



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

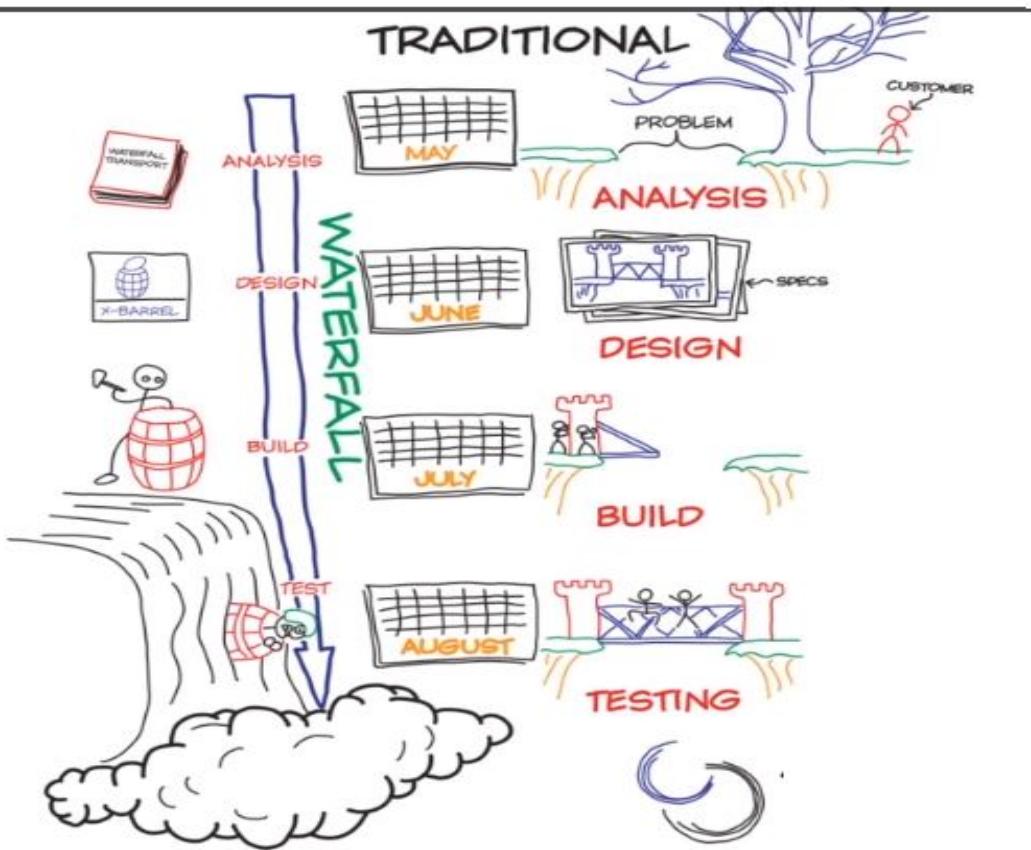
Questions?

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



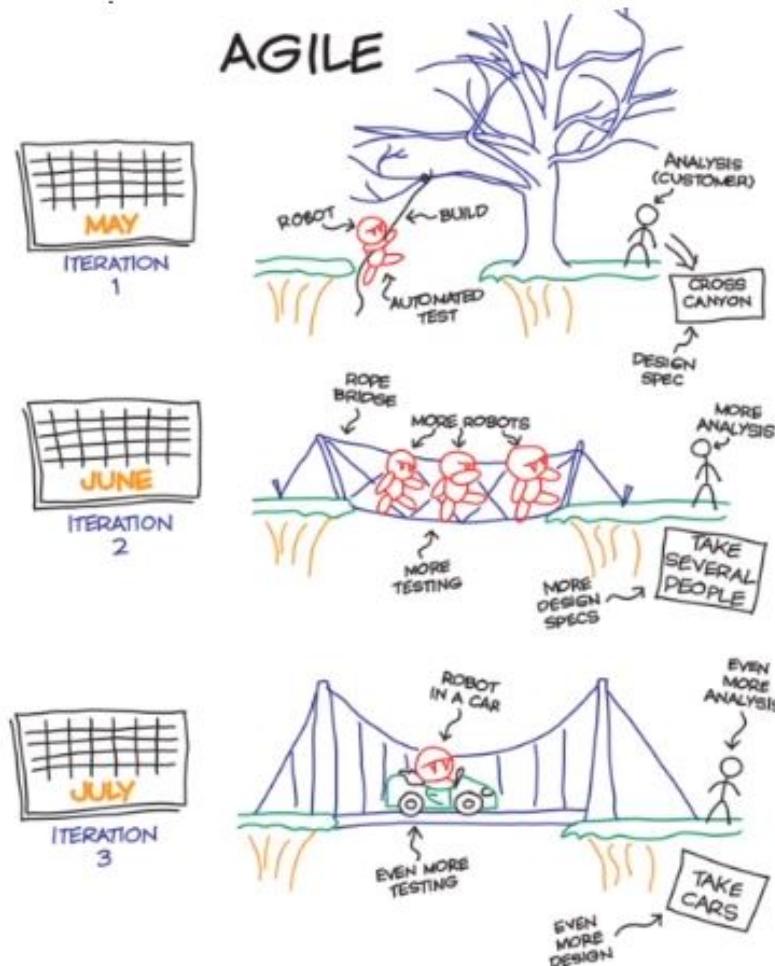
Project Management Model Water Fall Model and Agile

Traditional /Waterfall Development Approach(Rigid)



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

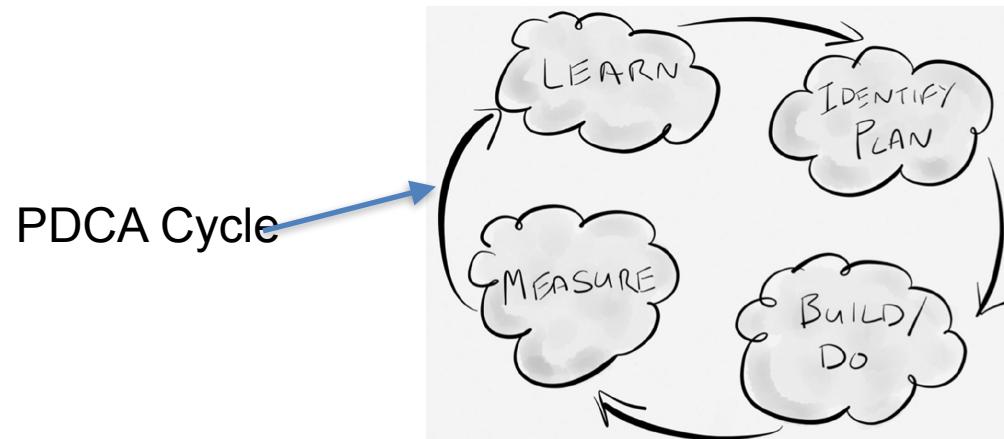
Agile Software Development Approach (Empirical)



