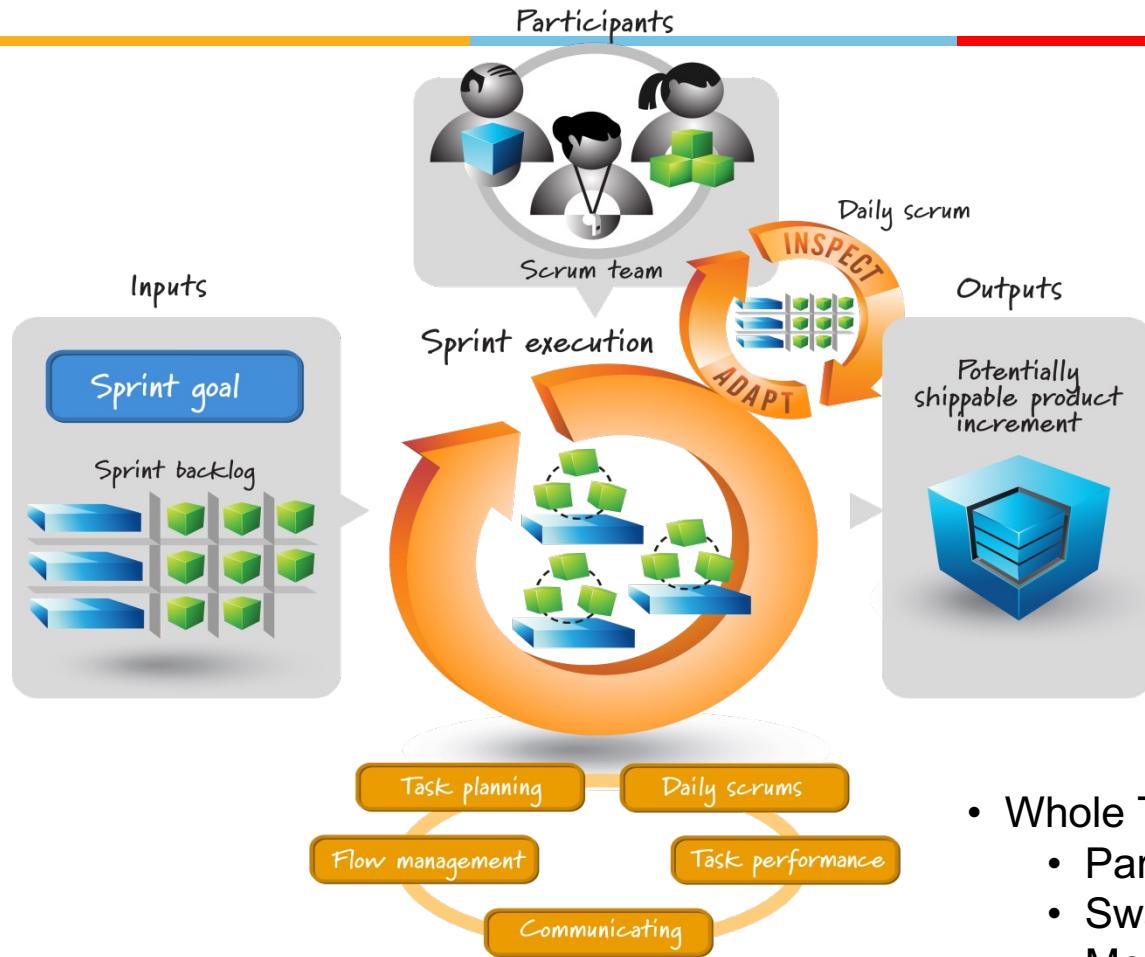
BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



Module8- Executing a Sprint

Sprint Execution



- Whole Team
 - Paring
 - Swarming
 - Mob Programming

Copyright © 2012, Kenneth S. Rubin and Innolution, LLC. All Rights Reserved.

Daily Scrum

- Dev. Team, Scrum master, Product Owner (Optional),

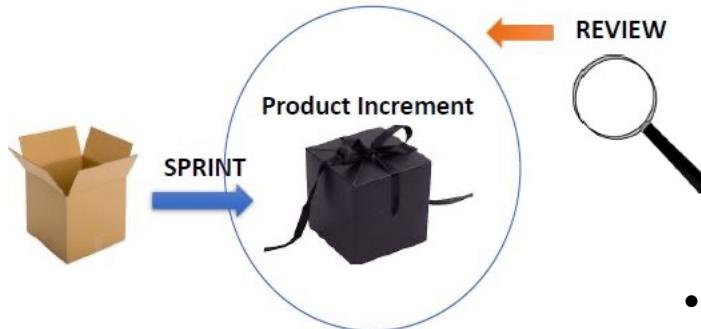


15-minutes everyday
Same place, same time

3 important questions in Daily Scrum

- What did I do yesterday?
- What will I do today?
- Any difficulties or impediments stopping me from the Sprint goal?

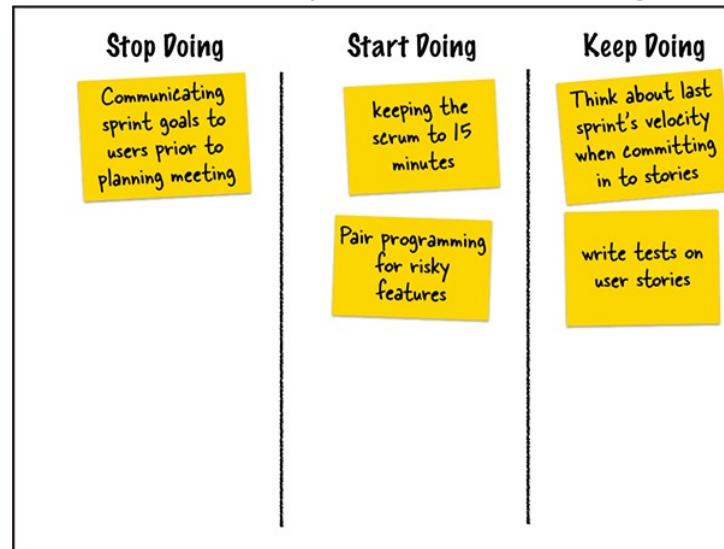
Sprint Review



- Duration: Max 4 hrs. or less for 4 week sprint
- Product invites all attendees
- Development Team Demo the completed items.
- Product owner verifies and lists completed and incomplete items.
- Development team share sprint experience and highlights challenges.
- The Product Owner updates the Product Backlog and discusses the next sprint's activities.

Sprint Retrospective

- Simplest Approach
- Each team member is asked to identify specific things that the team should:
 - Start doing
 - Stop doing
 - Continue doing

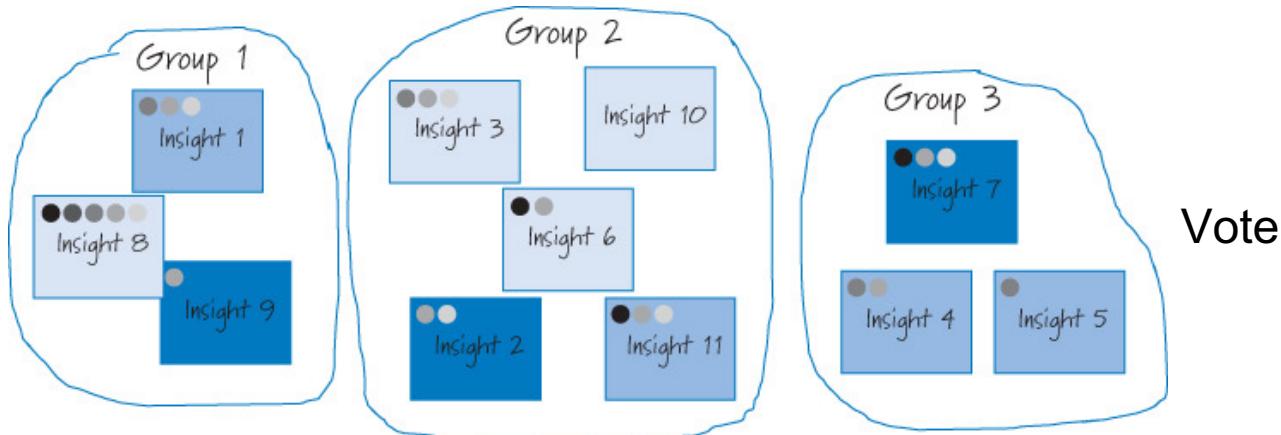


The team had a problem with setting user expectations before they had a planning meeting, so they want to make sure to stop doing that in the next sprint.

- Note:
 - Choosing Retrospective Topics
 - Use retrospective games, Videos, Movie
 - Use abstraction
 - Avoid-Technical and Management issues

Sprint Retrospective

- Group insights



Determine Actions

Start Doing	Stop Doing	Continue Doing
Recalculate velocity after each iteration	Allow the daily stand-up meeting to last more than 15 minutes	Team lunch on Fridays
Enroll the team in a clean coding course	Write new code when unresolved defects exist	Retrospectives
Encourage testers to pair program with developers		Customer feedback sessions

Communicating: The Progress

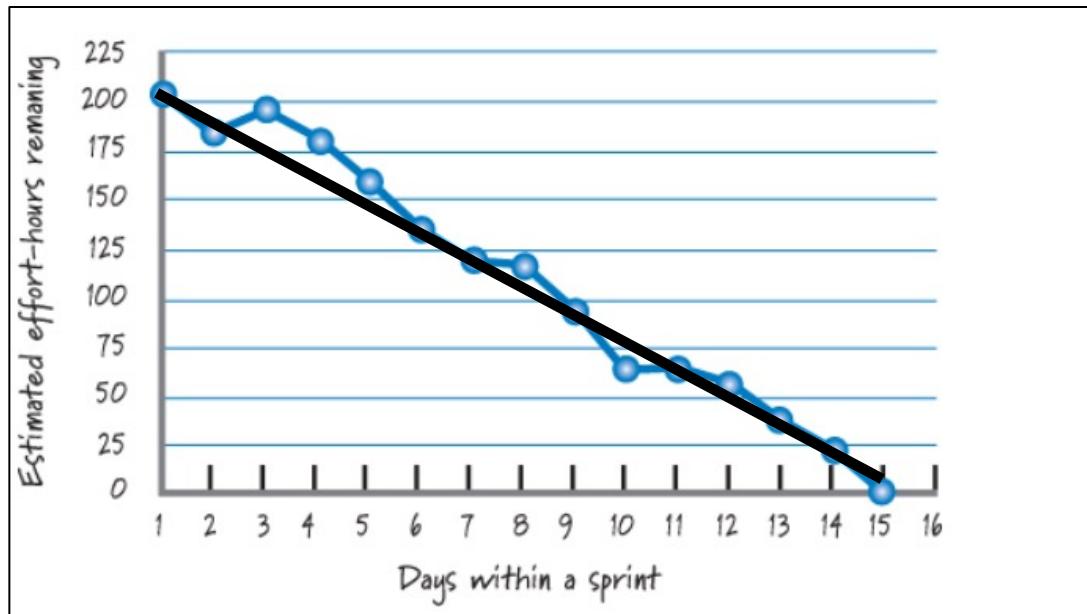
- Information radiator
 - Self interpretable format
- Task Board
- Burndown charts
- Burnup charts

Sprint Task Board

Story	To Do	In Process	To Verify	Done
As a user, I... 8 points	Code the... 9 Code the... 2 Test the... 8	Test the... 8 Code the... 8 Test the... 8	Code the... DC 4 Test the... SC 8	Code the... D 8 Test the... SC 8 Test the... SC 8 Test the... SC 6
As a user, I... 5 points	Code the... 8 Code the... 4	Test the... 8 Code the... 6	Code the... DC 8	Test the... SC 8 Test the... SC 6

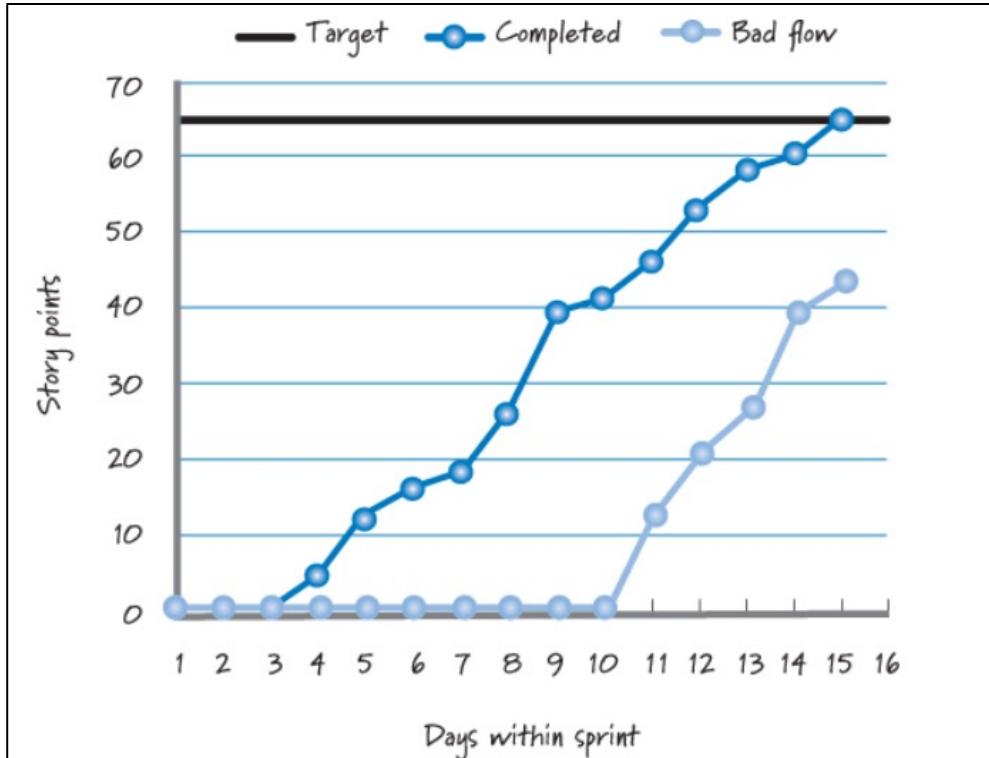
Sprint burndown

Effort <> Days



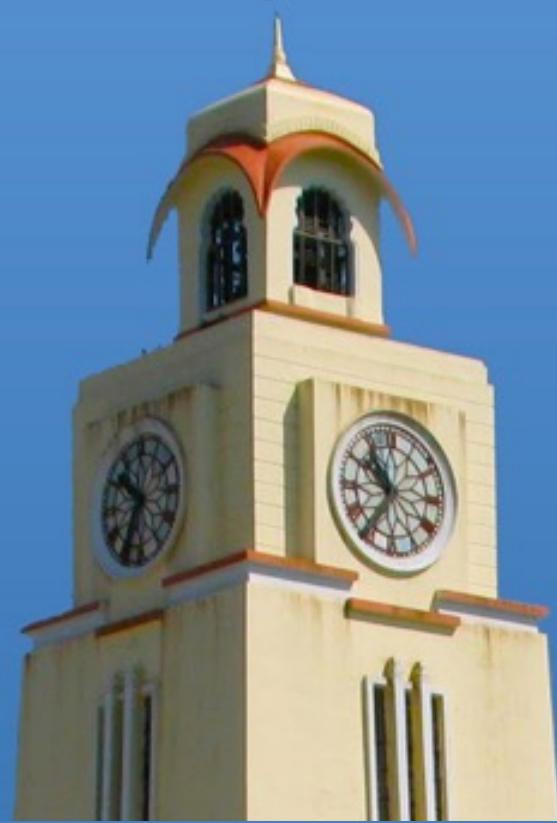
Sprint burndown

Story Points <> Days



Quiz

- » <https://forms.gle/yRbPFB8Yv9YckeX7>
- » <https://forms.gle/Q3QpLLYGuBTZkZeo7>
- » <https://forms.gle/uRZvRQDwtUCqTwEW7>
- » <https://forms.gle/BtDEjG19sCrqvfZF8>