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

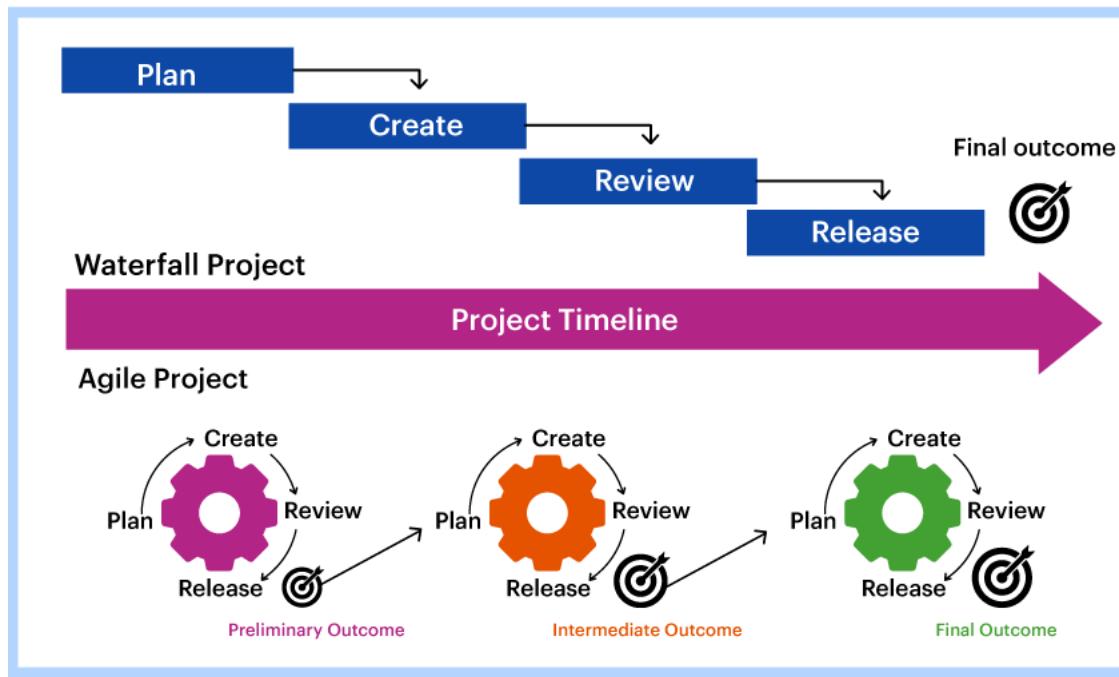
- Responsive planning
- Business-value-driven work
- Hands-on business outputs
- Direct stakeholder engagement
- Immovable deadlines
- Management by self-motivation
- 'Just-in-time' communication
- Immediate status tracking
- Waste management
- Constantly measurable quality
- Continuous improvement



Module-1-Additional Notes

Topics Module-1

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



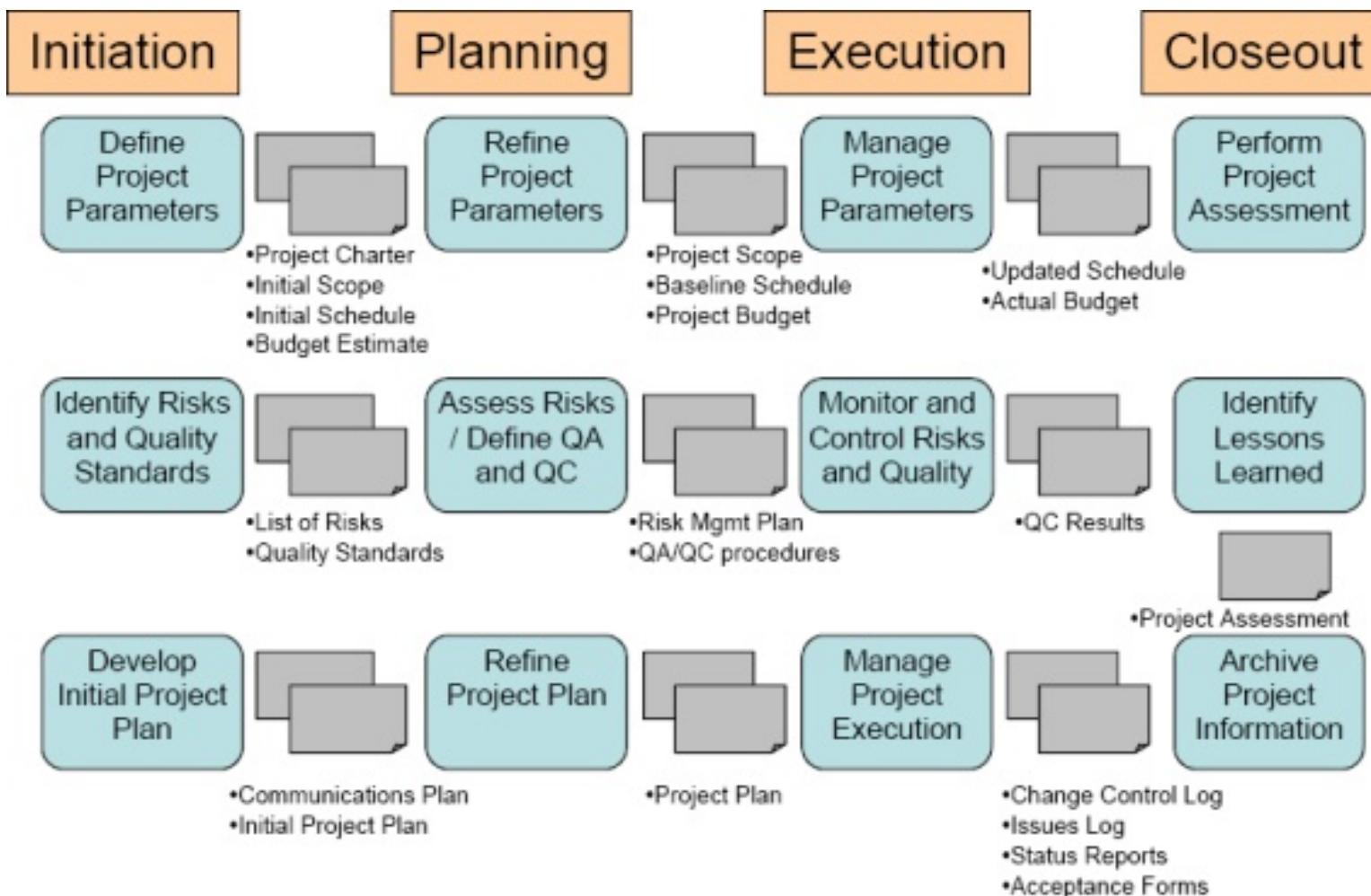
<https://kissflow.com/project/agile/traditional-vs-agile-project-management/>

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**.

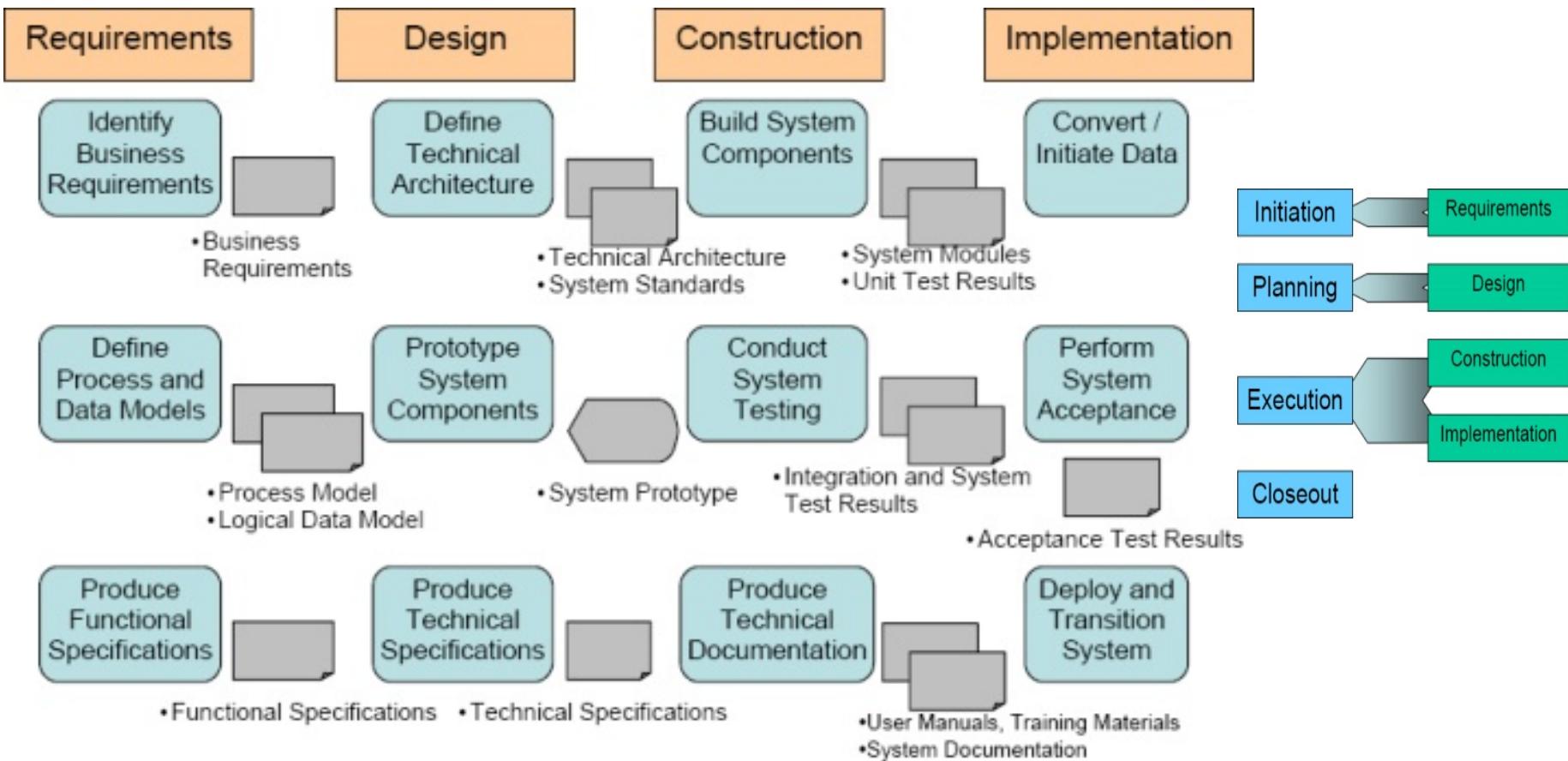
- Development Project
- Maintenance or Support Project (Operational work)

Project Management Phases



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

System Development Phases (Engineering activities)



Traditional Software Development Approaches

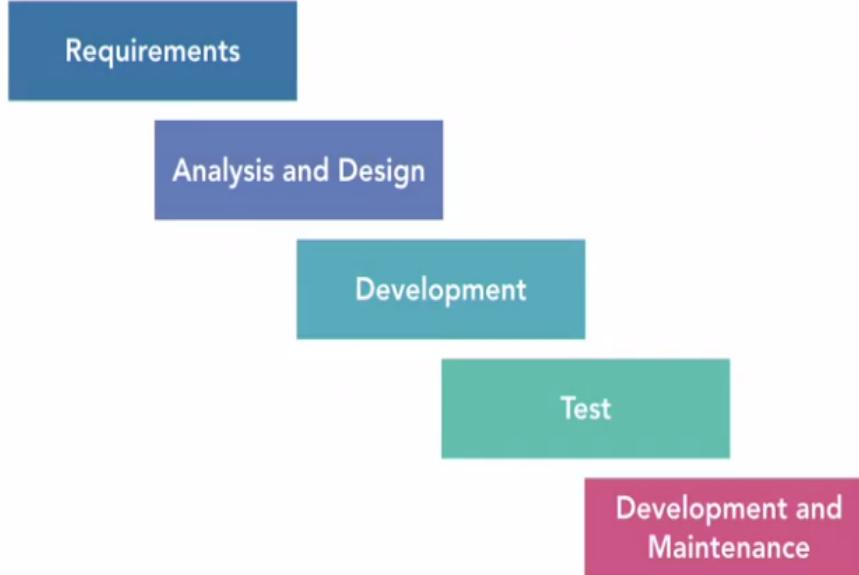
BITS Pilani

Pilani Campus



Traditional Software Development Model – Waterfall Model

Waterfall Approach

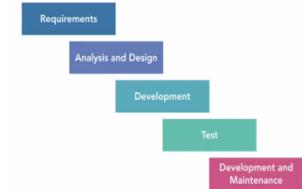


- Move to the next phase only when the prior one is complete — hence, the name waterfall.
- Origin from manufacturing like production plant
- **Upfront Planning**
- **Detailed documentation**
- **Scope of work is generally fixed.**
- Output of a phase becomes input to next phase
- Include well defined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

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

Issues with Waterfall approach

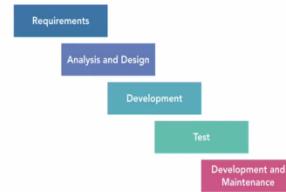
- **Error in one phase will propagate to next phase**
- **Missing requirements will result in missing software feature**
- **Error correction is costly** if it is detected at later phase
- **Customer does not get to see the product** before the early testing phase which is usually two-thirds the way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

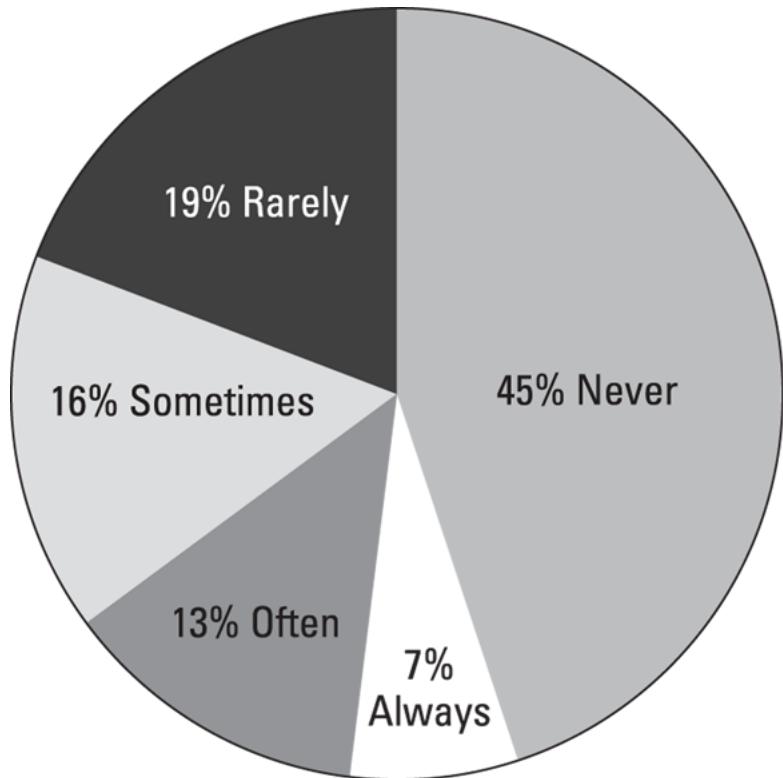
Software Project Success and Failure



- 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.