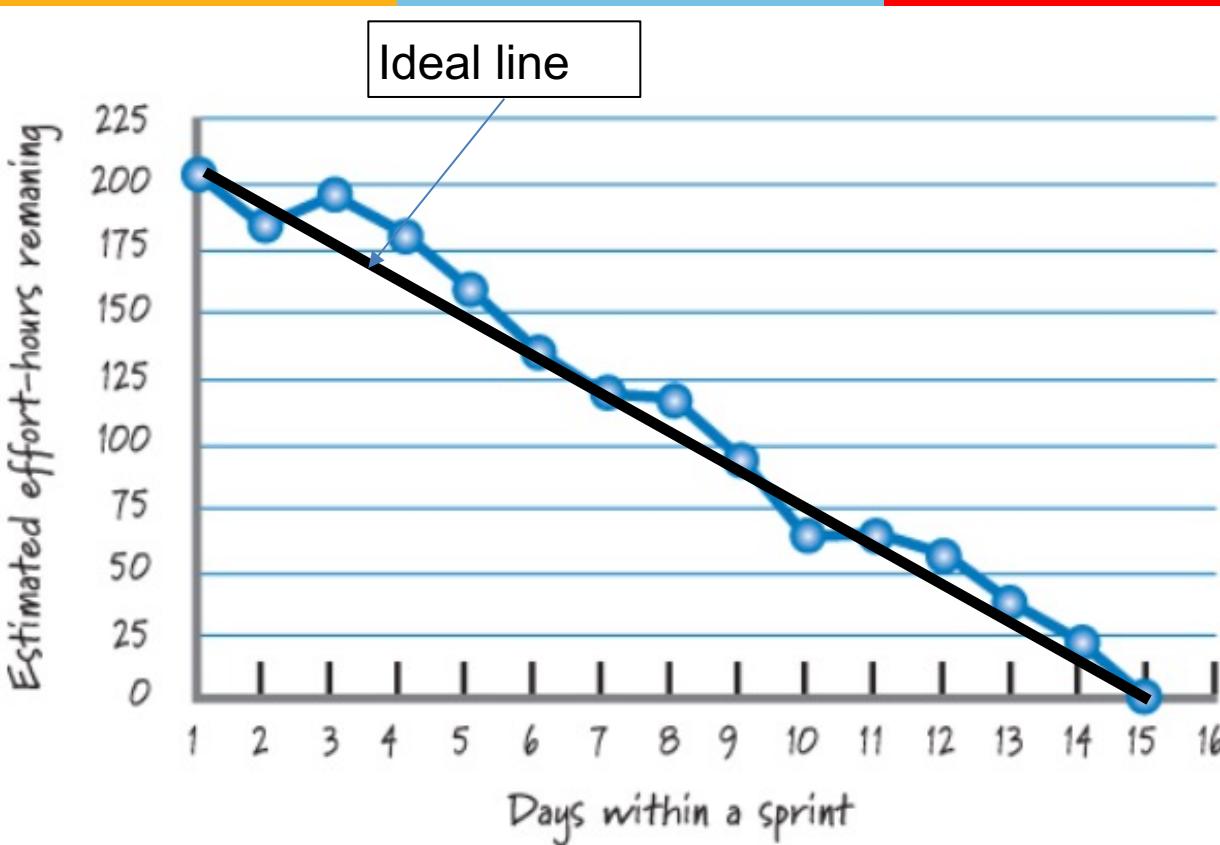
Additional Notes

Sprint Execution: The Process



- Task planning
 - No Gantt chart, Just-in-time, dependency planning (e.g. Stress testing)
- Flow Management
 - Team responsibility to **organize the flow work**, what should be done next and in parallel. Don't aim to make everyone 100% busy.
 - Parallel work and swarming
 - Which item to start? What Tasks Needs to Be Done? – Height priority/Value.
 - How to organize the work? - No hands-off approach, no artificial wall across boundaries (e.g. UX/ Business logic/Database).
 - Who does the work? – Person with appropriate skills.

Sprint burndown chart- Effort



X-Axis: Represents the days within a sprint.

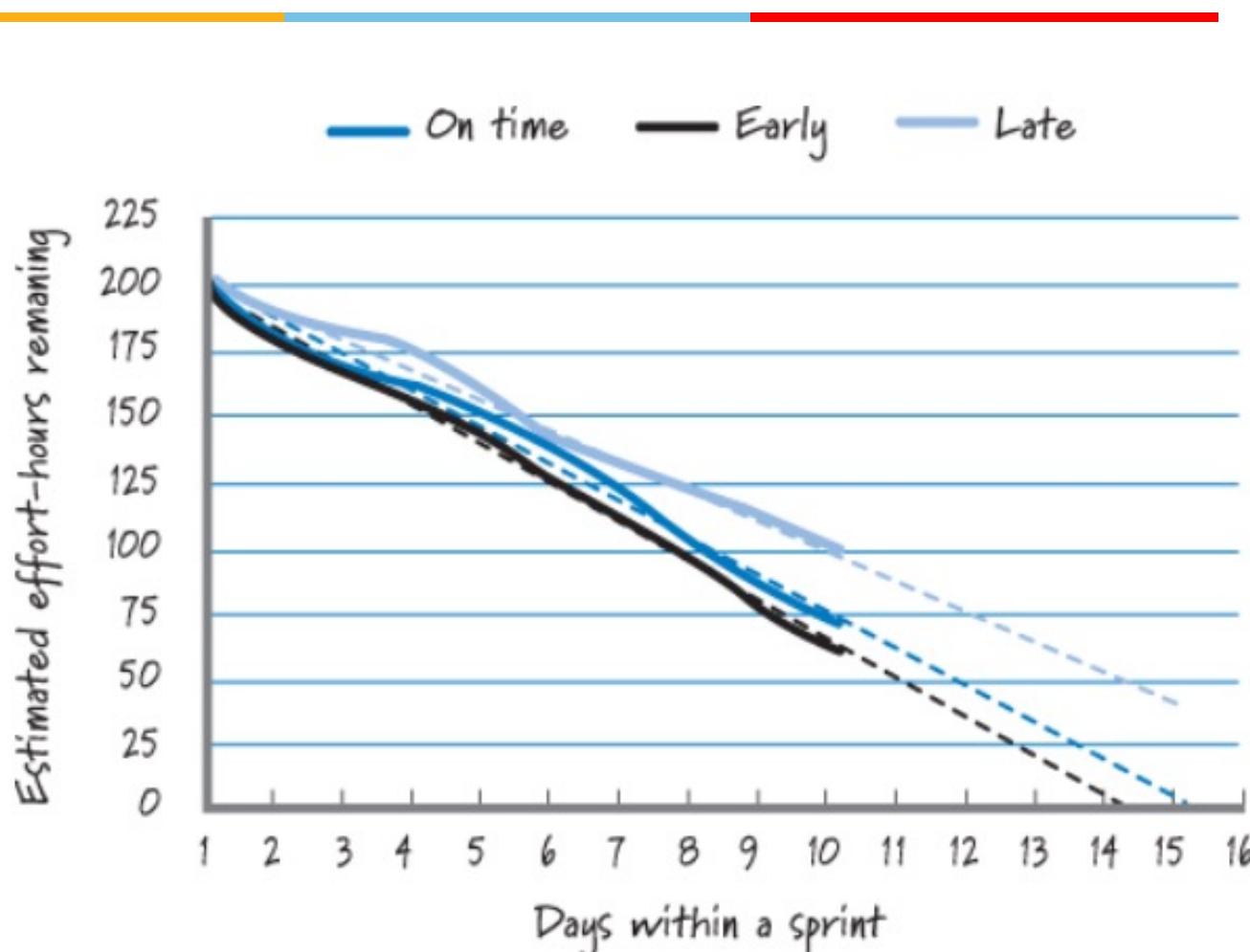
Y-Axis: Remaining estimated effort-hours

Should be updated every day

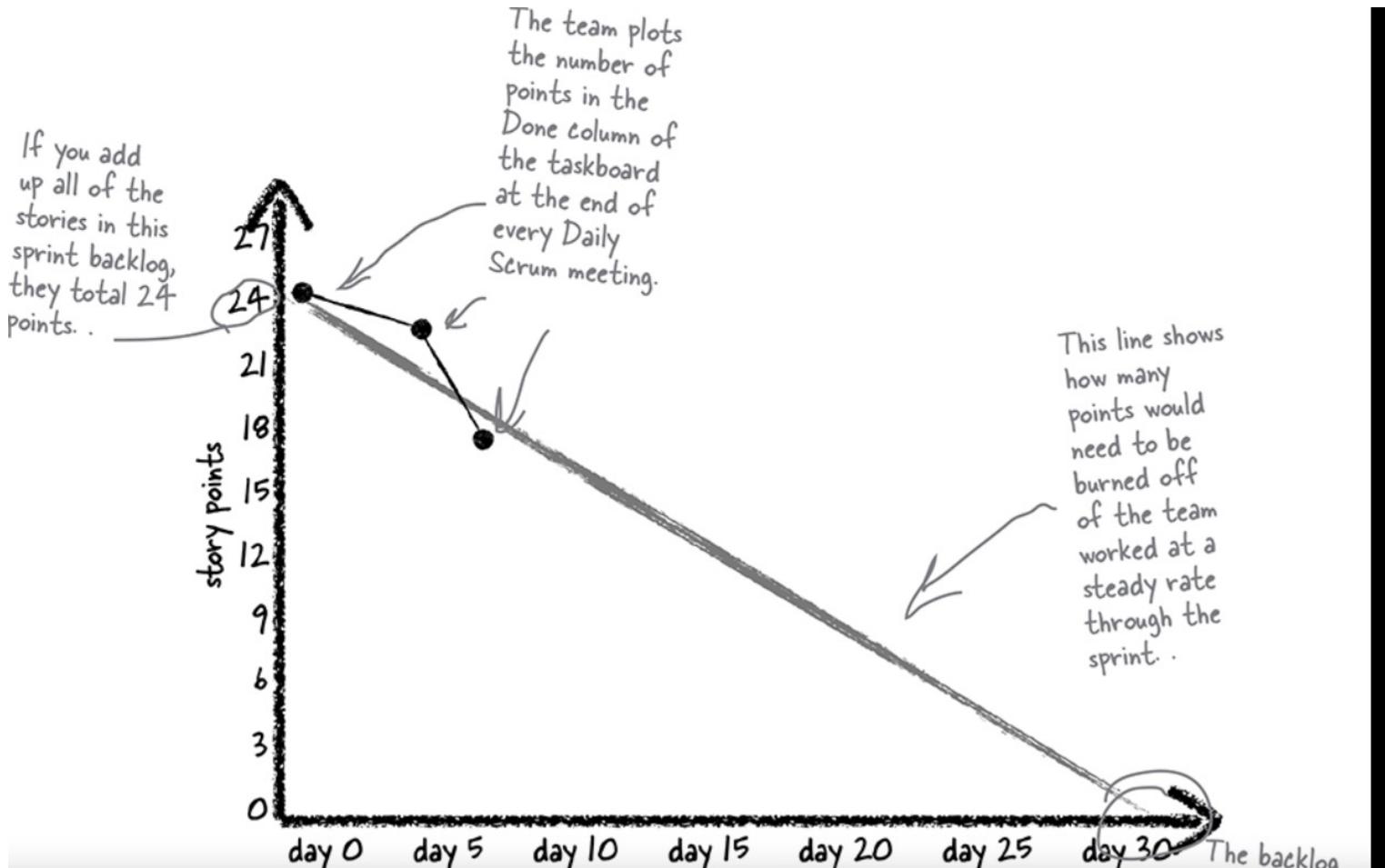
- Charts shows the trend – likelihood of completing the work by end of the sprint

- Only shows the number of Story points/Efforts that have been completed. The burndown chart doesn't show any changes in the scope of work. For that, We use burnup charts.

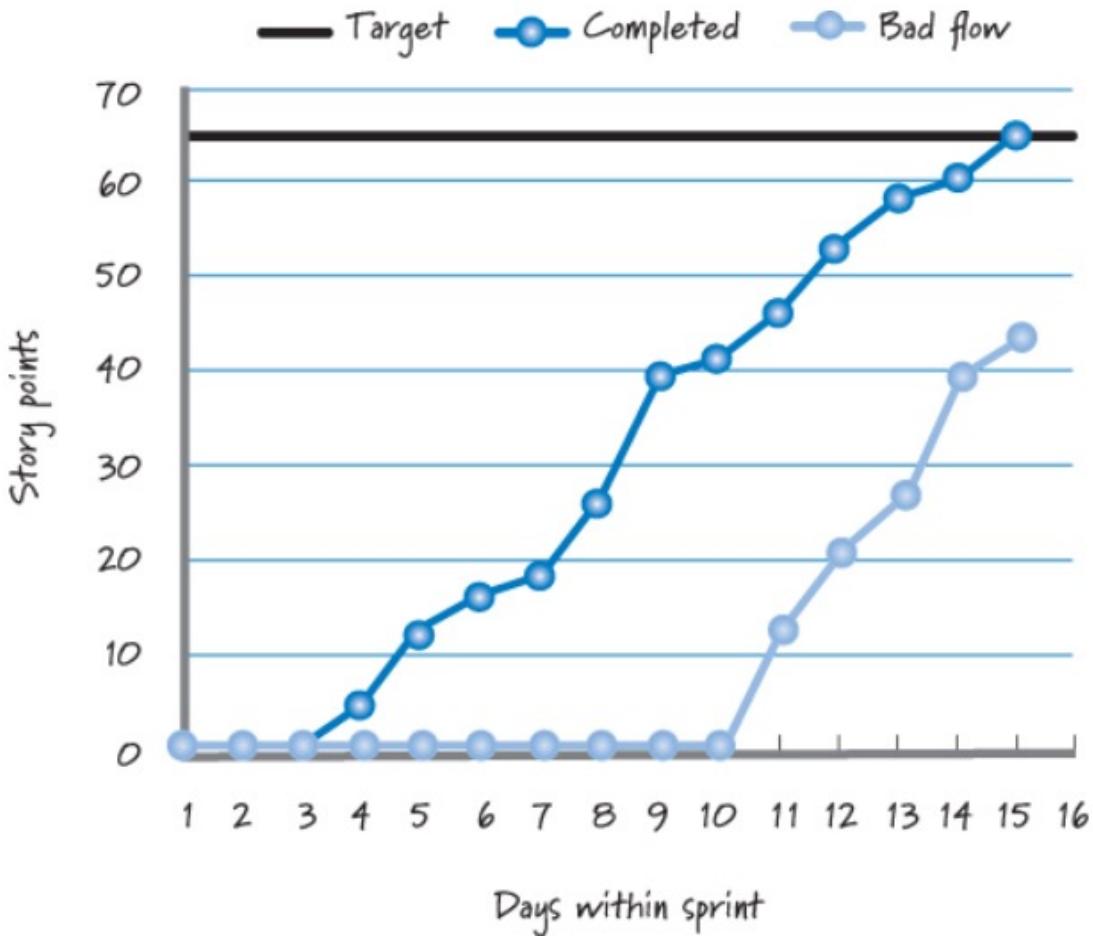
Sprint burndown chart with trend lines



Sprint burndown chart- Story points



Sprint burnup chart



- Many people prefer to use story points in their burnup charts, because it represents business value
- **The “Bad flow” line illustrates** - the team starts too many product backlog items at the same time, doing too much work in parallel.
- Works on product backlog items that are large and therefore take a long time to finish, or takes other actions that result in bad flow.

Sprint Execution: Daily Scrum team meeting

- Daily Scrum – 15 minutes or less, held at the same time everyday. Not a problem solving meeting.
- Also called daily standup to promote brevity.
- **ScrumMaster facilitating** and each team member taking turns answering three questions for the benefit of the other team members:
 - What have I done since our last meeting?
 - What am I planning on doing between now and our next meeting?
 - What roadblocks are in my way?
- The daily scrum is an inspection, synchronization, and adaptive daily planning activity that helps a self-organizing team do its job better.

Sprint Execution

- Sprint execution is the work the Scrum team performs during each sprint to meet the sprint goal.
- Performs all of the work necessary to deliver a potentially shippable product implement. The team's work is guided by the sprint goal and sprint backlog.
- We shall focus on the principles and techniques that guide how the Scrum team plans, manages, performs, and communicates during sprint execution.

Sprint Execution: Timing

- The majority of the team's time each sprint should be spent in sprint execution.
- It begins after sprint planning and continues until the sprint review begins.
- For a two-week sprint, sprint execution would likely count for 8 to 8.5 of the 10 days

Sprint Execution: Participants

- During sprint execution:
- The **development team** members self-organize to determine the best way possible to meet the sprint goal.
- The **ScrumMaster** coaches, facilitates, and removes any impediments that block or slow the team's progress.
- The **product owner** is available to answer questions, review intermediate work, and provide feedback to the team. The product owner might also be called upon to discuss adjustments to the sprint goal or to verify acceptance criteria.
- The ScrumMaster doesn't assign work to the team or tell the team how to do the work. A self-organizing team figures these things out for itself.