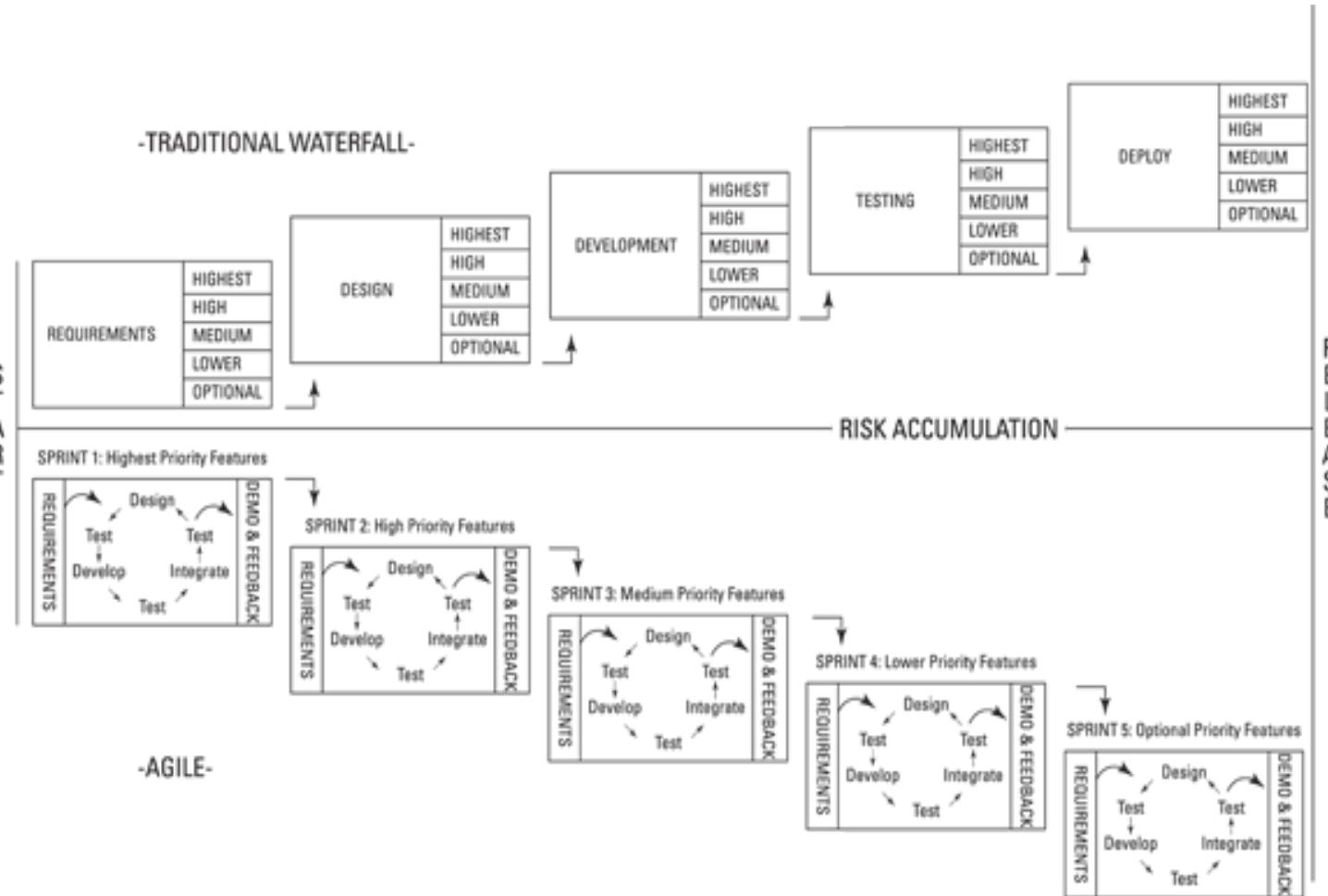
Definable Work

- 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.**

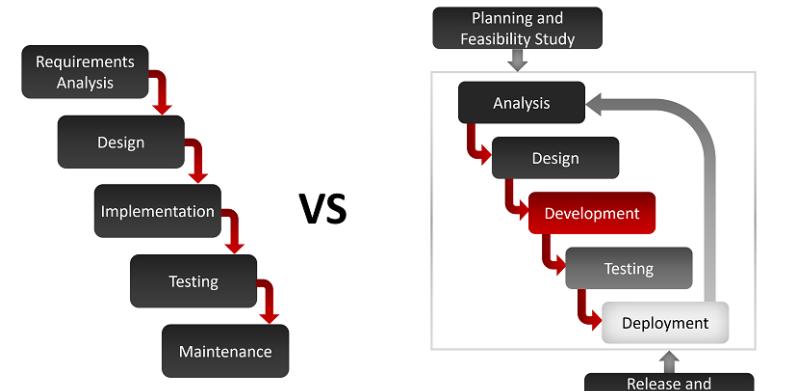
Waterfall vs agile project



Mixing traditional project management methods with agile approaches:

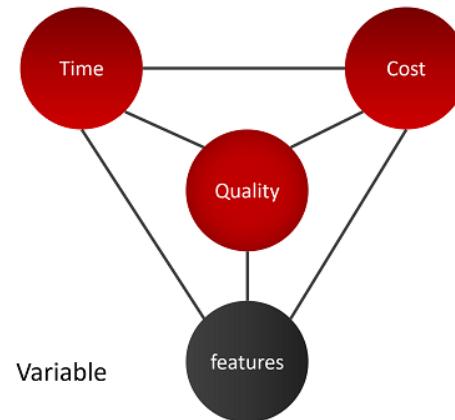
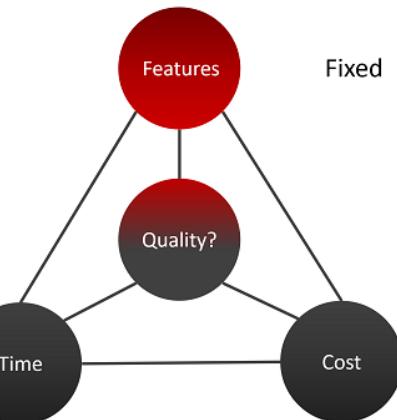
The answer, of course, is **you can't**. If you fully commit to an agile approach, you will have a better chance of deriving benefits of Agile project