Sprint execution: Process Task Planning

- During sprint planning the team produces a high-level plan for how to achieve the sprint goal, usually in the form of a sprint backlog.
- Team members perform **just-in-time task-level planning** as needed, as opposed to trying to formulate a detailed plan or Gantt chart.
- Massive influx of learning comes from building and testing. This will disrupt even a well laid out plan.
- However, **some upfront planning helps** in exposing the task level dependencies
 - Example: if a feature being developed to be subjected two day long stress testing. Develop the feature and plan for the test at least two days before the end of sprint execution.

Sprint execution: Process Flow Management

- It is the team's responsibility to manage the flow of work throughout the sprint to meet the sprint goal.
- This includes making decisions about how much work the team should do in parallel, which work to start, how to organize the task-level work, which work to do, and who should do the work.
- When answering these questions, teams should discard old behaviors, such as trying to keep everyone 100% busy believing that work must be done sequentially, and having each person focus on just her part of the solution.
 - Example: Don't create artificial wall across technical boundaries (UX/Business logic/Backend work/Testing)
 - Sit together and discuss: How the work can be accomplished iteratively and efficiently

Flow Management: Parallel Work and Swarming

- The team must determine how many product backlog items to work on in parallel, in other words, at the same time. Working on too many items at once slows the team down. But working on too few items at once is equally wasteful. To find the proper balance, I recommend that teams work on the number of items that leverages but does not overburden the T-shaped skills and available capacity of the team members.
- The goal is to reduce the time required to complete individual items while maximizing the total value delivered during the sprint. Another name for this approach is swarming. Swarming is when team members with available capacity work together to complete one unfinished item rather than moving ahead to work on new items. This doesn't mean teams should always work on only one item at a time—the actual number of open items at any one time is highly dependent on context. Teams will have to experiment to find the balance that maximizes the value they deliver each sprint.

Flow Management: Which Items to start



- The simplest way to choose the product backlog item to work on next is to select the highest-priority item as specified by the product owner.
- Unfortunately, this doesn't always work. In reality, dependencies or skills capacity constraints might dictate a different order. The development team should be enabled to opportunistically select work as appropriate.

Flow Management: How to Organize the work?

- It's tempting for new agile teams to approach task level work in a waterfall fashion: design it, code it, and then test it.
- It's better, however, to approach the work from a value and delivery-focused mindset.
- This means minimizing the amount of time work sits idle and reducing the size of handoffs
- In practice, this sometimes looks like a developer and tester pairing at the start of a task to work in a highly interleaved fashion, with rapid cycles of test creation, code creation, test execution, and test and code refinement. This approach keeps work flowing, supports very fast feedback, and enables team members with T-shaped skills to swarm on an item to get it done.

Flow Management: What tasks need to done?

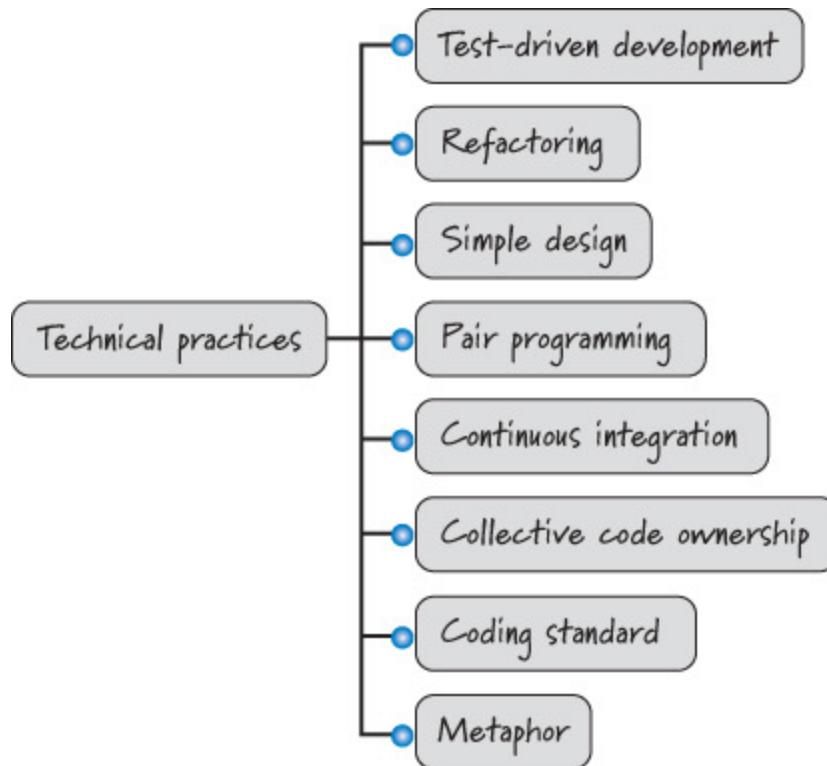
- The team should decide which task-level work it needs to perform to complete a product backlog item. Product owners and stakeholders influence these choices by defining the scope of a feature and creating acceptance criteria.
- They also provide business-facing requirements for the team's definition of done. Overall, the team and the product owner must work together to ensure that technical decisions with important business consequences are made in an economically sensible way.

Flow Management: Who does the work?

- Who should work on each task? An obvious answer is the person best able to quickly and correctly get it done. And if that person is unavailable, the team should decide on the next best person.

Task Performance: Technical Practices

- Development team members are expected to be technically good at what they do.



- Most teams achieve the long-term benefits of Scrum only if they also embrace strong technical practices when performing task-level work.

Communicating: The Progress

- Most teams use a combination of a task board and a burndown and/or burnup chart as their principal **information radiator**.
- **Information radiator:**
- A visual display that presents up-to-date, sufficiently detailed, and important information to passersby in an easy, self-interpretable format.

Thank you



BITS Pilani

Pilani Campus

BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



SE CZ 544 , Agile Software Process

CS-10 – Agile Metrics and Tools

Agile Metrics

- **Examples:**
 - Velocity, Lead time, Cycle time, Charts, Escape defects and so on.
- Helps to assess the quality of a product and track team performance.
- **The Concept:**
 - Define Metrics that can be used by Agile teams and Team management, Agile metrics that matter.
- **The Opportunity**
 - Reduced costs, Increase Product Quality, Increased team satisfaction
- **The Potential**
 - Auto Generate using exposed APIs provided by various PM tools.

Quantitative & Qualitative Metric

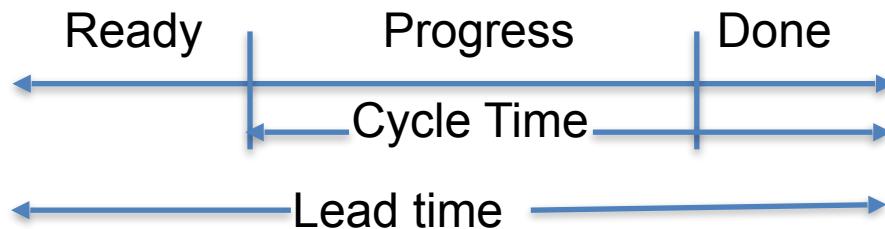


- Quantitative Metric
 - Measurement number: Lead time, Number of defects
- Qualitative Metric
 - Based on subjective opinion: Maintainability, Team happiness index ...

Source: <https://www.infoq.com/articles/metrics-agile-teams/>

An example Quantitative Metric

- Example: **Lead time** is a useful quantitative statistic for evaluating team performance.
- Determinant metrics:
 - A set of measurements related to a specific measurement
 - **Associated metrics:**
 - Flow efficiency (wait time)
 - Speeding tickets(%), (Tickets moves through multiple statuses)
 - Total sprint completion (Committed vs Actual Story points)
 - Defects returned from QA(%)
 - Escape defects(%)
 - Bug fixing Vs working on feature (% time)



An example - Good Qualitative Agile Metrics: Team Adoption to Agile

Team Metrics

Checklist Items	Sprint N	Sprint N+1
	28.06.16	09.08.16
Core		
Clearly defined PO	Green	Green
Team has a sprint backlog	Yellow	Yellow
Daily Scrum happens	Green	Green
Demo happens after every sprint	Red	Red
Definition of Done available	Yellow	Yellow
Retrospective happens after every sprint	Green	Green
PO has a product backlog (PBL)	Yellow	Red
Have sprint planning meetings	Green	Green
Timeboxed iterations	Yellow	Yellow
Team members sit together	Green	Green
Recommended		
Team has all skills to bring backlog item to Done	Yellow	Yellow
Team members not locked into specific roles	Yellow	Yellow
Iterations doomed to fail are terminated early	Red	Red
PO has product vision that is in sync with PBL	Yellow	Red
PBL and product vision is highly visible	Red	Red
Everyone on the team is participating in estimating	Green	Green
Estimate relative size (points) rather than time	Yellow	Yellow
PO is available when team is estimating	Red	Red
Whole team knows the top 3 impediments	Green	Green
Team has a Scrum master	Red	Yellow
PBL items are broken into task within a sprint	Yellow	Green
Velocity is measured	Red	Yellow
Team has a sprint burndown chart	Green	Red
Daily Scrum is every day, same time & place	Green	Red

Green: It worked for the team.

Orange: Room for improvement.

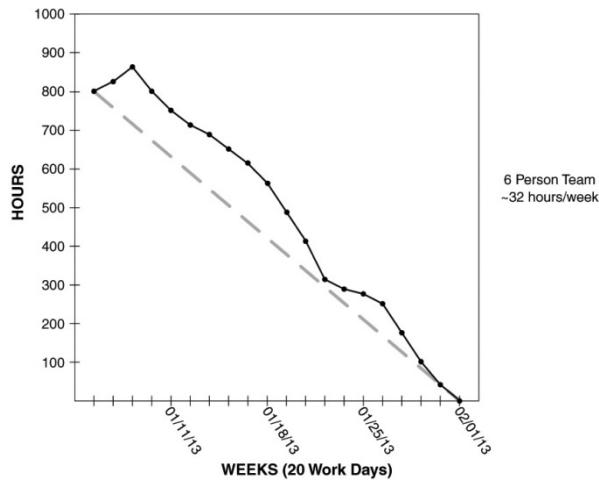
Red: Didn't apply or the practice is failing

<https://www.crisp.se/wp-content/uploads/2012/05/Scrum-checklist.pdf>

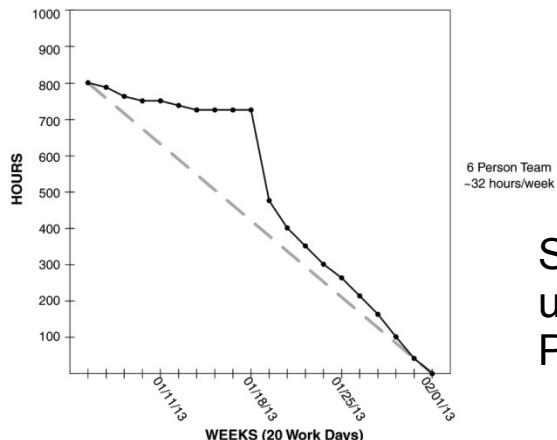
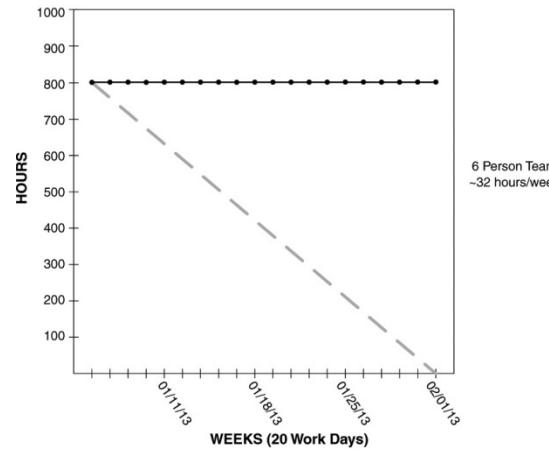
WHAT ARE SOME TRENDS OF BURNDOWN CHARTS AND

WHAT DO THE PATTERNS INDICATE?

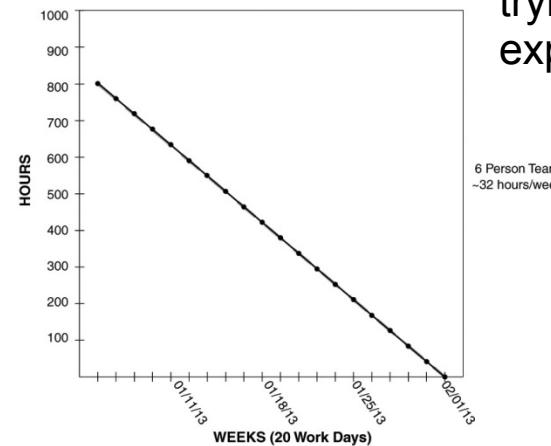
Uptick: New tasks/Stories added. Issue if continues.



Flatline: Multiple reasons. Impediments, Task/Stories added at the same rate as work complete.



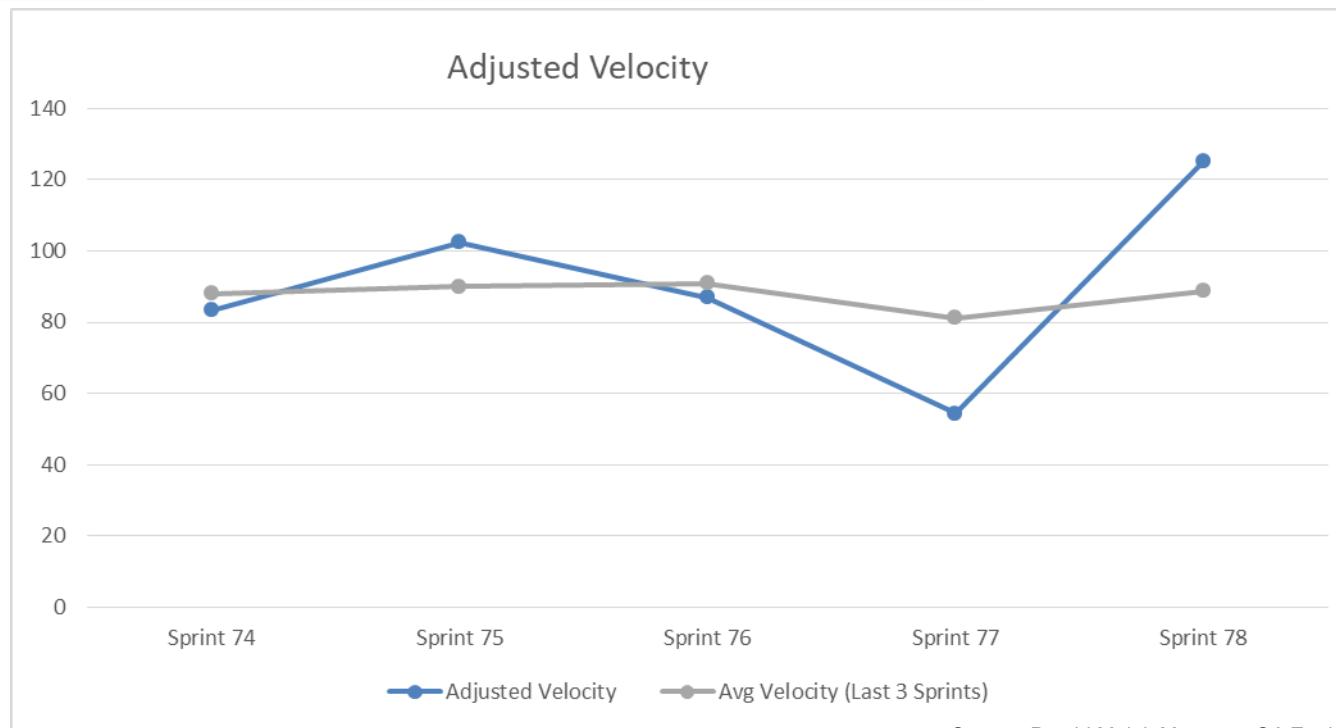
Sharpdrop: Team not updating the chart/
Pointed removed



Perfect line: Team trying to align with expectations

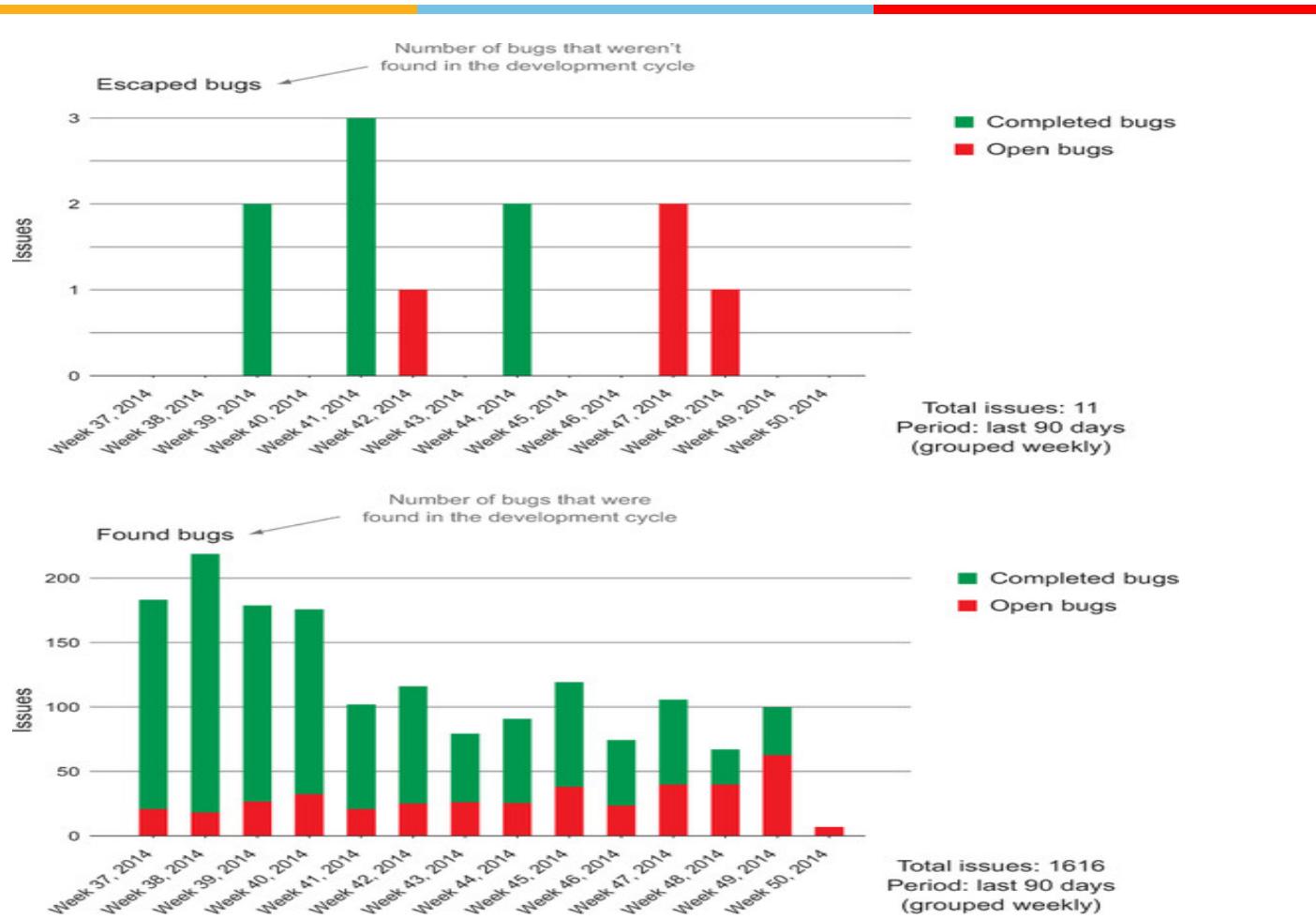
Velocity

Capacity	Sprint 74	Sprint 75	Sprint 76	Sprint 77	Sprint 78
Team Size	8	8	8	8	8
Available Days	80	80	80	80	80
Unavailable Days	10	12	11	5	0
Net Days (Capacity)	70	68	69	75	80
Velocity					
Total Points Completed	73	87	75	51	125
Adjusted Velocity	83	102	87	54	125
Avg Velocity (Last 3 Sprints)	88	90	91	81	89



Source: Prachi Maini, Manager, QA Engineering, Morningstar, Inc.

Bug counts



Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis , Published by Manning Publications, 2015

Summary

- Metrics never convey the whole picture. Management by metrics and dashboards needs to be supplemented with management **by context and conversations**.
- Stop measurements that lead to counterproductive behavior and stop at measurements (i.e., don't continue to targets) that lead to desired behavior.
- Prefer outcome-oriented metrics to activity-oriented ones. Prefer aggregate metrics to fine-grained ones.
- Get comfortable with lagging (or trailing) indicators. When fast feedback is available, lagging indicators are a reliable alternative to speculative leading indicators.

Q1

<https://forms.gle/jWqnUxRgTDg83phf9>

Project Progress

- Burndown Chart
 - Graphical representation of work remaining vs time.
- Committed vs Completed
 - The percentage of points completed by the squad as a percentage of the committed points for the sprint
- Tech Category
 - This helps identify how an agile team is spending its time. The possible values for tech category can be client customization, new product development, operations or maintenance.

What To Watch For

- ✓ The team finishes early sprint after sprint because they are not committing enough points
- ✓ The team is not meeting its commitment because they are overcommitting each sprint.
- ✓ Burndown is steep rather than gradual because work is not broken down into granular units.
- ✓ Scope is often added or changed mid-sprint.
- ✓ Time spent on Feature<>Defects

Q2

<https://forms.gle/7FvNAirb5Qnv8Drf8>

Quality of Code

- First Pass Rate
 - Used for measuring the amount of rework in the process
 - Defined as number of test cases passed on first execution.
$$FPR = \text{Passed} \backslash \text{Total on First Execution}$$
 - For stories that deal with the development of new APIs or Features
 - For stories that deal with addendums to APIs or Features FPR should include regression.

What To Watch For

- Lower first pass rates indicate that Agile tools like desks checks , unit testing are not used sufficiently.
- Lower first pass rate could indicate lack of understanding of requirements.
- Higher first pass rate combined with high defect rate in production could indicate lack of proper QA.

Q3

<https://forms.gle/zkRwgknW7knAtzkp9>

Bug Dashboard

– Net Open Bugs / Created vs Resolved

- This gives a view of the team flow rate. Are we creating more technical debt and defects than what the team can resolve.

– Functional Vs Regression Bugs Trend

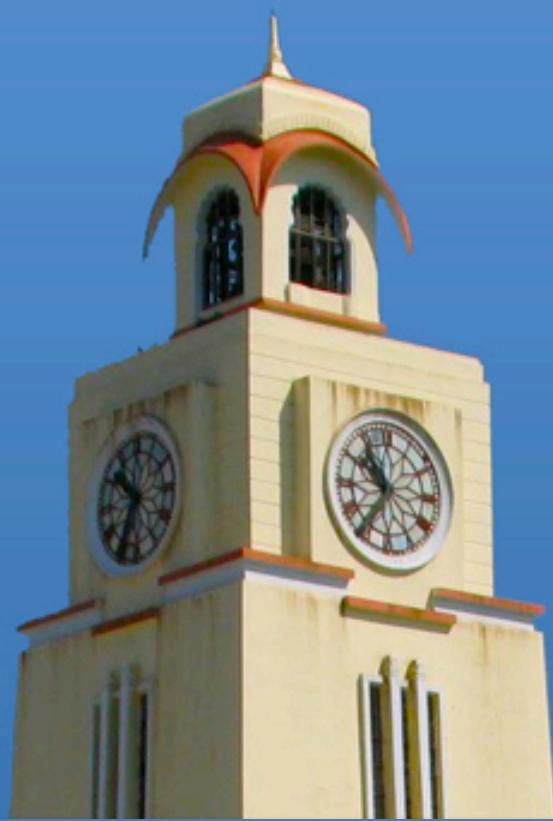
- This helps identify the defects found in new development vs regression.

– Defects Detected in

- This helps identify the environment in which the defect is detected. (QA, Staging, UAT, Production)
- For defects detected in environment higher than QA, an RCA is needed.

What To Watch For

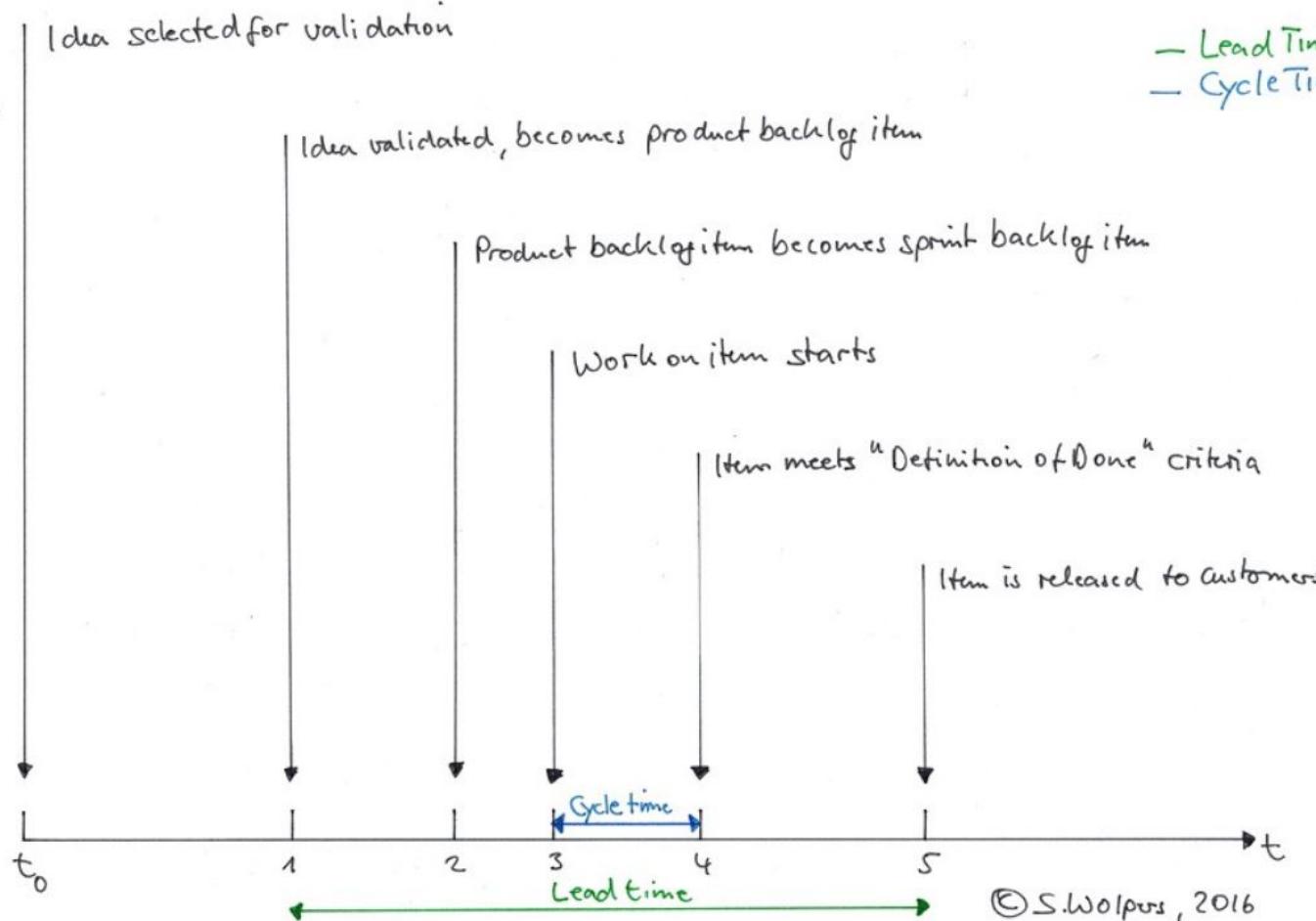
- An increase in regression bug count indicates the impact of code refactoring.
- An increase bug count in non-QA environment due to environment differences requires revisiting the environment strategy.
- An increase bug count in non-QA environment due to QA oversight requires revisiting the testing strategy.



Additional Notes - Measuring Agile Performance

Good Quantitative Agile Metrics: Lead Time and Cycle Time

Agile Metrics : Lead & Cycle Time



Important to measure the things that drive/determine Lead Times. Levers that teams can actively (e.g. determinant metrics like Flow Efficiency).

Most common and meaningful metrics for Team SI



Simple SI metrics	Metric	Comment
Overall SI goal metrics	Lead Time, Cycle Time	Good measures of overall Time to Value
Determinant metrics:		
Best practice and tool use	Speeding tickets (%)	Tickets that have been moved through multiple Statuses (e.g. in Jira) after the event (so there is no real visibility of workflow stages)
Timing accuracy	Total Sprint Completion (%)	Percentage of completed story points for a given sprint(s). The factor takes into account story points added once a sprint has started.

Source: <https://www.infoq.com/articles/metrics-agile-teams/>

Most common and meaningful metrics for Team SI ...



Simple SI metrics	Metric	Comment
Productivity	Flow efficiency (%)	Percentage of time spent active versus inactive within a workflow
	Return rate (%)	Percentage of tickets returned from QA (for whatever reason) during the dev process. This generates Rework.
	% Time bug fixing (The ratio of fixing work to feature work.)	Percentage of time a team spends bug fixing versus feature contribution.
	Number of defects escaping to production.	This is category of fixing the work.

Source: <https://www.infoq.com/articles/metrics-agile-teams/>

Most common and meaningful metrics for Team SI ...



Simple SI metrics	Metric	Comment
Team Wellness	Team Happiness Team Sprint Effectiveness Rating	Self Assessment tests: Individual engineers polled each Sprint/cycle.

Source: <https://www.infoq.com/articles/metrics-agile-teams/>

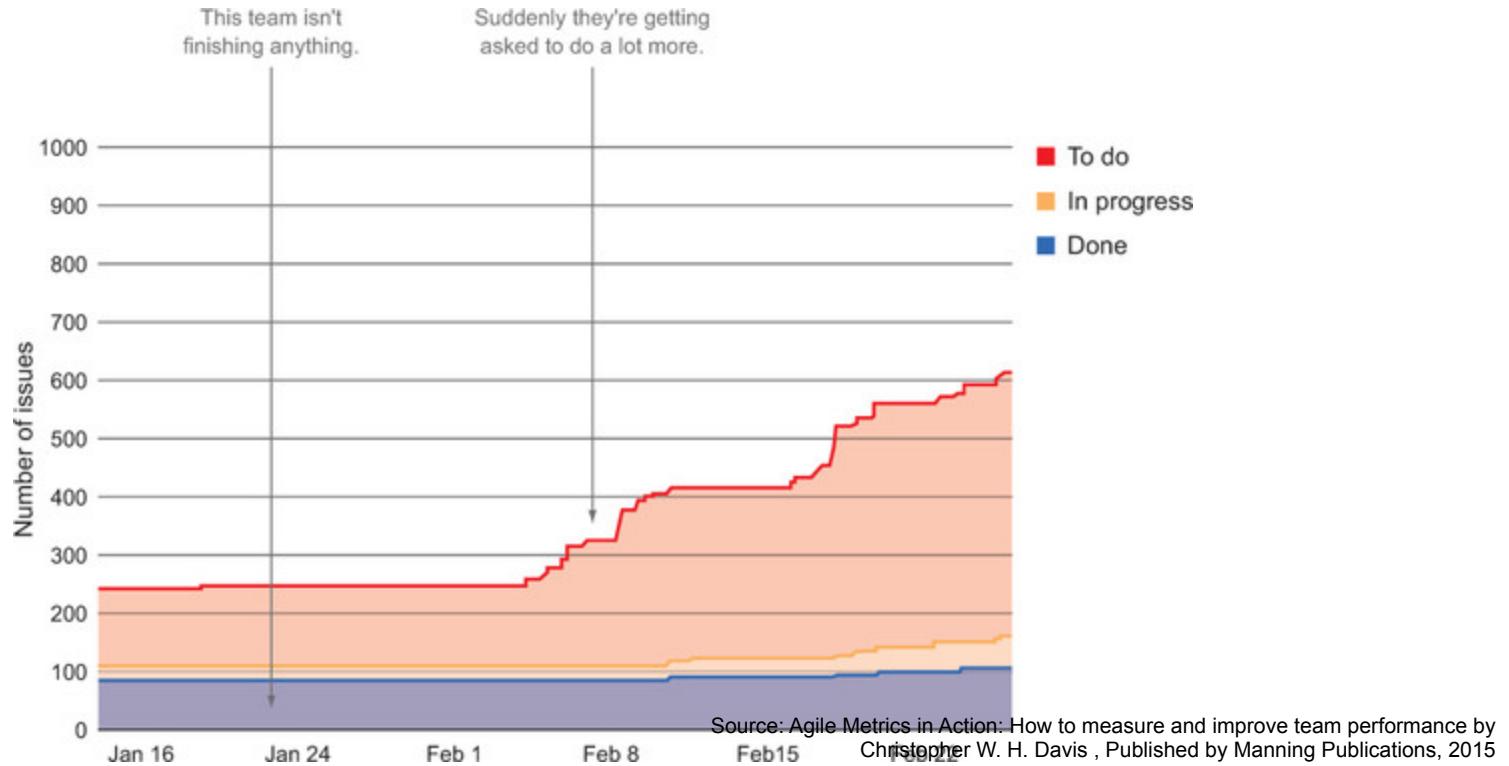
The Importance of Metrics to Agile Teams