Summary: Difference between Traditional and Agile Project Management



Traditional Approach

Agile Approach



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

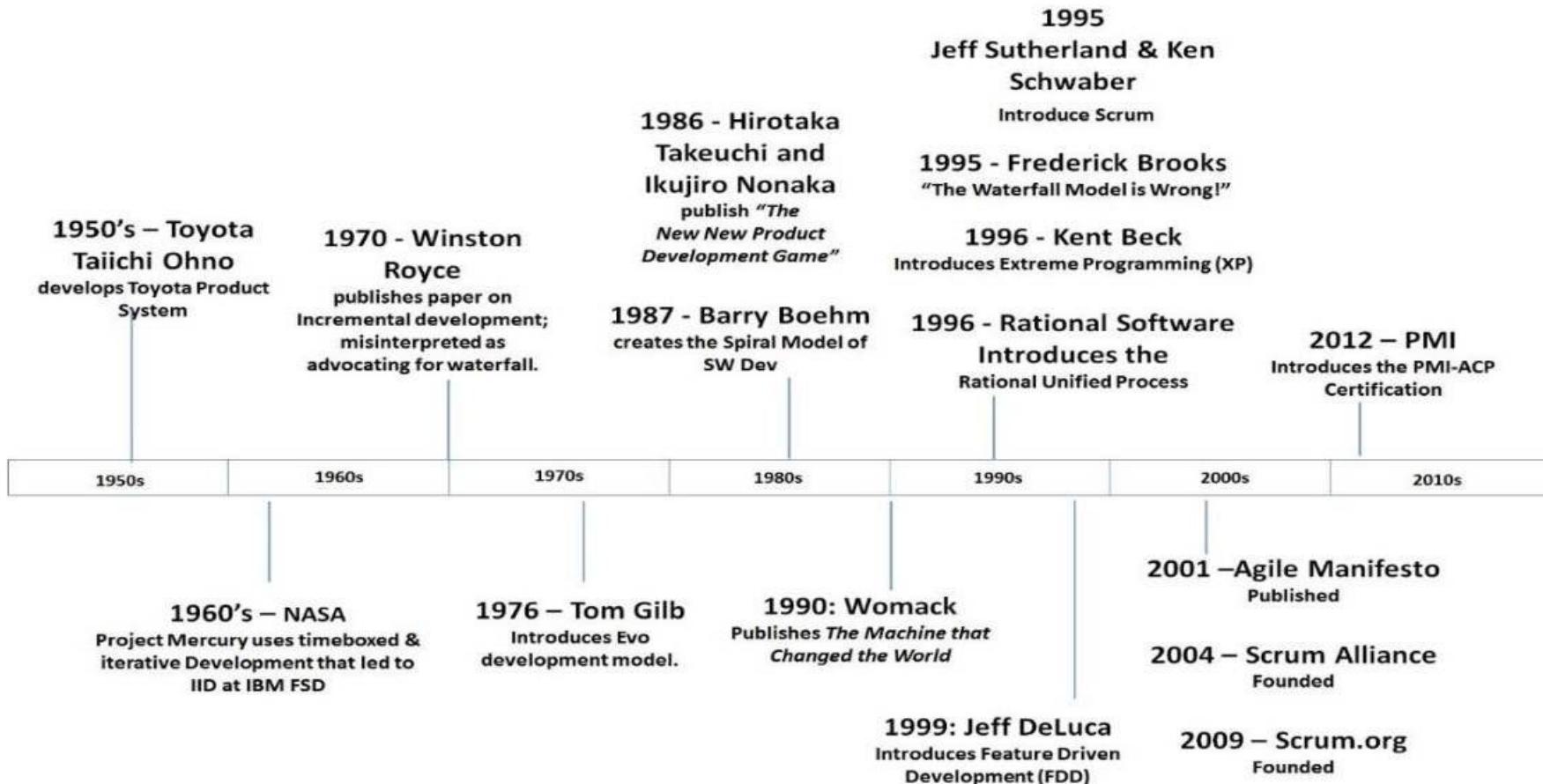
Agile Project Management

- *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.

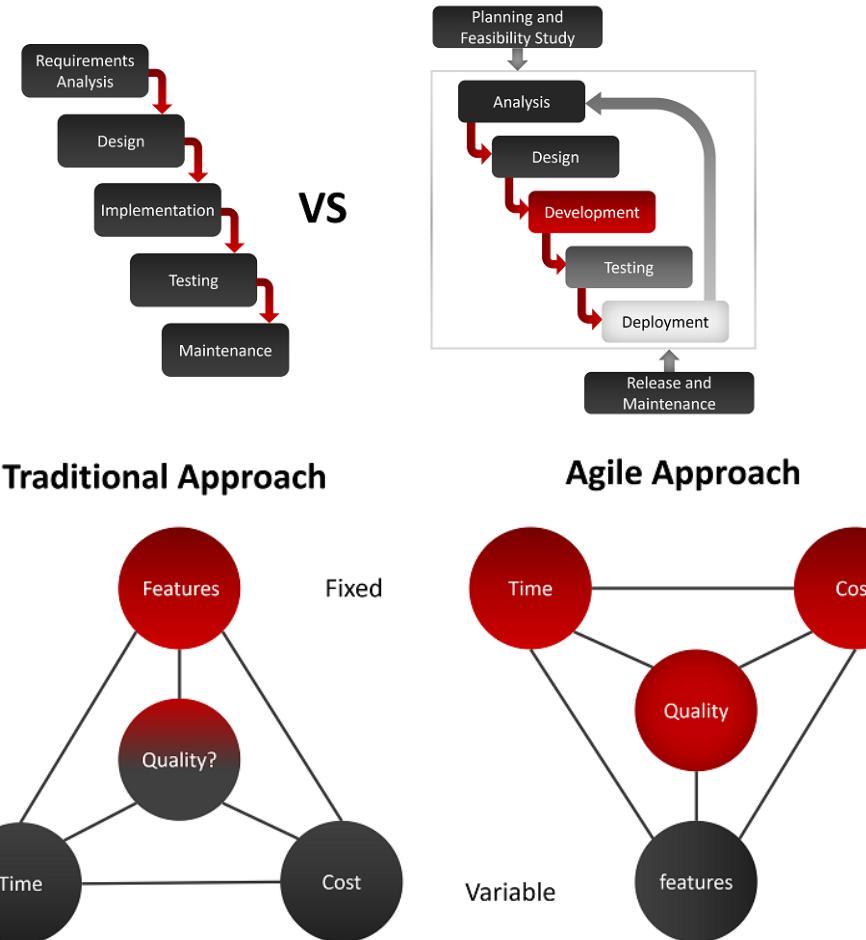
Benefits of Agile Methods/Approaches/ Practices/Techniques ...



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)

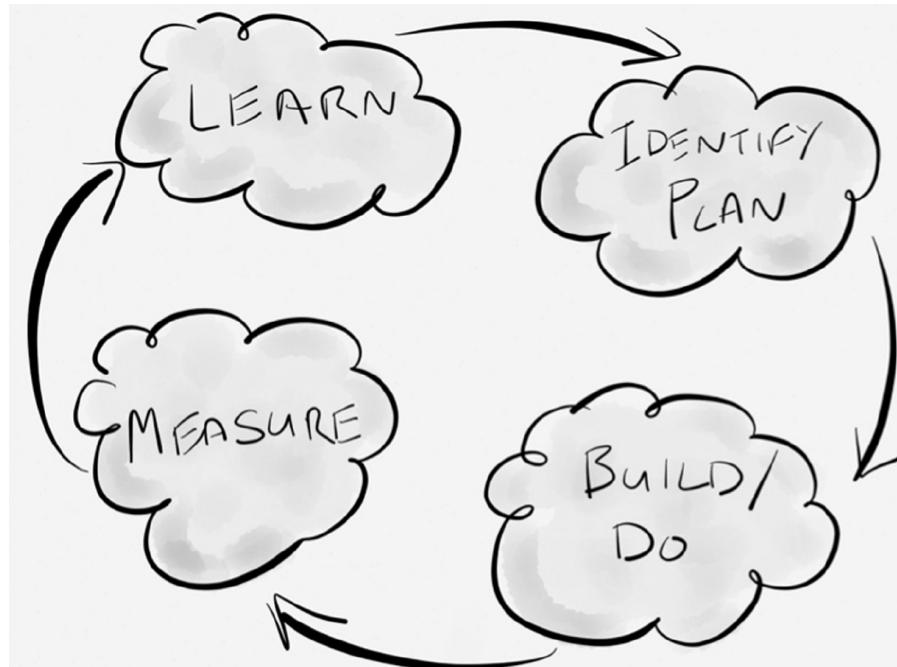
<https://www.kipartners.com/blog/traditional-vs-agile-software-development-methodologies#:~:text=The%20main%20difference%20between%20traditional,in%20Agile%2C%20it%20is%20iterative.>