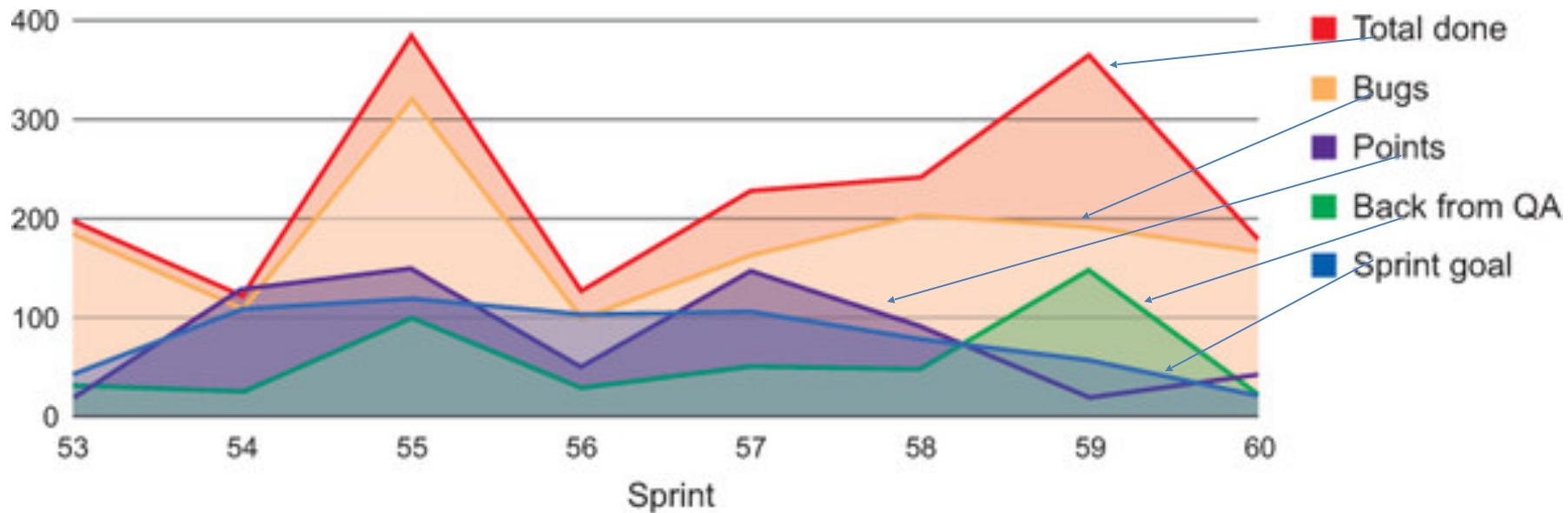
- The philosophy of Continuous Improvement (CI) is central to Agile.
- CI- should not be imposed and driven top-down – instead it should be led by Agile teams themselves, so Self-Improvement (SI) is a more suitable terminology.
- SI is hard requires organization leadership long term support, recognition and suitable framework. Crucially,
 - **A set of meaningful and agreed Agile metrics to track performance improvement over time; and a means to surface these metrics in near real time**, with minimum/no effort involved for the teams themselves.
 - Keep metrics **simple and deterministic (no ambiguity)**.
 - For each of these metrics, it is the *trend* that is important, not an absolute number. The trend will tell you if your attempts at improvement are having an effect.

Cumulative Flow

- An example cumulative flow diagram showing a team getting asked to do a lot more than they have been able to accomplish historically



Example - Combination of data

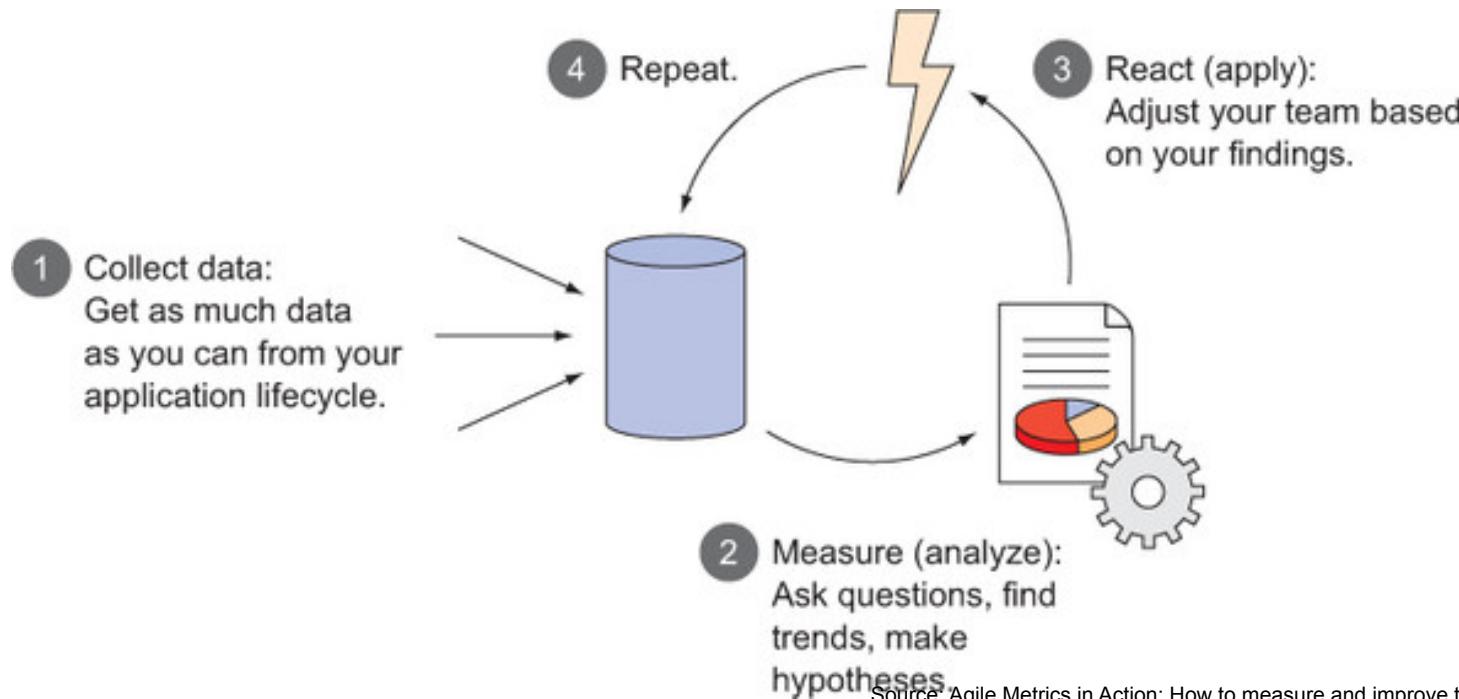


- **Spikes in this data point can indicate potential problems:**
- There's a communication gap somewhere on the team.
- Completion criteria (a.k.a. done) are not defined clearly to everyone on the team.
- Tasks are being rushed, usually due to pressure to hit a release date.

Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis, Published by Manning Publications, 2015

COLLECT, MEASURE, REACT, REPEAT—THE FEEDBACK LOOP

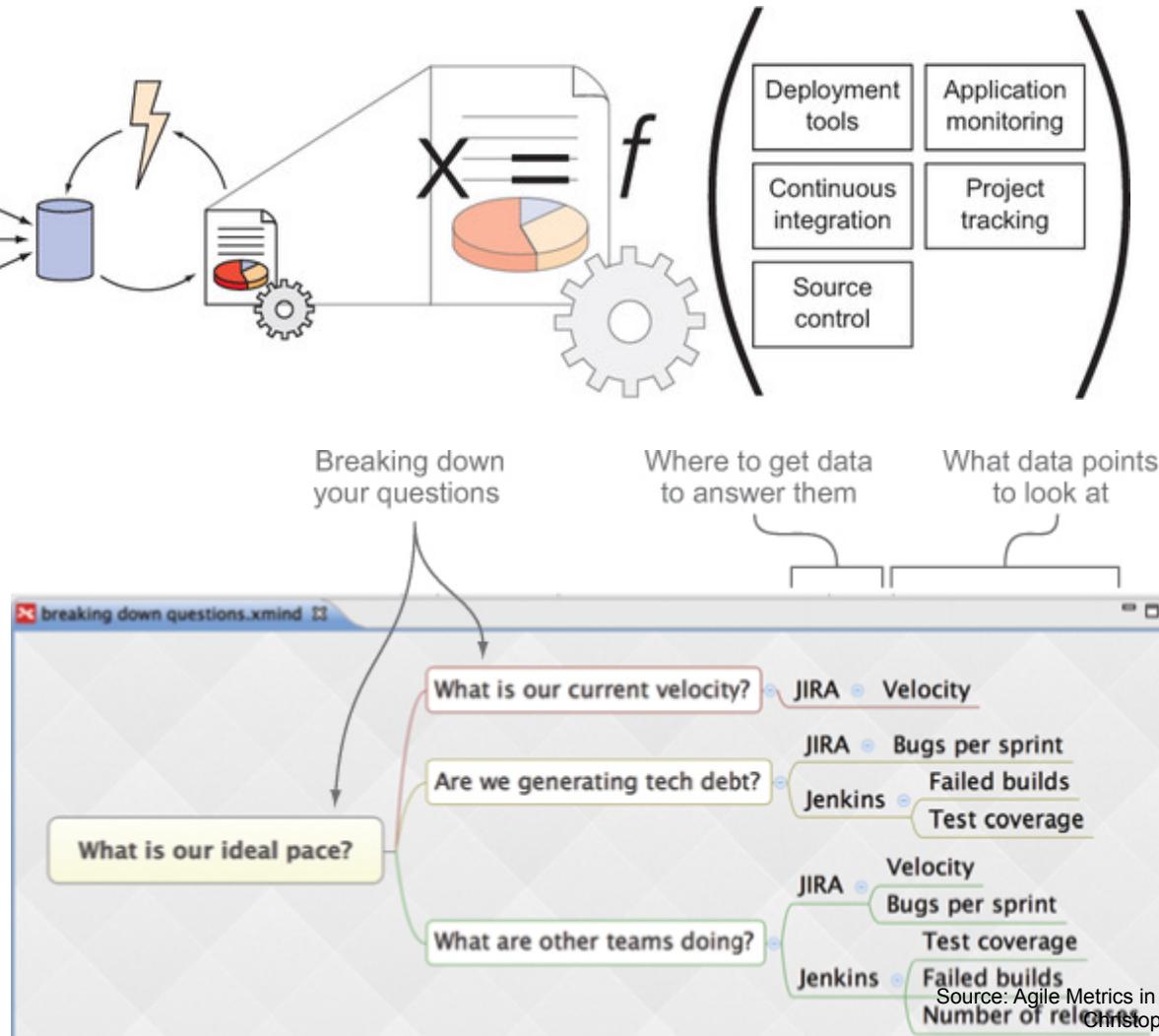
- There isn't a silver-bullet metric that will tell you if your agile teams are performing as well as they can.
- Collecting and analyzing data in the form of metrics is an objective way to learn more about your team and a way to measure any adjustments you decide to make to your team's behavior.



Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis , Published by Manning Publications, 2015

Figuring out what matters

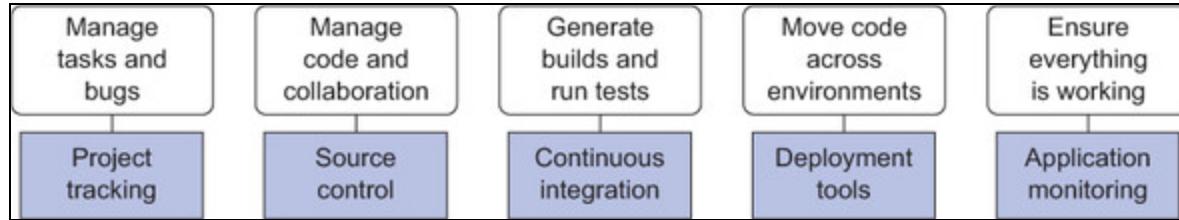
(X is what you want to answer; some combination of your data can get you there.)



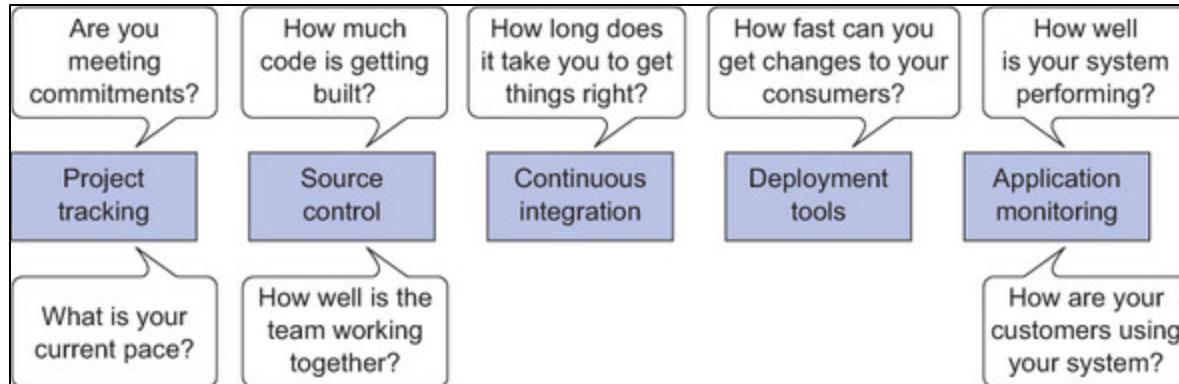
- Mind mapping is a brainstorming technique where you start with an idea and then keep deconstructing it until it's broken down into small elements. XMind (www.xmind.net/),

Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis, Published by Manning Publications, 2015

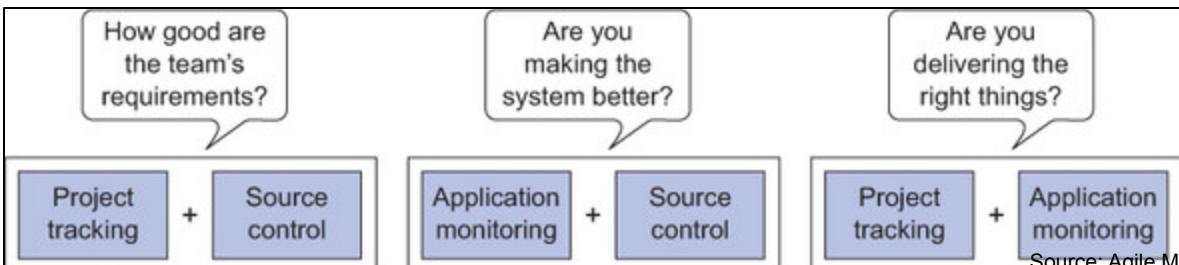
Project performance data



Data is all over the place without a unified view



Questions you can answer with data from systems in your SDLC.