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

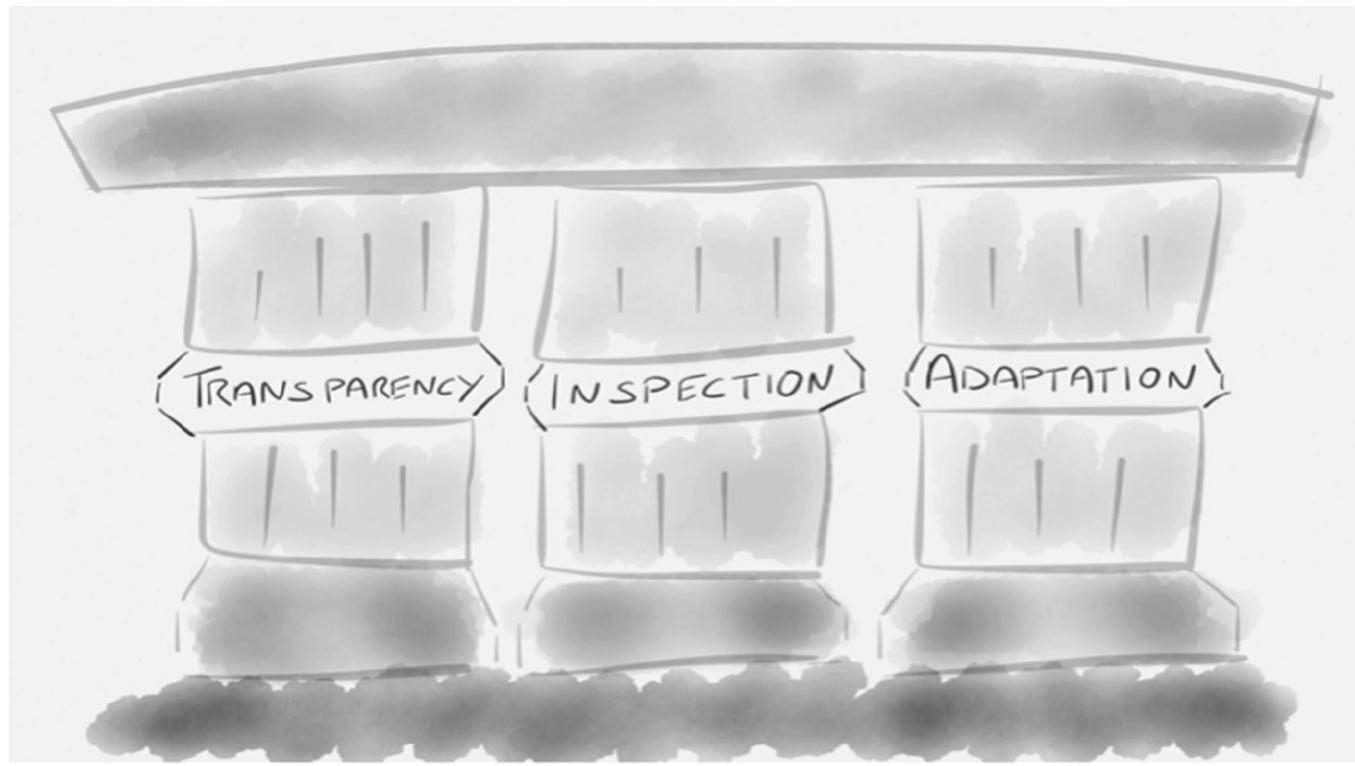
23-Aug-22

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

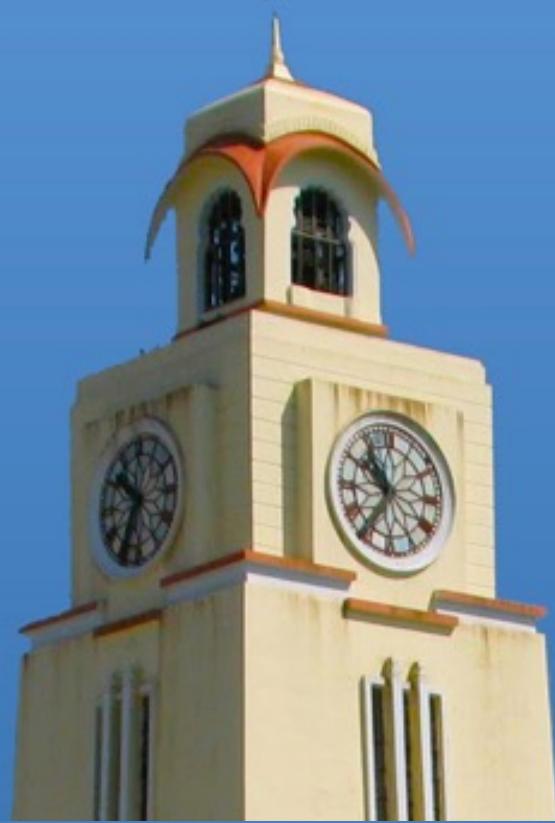
Module-2 Topics

1. Project Life Cycle Models

- Iterative, Incremental and (Adaptive or Agile) Approaches

2. Early Agile Models

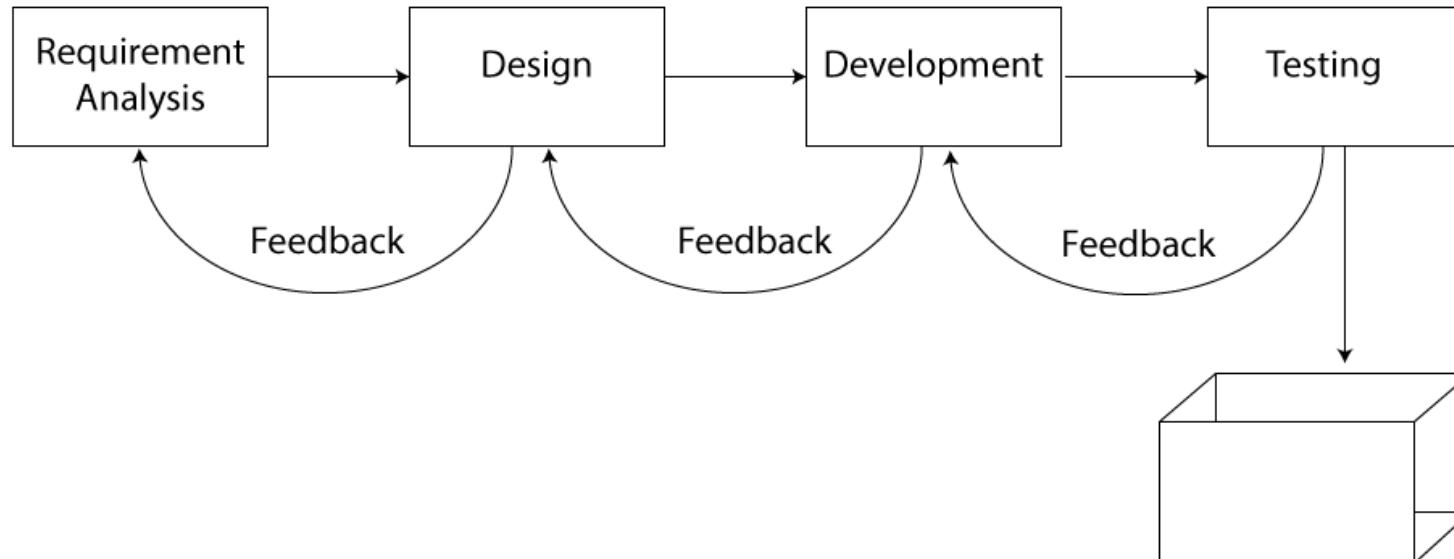
3. Popular Agile Methods



Project Life cycle models

Predictive Project Development Life Cycle

- Fully Plan-Driven aka Waterfall - Risky Invite Failure



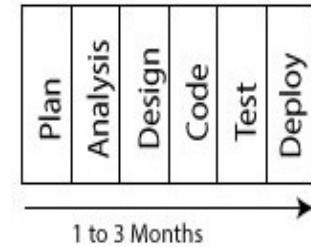
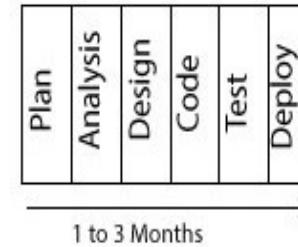
- Goal, Characteristics: Cost, Requirements-Fixed, Single Delivery

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>

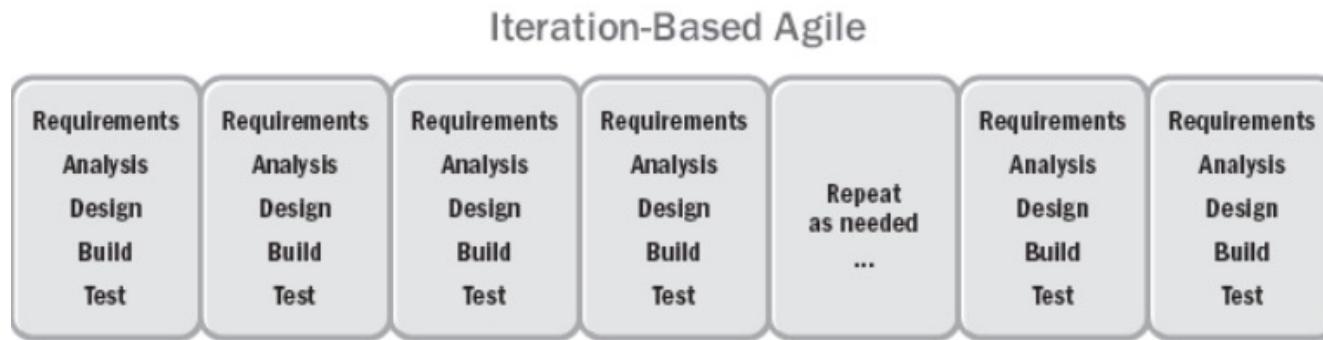
<https://forms.gle/A9W5ikfCJPaT5tbg8>

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.

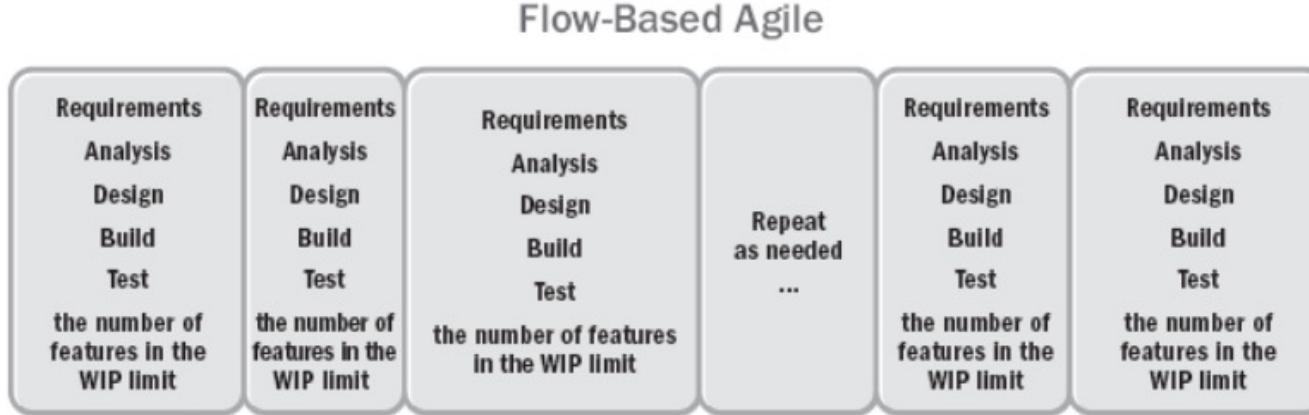
Fixed Time box : 1-4 weeks equal duration for each iteration