Adding data together to answer high-level questions

Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis, Published by Manning Publications, 2015



Thank you



BITS Pilani
Pilani Campus

BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



Module-10 – Managing Quality and Risks in Agile Project

Key Differences between Agile and Traditional Quality Management

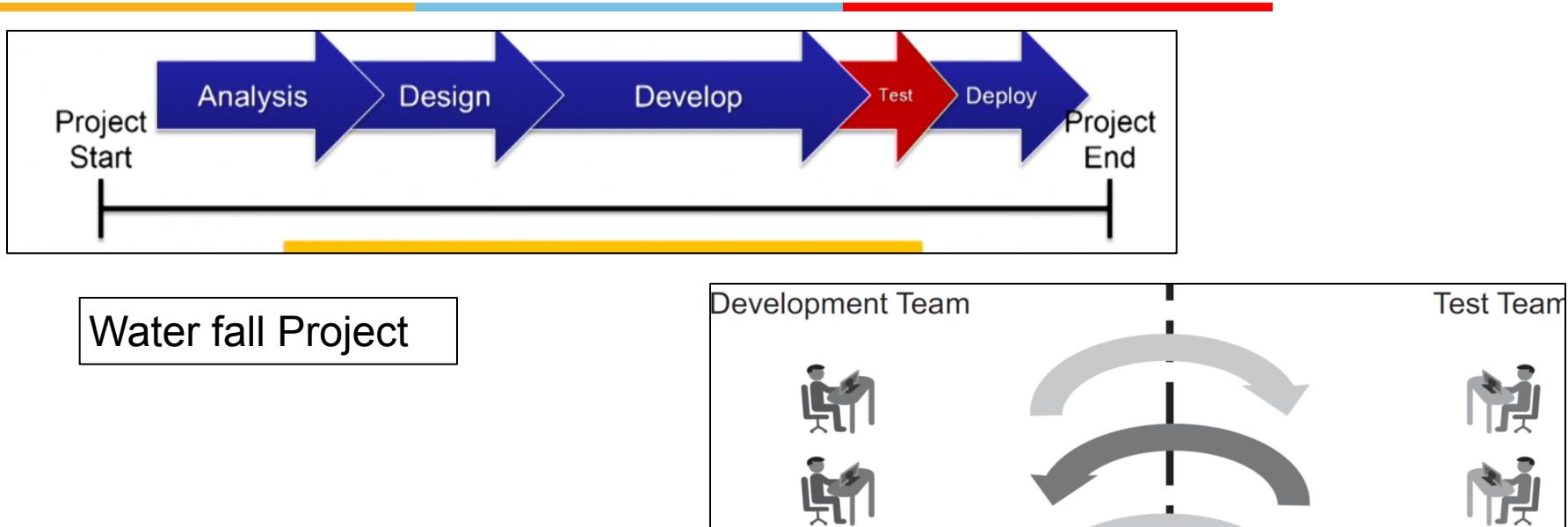


- Integration of Testing with Development
 - Concurrent vs Sequential
- Testing Approach
 - More reactive vs More Proactive
- Responsibility of Quality
 - Overall Team <> QA Team
- Regression testing
 - Frequent (Code Changes), At end after Code stabilizes

Agile Development and Testing Practices

- Agile Development Practices
 - Continuous Integration
 - Code Refactoring
 - TDD
 - Pair Programming
- Agile Testing Practices
 - Repeatable Test Automation, Acceptance Drive test development
 - Exploratory testing, Concurrent Testing, Value & Risk based testing

Agile Approach to Quality

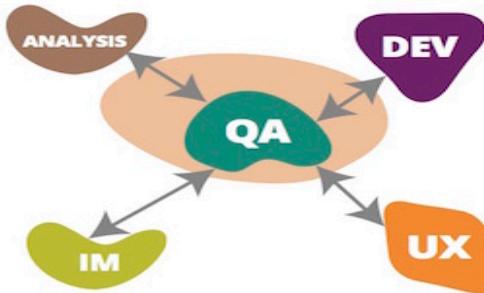
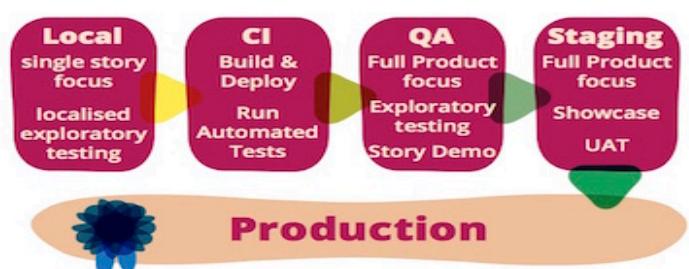
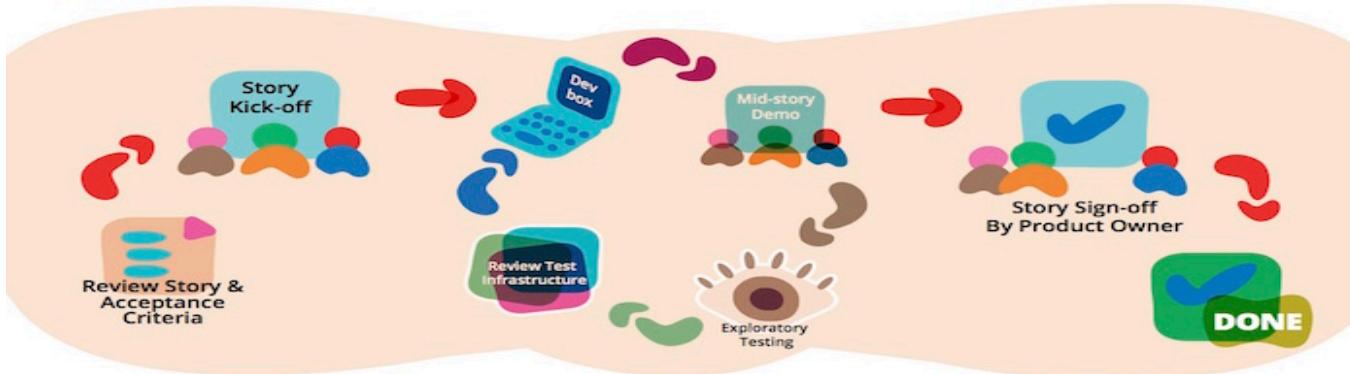
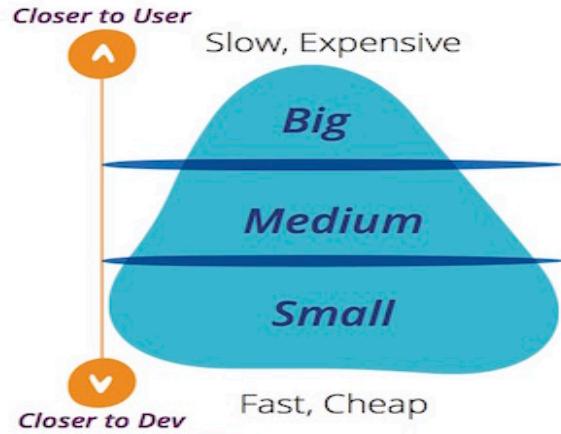


- Agile approach to building quality product
 - Early delivery & Testing, Sprint Review, Customer feedback
 - Customer collaboration
 - Good Technical practices
 - Whole team participation to Quality
 - Test Automation

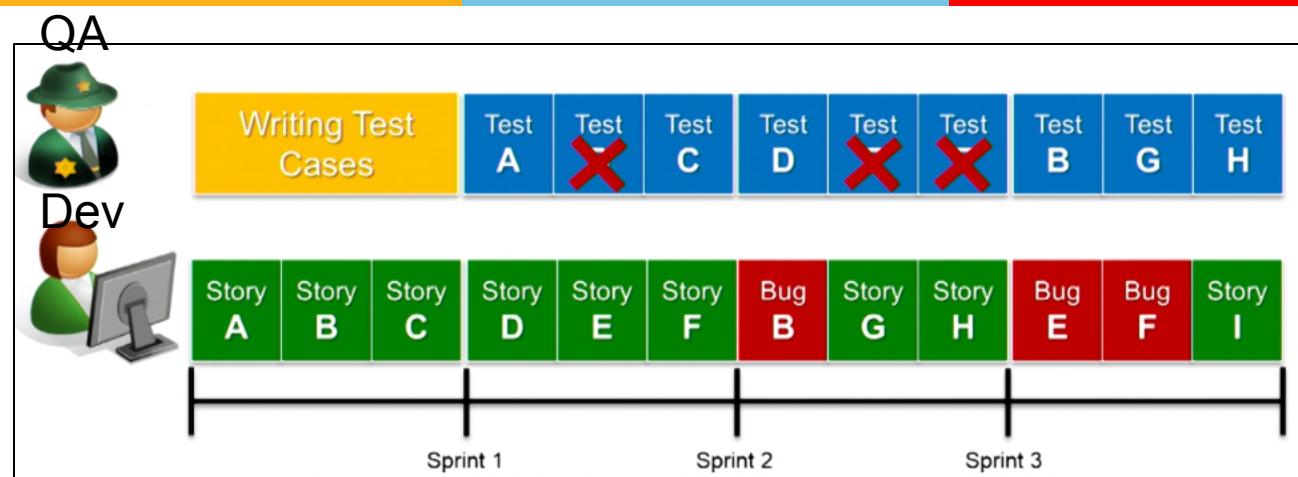
Ref: <https://www.vivifyscrum.com/insights/qa-agile-project-management>

QA Role in Agile Project

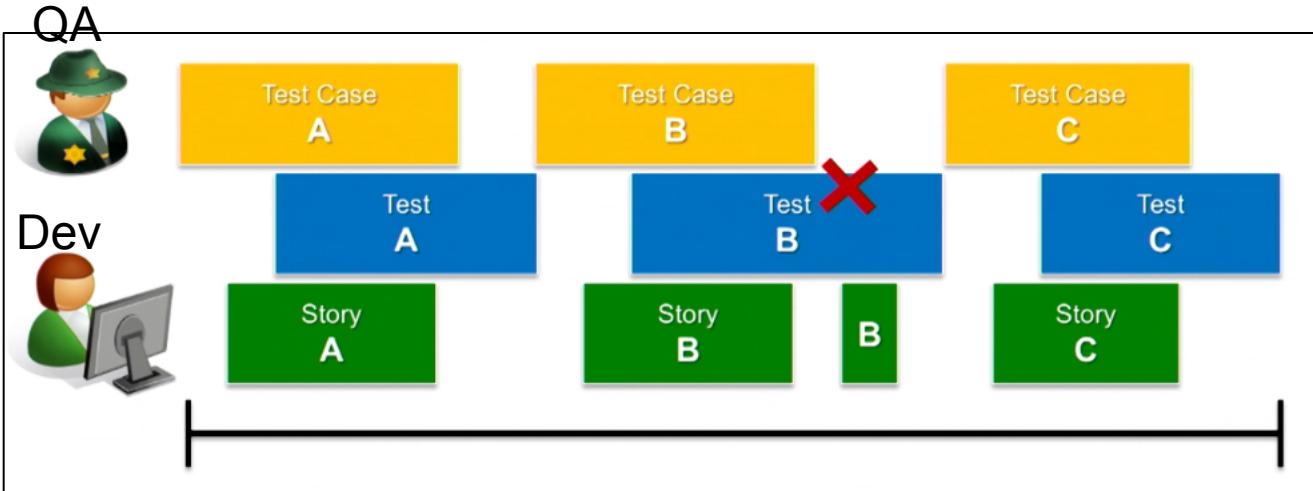
Are We **Building** the Correct Product?



Testing within Sprint



- QA lagging behind in Testing
 - Bugs Snowballing effect



- Collaboratively testing with Dev.
 - Fully tested Software
 - Minimal Hands-off.

Dealing with bugs:
•Critical, Non-Critical,
Enhancements

QA good Practices

QA Best Practices

Hire good quality QA **ENGINEERS**.

QA and dev sit together.

QA is involved in analysis and design.

Test as you go.

Testing is part of your definition of done.

Limit your work in progress.

Everyone can help test.

Frequent, incremental releases for feedback.

Don't Accumulate defects
Set bug queue limits.

Process Quality
Product Quality

Quizzes

- Q1, Q2, Q3



Risk Management in Agile

Risk management in Agile

- Risks are uncertain event(s)
 - May affect your project positively or negatively
 - Positive Risk: A technology currently being developed that will save you time if released.
 - Negative Risk: Unavailability of Skilled resources.
- Agile methods have a built-in risk mitigation component.
 - Identify, Assess, Prioritize, Mitigate, Communicate
 - Daily meeting, Sprint review, Story Grooming, Retrospective

Mitigation Strategies for positive risks or opportunities

Exploit:

- This strategy ensures that opportunity definitely happens. For example, assigning the most talented resource to your project to reduce the duration of the project.

Share:

- Allocating part of the ownership of opportunity to a third party to ensure that the opportunity definitely happens and risk is reduced. For example, going for a joint venture.

Enhance:

- This strategy increases the positive impact of the opportunity. For example, adding more buffer resources to an activity to finish it early.

Mitigation Strategies for Negative Risks (Threats)

- **Avoidance:**
 - Eliminating a specific threat by eliminating the cause.
 - Use different set of tools/Libraries
- **Transference:**
 - Contracting, insurance warranties, guarantees, outsourcing the work are the examples of risk transfer.
- **Mitigation:**
 - Reducing risk probability and impact
 - Insufficient server resource: Increase CPU/Memory to reduce server crash
- **Accept:**
 - Accept the risk. Do not do anything.
 - Taking a risky project with potential for future benefits.

Communicating - Risk Register – An Example

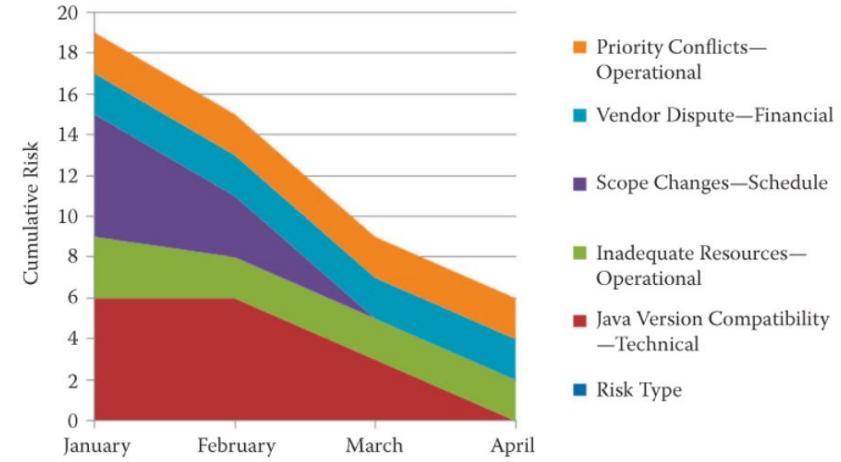
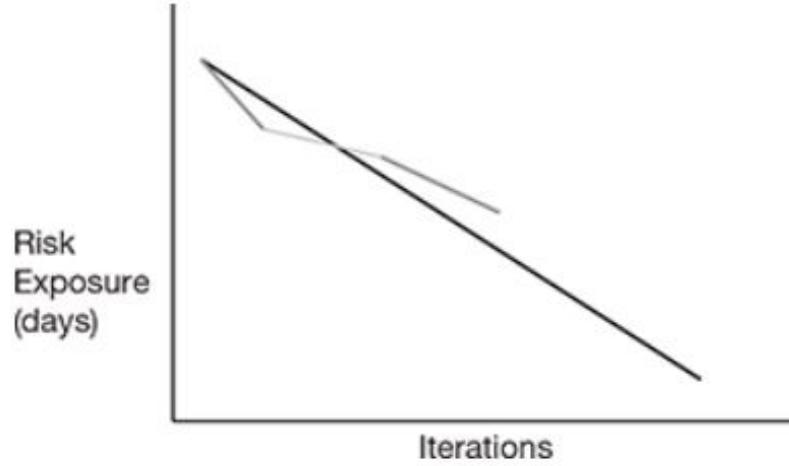


Risk Description	Probability of Occurrence	Impact		Risk Exposure (Days)
		Loss Size (Days)	Impact	
Insufficient QA time to validate on all browsers and OS types.	45%	6	2.7	
Lack of verifiable sample data may affect the ability of the primary external stakeholder to validate end product.	35%	18	6.3	
Inadequate staff available from external stakeholders until very late in cycle.	25%	7	1.8	
Following end-user testing, more effort on the user guide may be necessary.	25%	18	4.5	
Backup and restore requires 3rd-party solutions (not evaluated yet).	20%	12	2.4	
Insufficient time for external stakeholders to submit feedback on layout and composition of reports.	10%	5	0.5	
			Total Risk Exposure	18.2

- Risk Impact : Measure the negative impact of the risk.
- Risk Impact Objectives: Cost, Time, Quality, Scope
- There could be many other columns in the risk register such as Date, Owner, Status, Priority etc..
- Risk Exposure: Probability * Impact

Source: <https://www.castsoftware.com/research-labs/software-development-risk-management-plan-with-examples>

Risk burndown chart



- We can draw risk burn-down chart (graph) which contains iterative cycle number vs risk exposure days. Risks are monitored by the use of information radiators, daily stand-up meetings, and iterative cycle reviews and retrospectives. Y-axis of the risk burn-down chart contains risk exposure days. The X-axis of the risk burn-down chart contains the iterative number.

Use Risk Management to Make Solid Commitments to executives



- Use Risk Multiplier in your estimation forecast
- Account for common risks - Turnover, Changing Requirements, Work disruption etc..

<u>Risk Multiplier</u>			
<u>Chance</u>	<u>Rigorous Process</u>	<u>Risky Process</u>	<u>Description</u>
10%	1	1	Ignore--almost impossible
50%	1.4	2	Stretch goal--50/50 chance
90%	1.8	4	Commitment-- virtually certain

**these multipliers are estimates gleaned from DeMarco & Lister's RISKOLOGY simulator and Todd Little's detailed analysis of hundreds of projects. The most accurate approach is to calculate your own risk multipliers from past project history.*

Source: <https://www.jamesshore.com/v2/blog/2008/use-risk-management-to-make-solid-commitments>

Using risk multiplier in your estimation



- For example, if you are using a **rigorous approach**, your release is 12 iterations away, your velocity is 14 points, and your **risk exposure is one iteration**.

You would calculate the range of possibilities as:

- Iteration remaining = $12 - 1 = 11$
- Points remaining = $11 \times 14 = 154$ points
 - Risk Multiplier* = **1,1.4,1.8** for Rigorous Approach, Risky Approach = **1,2,4**
 - 10 % chance: $154/1 = 154$ points
 - 50 % chance: $154/1.4 = 110$ points
 - 90 % chance: $154/1.8 = 86$ points
 - In other words, when it is time to release, you are 90% likely to have finished 86 more points of work, 50% likely to have finished 110 more points, and only 10% likely to have finished 154 more points.

An Example

Suppose, estimated number of sprints is 10 for a release

Product Backlog at end of Sprint 5 - F1,F2,F3,F4,F5,F6

Assume each feature size = 20, Velocity = 20

Total size of the remaining features = $6 \times 20 = 120$ points

Sprint Remaining = 5, Risk Multiplier = 1,1,4,1.8

6th Sprint Commitments:

10% Chance : Sprint remaining * Velocity - $5 \times 20 / 1 = 100$ points

– 10% chance of delivering F1,F2,F3,F4,F5 (100 points)

50% Chance : $5 \times 20 / 1.4 = 71.4$ points

– 50% chance of delivering F1,F2,F3 (60 points), Stretch = F4

90% Chance : $5 \times 20 / 1.8 = 55.5$ points

– 90% chance of delivering F1,F2 (40 points), Stretch = F3

- Repeat this process after completing Sprint6

Summary

- **Agile Quality Management**
 - Adaptive planning, Frequent reviews
 - Concurrent Regression testing
 - Test Automation
 - Proactive testing, Defect handling
 - QA Ownership, QA role is much larger compared waterfall method
- **Agile Risk Management**
 - Continuous risk assessment: Through daily standup meetings, Scrum planning meeting, release planning meeting, etc.
 - Agile projects have its own inbuilt risk handling mechanism, well aligned with quick risk identification, Ownership, and controlling mechanism.
 - The iterative nature of Agile projects identifies risks earlier in the project execution and also the risk process repeats for each and every iteration, thereby managing it in a better way.

Quizzes

-
- Q4,Q5,Q6,Q7,Q8



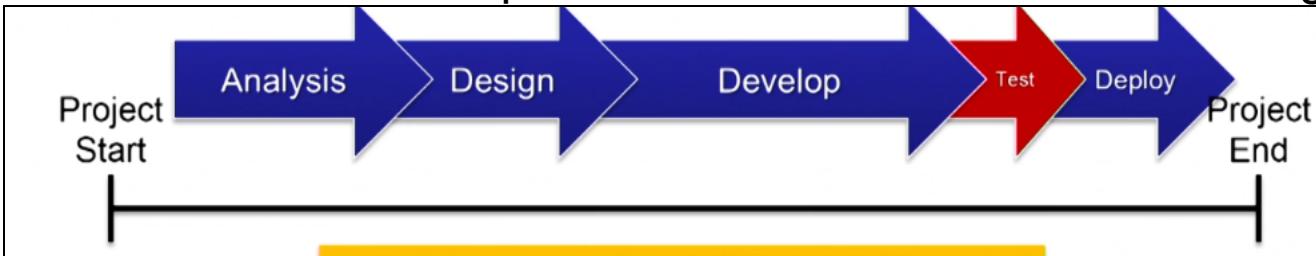
Additional notes – Quality & Risk management

Issues with Traditional Approaches to Quality Management

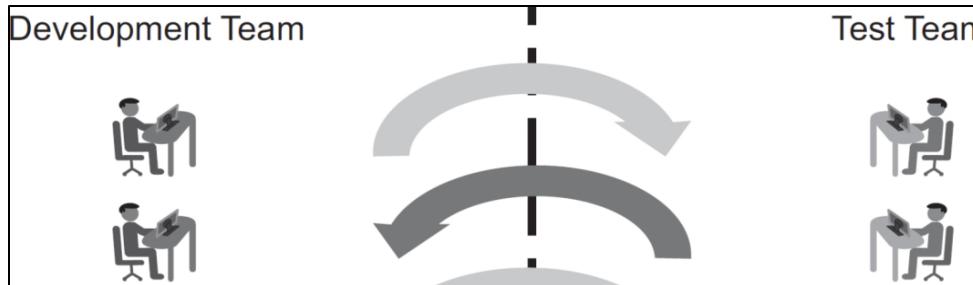
Traditional Approach to QA

1. QA Audit for compliance Review

2. Most QA Testing happens at the end



2. Transfer of responsibility from developer to tester and vice versa



- In traditional sequential, All of this 'back and forth' activity can easily create division within software development and QA teams if not managed correctly.

Ref: <https://www.vivifyscrum.com/insights/qa-agile-project-management>

Agile Approach to Quality Management



- Agile Manifesto & Agile Principles – Focus on **Building Quality In**
 - **Early delivery & Testing** of working software to customers as quickly as possible.
 - **Customers can also provide early feedback** on features, elements in the product which they like/dislike, and aspects of the solution that they wish to remove or modify.
 - **Agile values promotes collaboration with customer**, Team works with business team on daily basis, Simplicity, Technical excellence, Daily meetings, iteration feedback
 - **Good Technical practices** improves Quality: (Not specific to Agile)
 - TDD, CI, Collective code ownership, Pair programming, Refactoring, exploratory testing, reviews.
 - **Whole team approach** to Quality
 - In this way, Agile development can improve customer satisfaction and produce solutions that more closely meet customer needs.

Agile Approach to Quality Management



- The hand-offs between programmers and testers (if they exist at all) will be so small as not to be noticeable.
 - Team work, Doing a little of everything (designing, coding, testing, and so on) all the time helps teams work together.
 - Tester creates automated tests and the programmer programs. When both are done the results are integrated. Hands-off is insignificant.
- There should be as much test activity on the first day of a sprint as on the last day
 - No distinct analysis, design, coding, or testing phases within a sprint. Testers (and programmers and other specialists) are as busy on the first day of a sprint as they are on the last.
 - For example, testers may be specifying test cases and preparing test data on the first day and then executing automated tests on the last, but they are equally busy throughout.

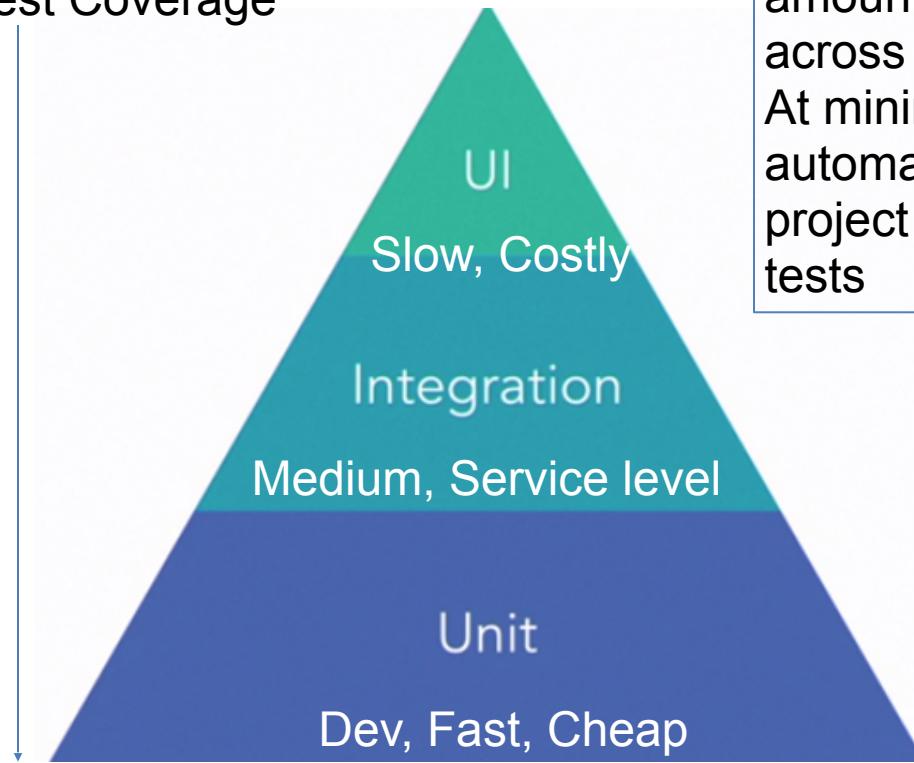
Agile Approach to Quality Management

.... Automate Tests at Different Levels



• Automation Pyramid

Test Coverage



Visual representation of the recommended amount of test coverage that should exist across each type of test.

At minimum we should have three type of automated tests. Depending upon the project type we can have more type of tests

Unit Test: Isolated tests , test functions, Fast, Need greater number of tests.

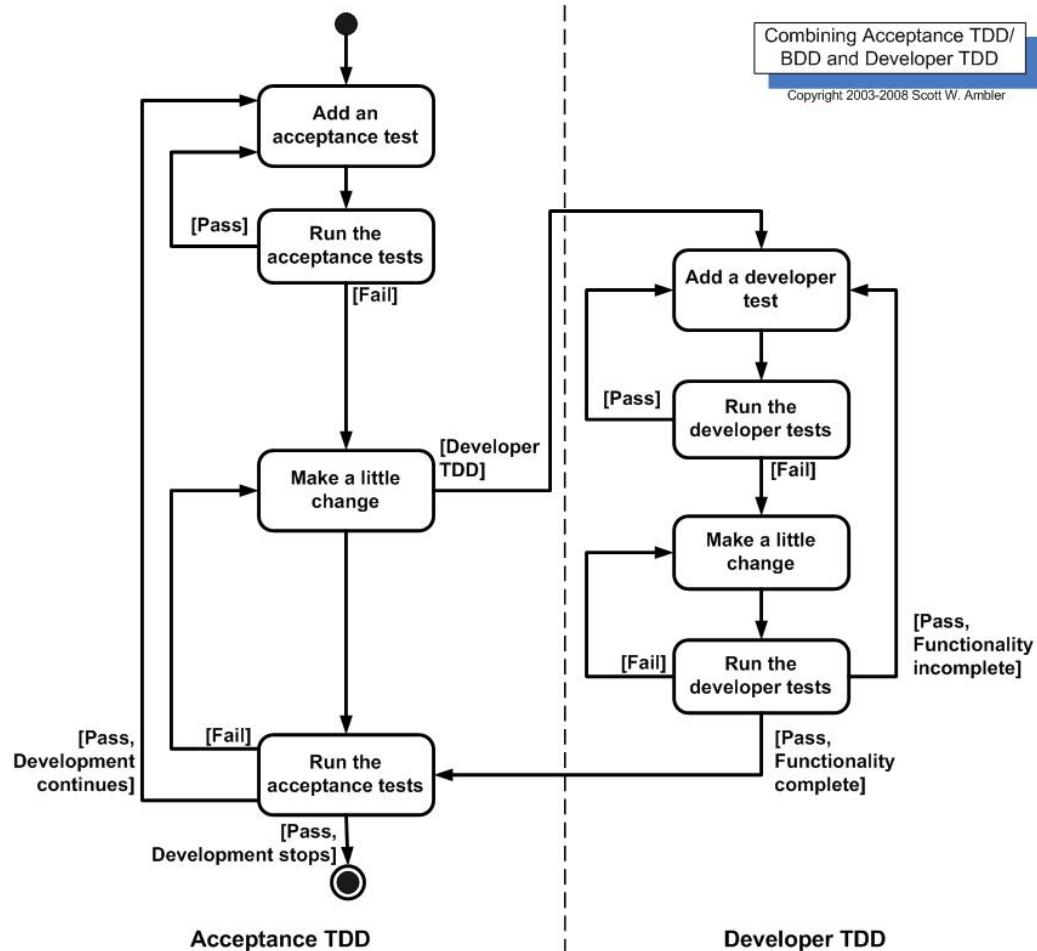
Integration tests: Slower, tests interfaces, databases, file system, other applications.

UI Tests: Tests end-end work flow. Much slower.

The Role of Manual Testing

- It is impossible to fully automate all tests for all environments. Further, some tests are prohibitively expensive to automate. Many tests that we cannot or choose not to automate involve hardware or integration to external systems.
- Exploratory testing
 - Free form manual testing, Quick Test planning, test design test execution sessions.
 - Can identify missing test cases
 - Exploratory testing can uncover ideas that are missing from the user story as initially understood.
- Automate within sprint (Automation not optional)
- Pay off Technical debt

Do Acceptance Test Driven Development



Analogous to test-driven development, Acceptance Test Driven Development (ATDD) involves team members with different perspectives (customer, development, testing) collaborating to **write acceptance tests in advance of implementing the corresponding functionality**.

What is Risk?

- A risk is considered to be an uncertain event(s) that has the potential to contribute to the success or failure of a project.
- Positive risks are defined as opportunities and threats are risks that can affect the project in a negative way.
 - Examples:
 - Positive Risk: A technology currently being developed that will save you time if released.
 - Negative Risk: Unavailability of Skilled resources.
- Risk Management
 - Identify, Assess, Prioritize, Mitigate, Communicate
- Agile methods have a built-in risk mitigation component.
- Risk Burndown Chart – For communicating the risks

Mitigating Risks with Agile Methods



- The flexibility of agile methods automatically reduces risk in the business environment.
 - Risk is mitigated because agile methods are flexible with adding or changing user requirements at any time in the project.
 - Missing or forgotten requirements can be included as soon as they are identified.
 - This results in low costs associated with managing this category of risks.
- Regular feedback reduces risk-related expectations.
 - As a result of the iterative nature of agile methods, there is adequate time to get feedback and establish expectations during the life cycle of the project.
 - Stakeholders and the agile team can avoid surprises because of requirements that have been communicated inadequately.

Mitigating Risks with Agile Methods

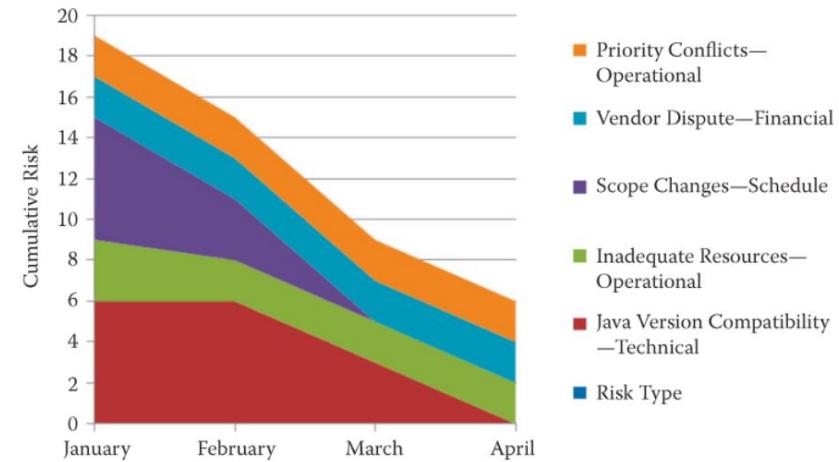


- Agile team ownership supports reduced estimation risk.
 - When the agile team takes responsibility for estimates of backlog items, this leads to increased accuracy of the estimates that they provide which in turn results in the timely delivery of the product.
- Transparency is a risk reducer of undetected risk.
 - As a result of transparency, risks are always detected and addressed as early as possible.
 - This leads to better risk management and mitigation. During daily meetings, obstacles are communicated on a regular basis.
- Iterative delivery causes a reduction in investment-related risk.
 - As value is being continuously delivered through the iterations, investment risk is automatically reduced for the end customer.

Risk Register- Another example

Risk	Type	Impact (0– 3)	Probability (0– 3)	Severity = Impact × Probability
------	------	---------------	--------------------	---------------------------------

1. Java version compatibility	Technical	3	2	6
2. Inadequate resources	Operational	3	2	6
3. Scope changes	Schedule	3	2	6
4. Vendor dispute	Financial	2	1	2
5. Priority conflicts	Operational	2	1	2





BITS Pilani
Pilani Campus

BITS Pilani presentation

K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in

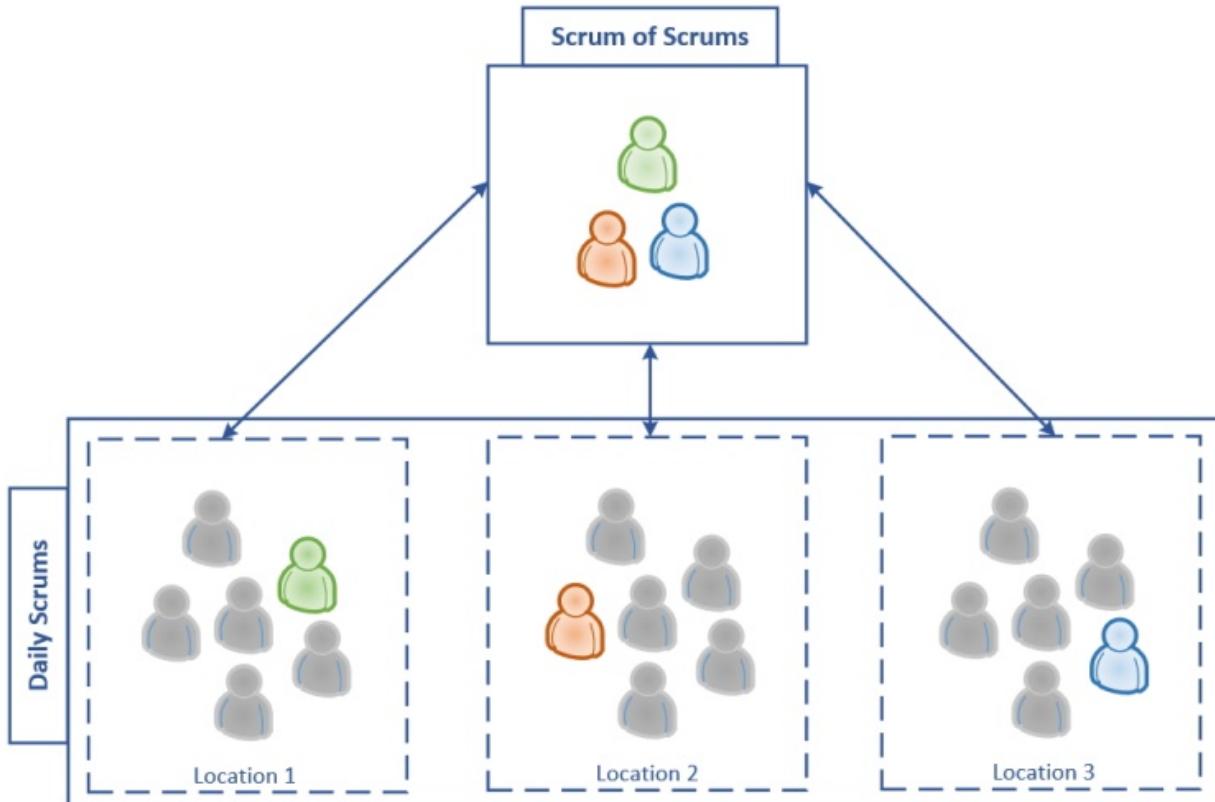


Module 11&12 - Agile Myths and Pitfalls, Ensuring Agile Success

Agile Myths/ Misconceptions

- **Myth #1) Agile is a Methodology**
 - Agile is a mindset, a philosophy that describes a set of values and principles coined in the Agile Manifesto. Not a step by step process or methodology.
- **Myth#2) Agile =Scrum**
 - Not true, Kanban, XP, Lean, Crystal ...
- **Myth#4) Agile is Anti Documentation**
 - Not true, Minimum documents needs to product for support and maintenance
- **Myth#4) Agile Means no planning**
 - Not true. Daily, Iteration, Release
- **Myth#5 Work must fit into sprint**
 - Not true, Kanban dose not require sprinting

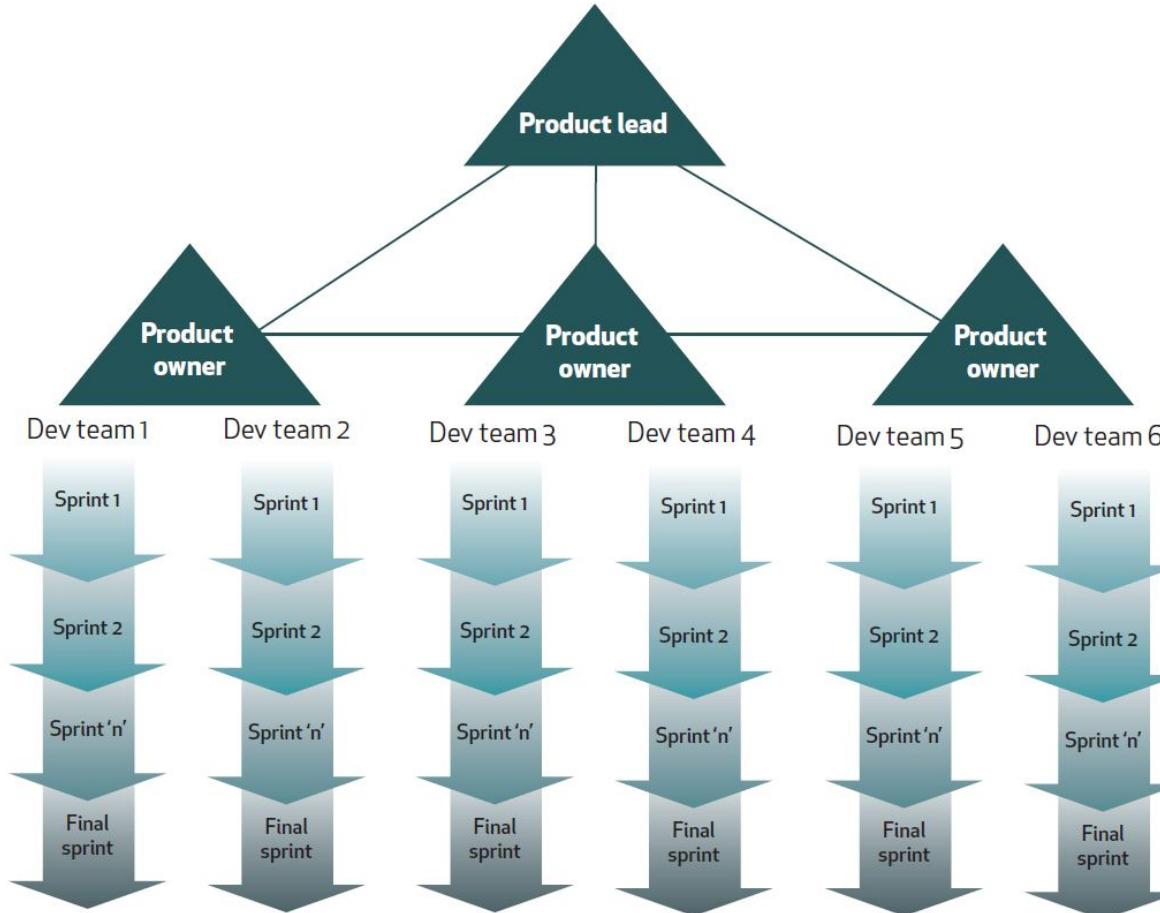
Scaling Scrum: Scrum of Scrums



- What work has your team completed since the last Scrum of Scrums?
- What work is your team planning to do before the next Scrum of Scrums?
- What current or predicted blockers does your team have?
- What blockers could you cause another Scrum team?

Source: Product Management Journal Vol.7.

Scaling Scrum/ Scrum of Scrum/ SAFe



Source: Product Management Journal Vol.7.

Distributed Agile Models

Location-Independent Agile Model	Local	Minimally Distributed	Significantly Distributed	Fully Distributed
	Model 1	Model 2	Model 3	Model 4
Business Knowledge at Distributed Location	Not Applicable	Medium to High	High	High
Nature/Criticality of Effort	Regulatory/ Urgent/ Volatile	All Except Regulatory/ Urgent/Volatile	All Except Regulatory/ Urgent/Volatile	All Except Regulatory/ Urgent/Volatile
Organization's Experience in Agile Way of Working	Nil to Low	Low to Medium	Medium to High	High

Source: <https://www.tcs.com/perspectives/articles/how-to-make-location-independent-agile-work>

Key issues with Distributed Model



- Communication
- Team Issues & Trust
- Release planning & Execution
- Lack of visibility

Distributed Agile team best practices



Best practices reinforce each other to mitigate risks

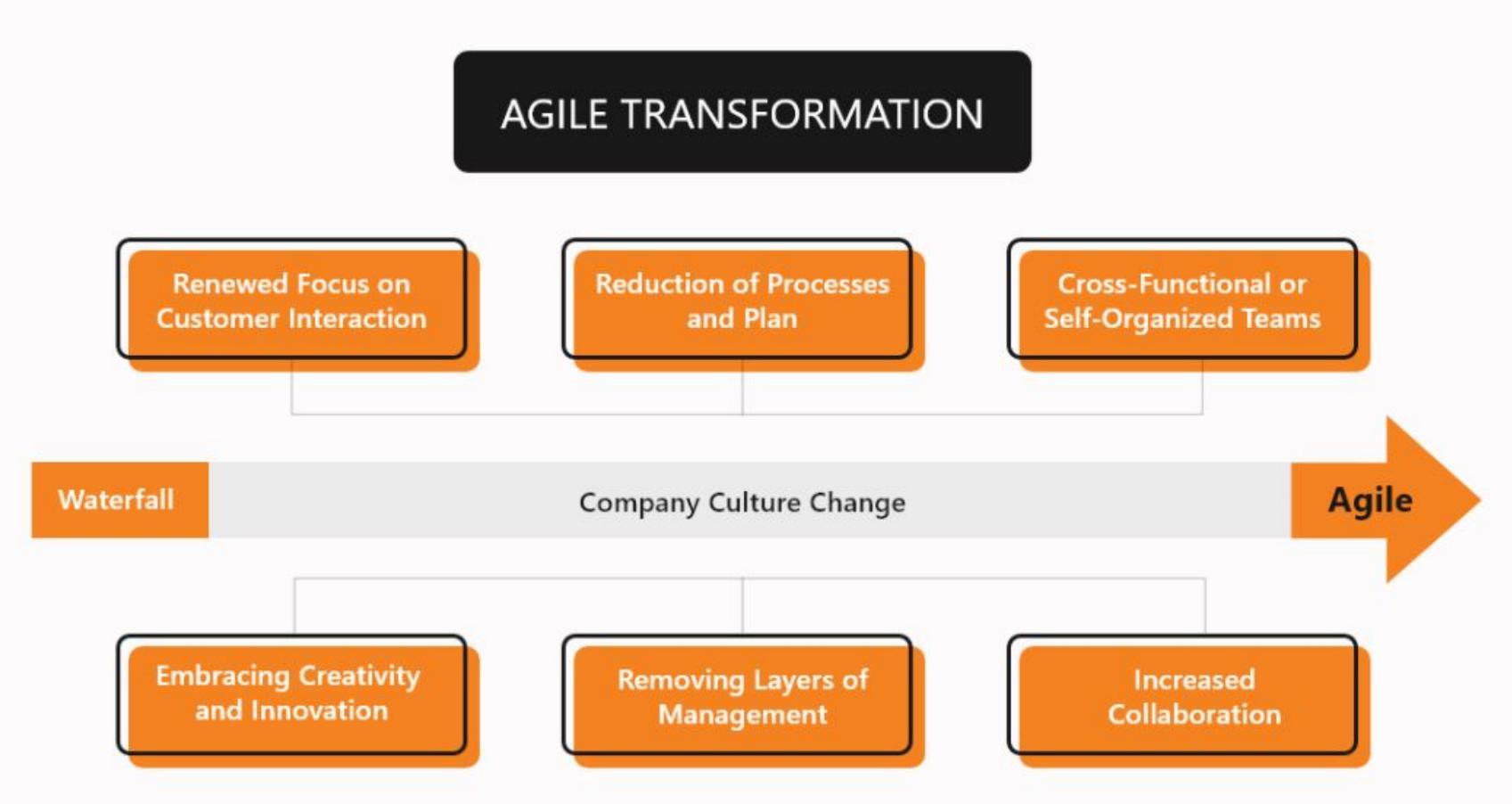
Best Practices	Key Risk Areas			
	Communications	Team & Trust	Release Plan	Visibility
Redundant Roles	X	X	X	X
Customer Proxy	X	X	X	X
Daily Hand-Off	X	X		X
Communication Infrastructure	X	X		X
Reinforce Agile Principles		X		
Cross Pollination	X	X		X
Co-Located Inception and Release Planning	X	X	X	X
Story Tracking Tool / Virtual Card Wall	X	X	X	X
Agile Tracking and Metrics	X	X	X	X

© ThoughtWorks 2008



Source: https://www.slideshare.net/ThoughtWorks/simons-rickmeierdistributedagilev3?from_action=save

Agile Transformation



Source: <https://customerthink.com/how-agile-transformation-is-different-from-digital-transformation/>



Additional Notes

Distributed Agile/Location Independent Agile teams



- Covid-19 sudden disruption and impact
- <https://www.mckinsey.com/business-functions/organization/our-insights/revisiting-agile-teams-after-an-abrupt-shift-to-remote>
- TCS Perspectives:
 - First: Assess the Organization:
 - What is the level of business expertise and other skills required, and to what extent do they exist at a specific location?
 - If a location lacks business expertise, it will require more of it to be able to support agile teams there.
 - How urgent and volatile is the work?
 - Location-independent agile teams should focus on work that is neither urgent nor volatile. If the work is both, if it has non-negotiable constraints (such as overnight fixes, intra-day scope changes, or regulatory requirements), or if there is a need for constant access to the project owner, it's best to work with teams in the same location, if at all possible.

Source: <https://www.tcs.com/perspectives/articles/how-to-make-location-independent-agile-work>

Distributed Agile/Location Independent Agile teams ...

- How mature is the organization?
 - When teams are relatively new to agile approaches, team members should be co-located. Having a common understanding of agile culture, especially among the leadership, indicates the organization can succeed with location-independent teams.

Source: <https://www.tcs.com/perspectives/articles/how-to-make-location-independent-agile-work>

Distributed Agile Models

Location-Independent Agile Model

	Model 1	Model 2	Model 3	Model 4
Business Knowledge at Distributed Location	Not Applicable	Medium to High	High	High
Nature/Criticality of Effort	Regulatory/ Urgent/ Volatile	All Except Regulatory/ Urgent/Volatile	All Except Regulatory/ Urgent/Volatile	All Except Regulatory/ Urgent/Volatile
Organization's Experience in Agile Way of Working	Nil to Low	Low to Medium	Medium to High	High

Local - Model-1 ; This model is best when the teams are new to the business area, when continuous access to a product owner is paramount, or when regulatory concerns require the project to be executed in a specific geography.

Minimally Distributed (Model 2): The product owner and a few members of a project or program are located together as one team, while the rest of them work together as another inter-related team in a different place. This model requires the teams to understand the underlying business processes for their product.

Significantly Distributed (Model 3): Team has shared understanding of the business processes of a project or program are well positioned to adopt significantly distributed agile work processes.

Fully Distributed (Model 4): The product owner may be at any site, while the rest of the project or program team members are grouped in agile teams across distributed locations. These teams each include product specialists with sufficient business knowledge to drive day-to- day decisions within a framework defined by the product owner.

Source: <https://www.tcs.com/perspectives/articles/how-to-make-location-independent-agile-work>

Agile Transformation

- An agile transformation is an act of transforming an entire organization into an elegant and thrive in a collaborative, flexible, self-organizing, and fast-changing environment based on Agile principles.
- Agile principles can be taught throughout any organization to develop teams to benefit from the rewards of healthy agility. The organizational mindset should change and embrace a culture of self-organization and collaboration.
- Agile transformation allows organizations to be reactive and better serve their clients' interests with less effort, which requires significant support and resources to stick it out when things get bumpy.