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

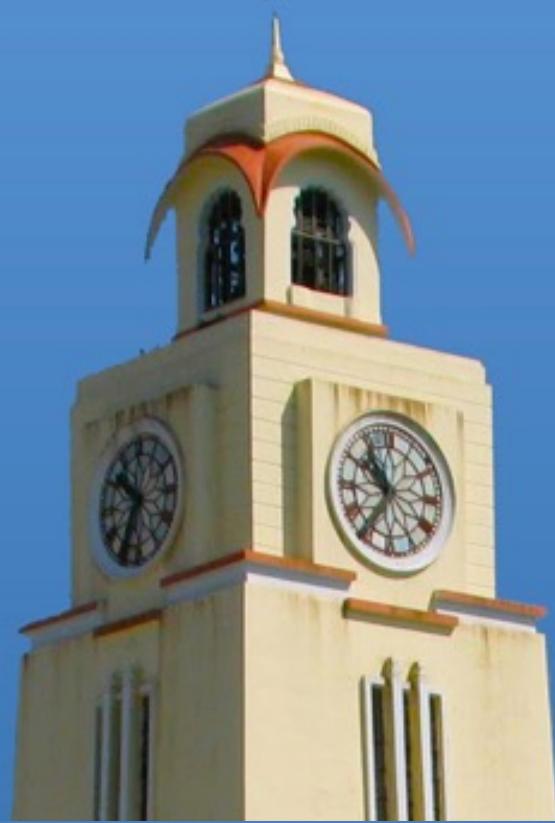
Q&A

Q3,Q4,Q5

<https://forms.gle/bz3784DRQYHD5BnN9>

<https://forms.gle/TZpHt55eRhKjnZT9>

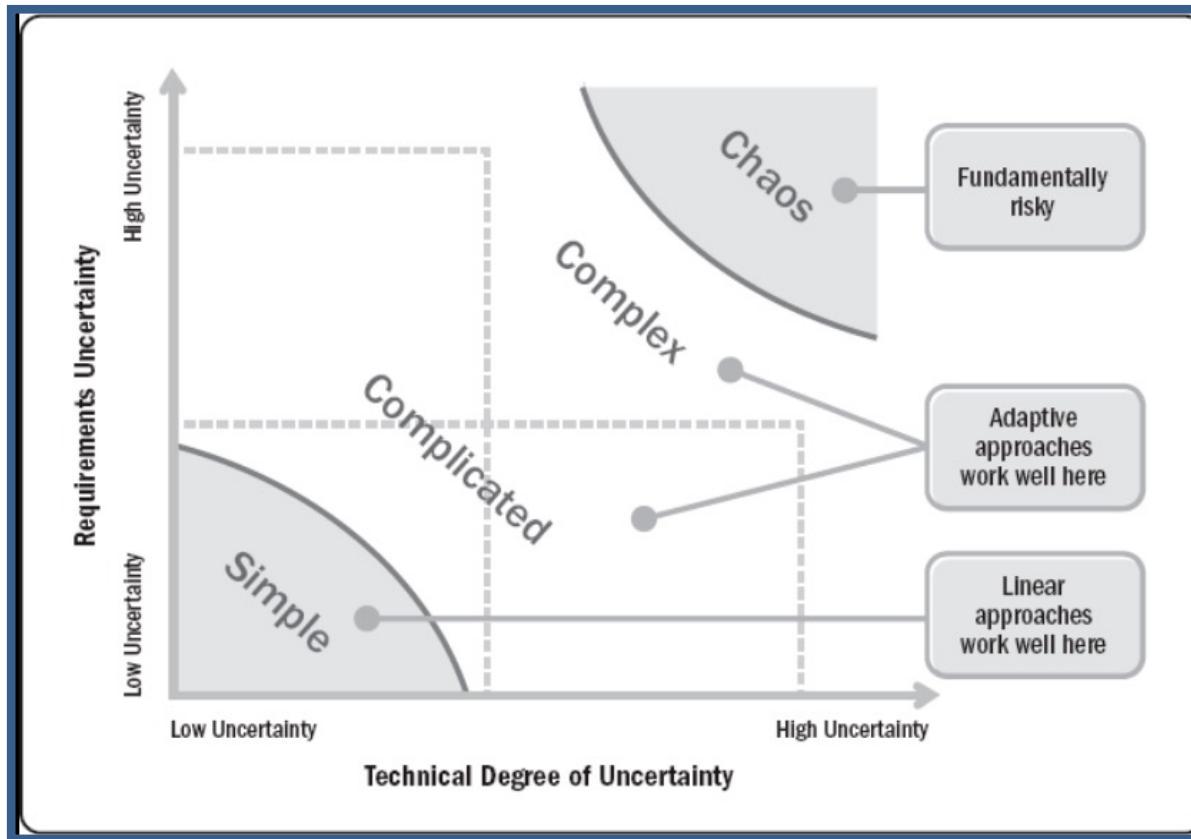
<https://forms.gle/7XCdp9E7yr9ABG2r8>



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



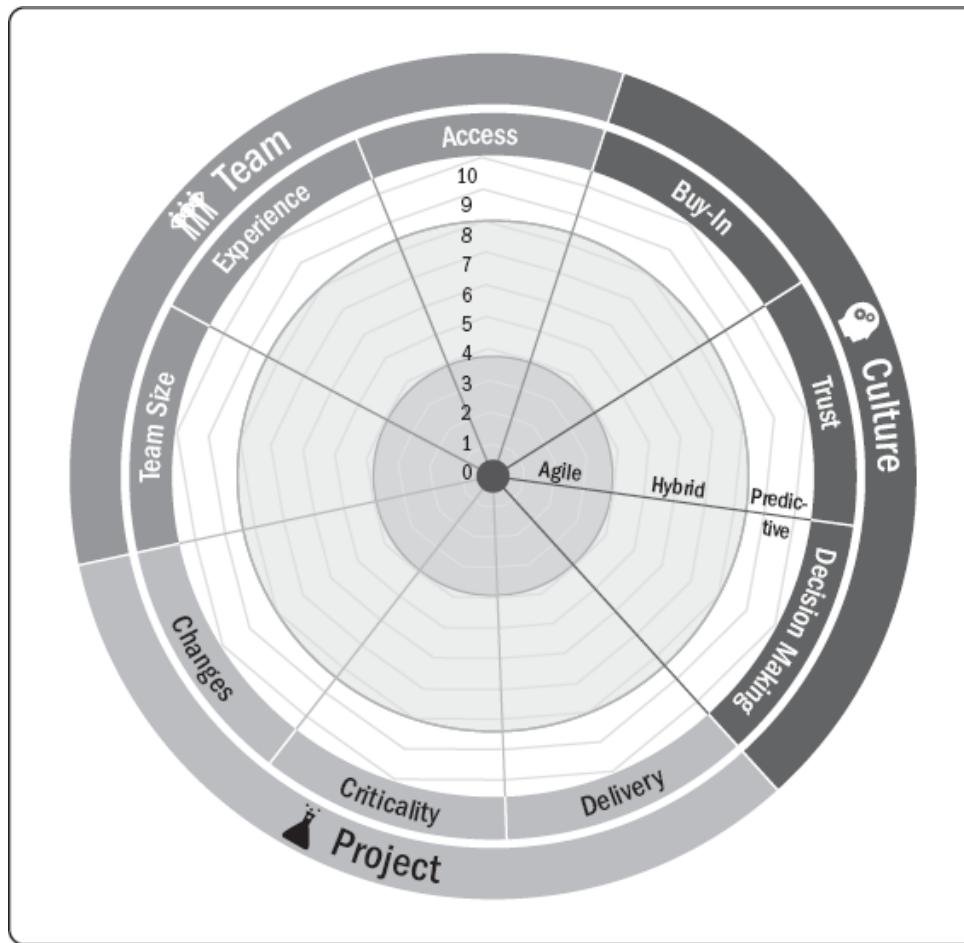
Pronounced as: kun-ev-in

Emergent

Developed in the early 2000s by David Snodown

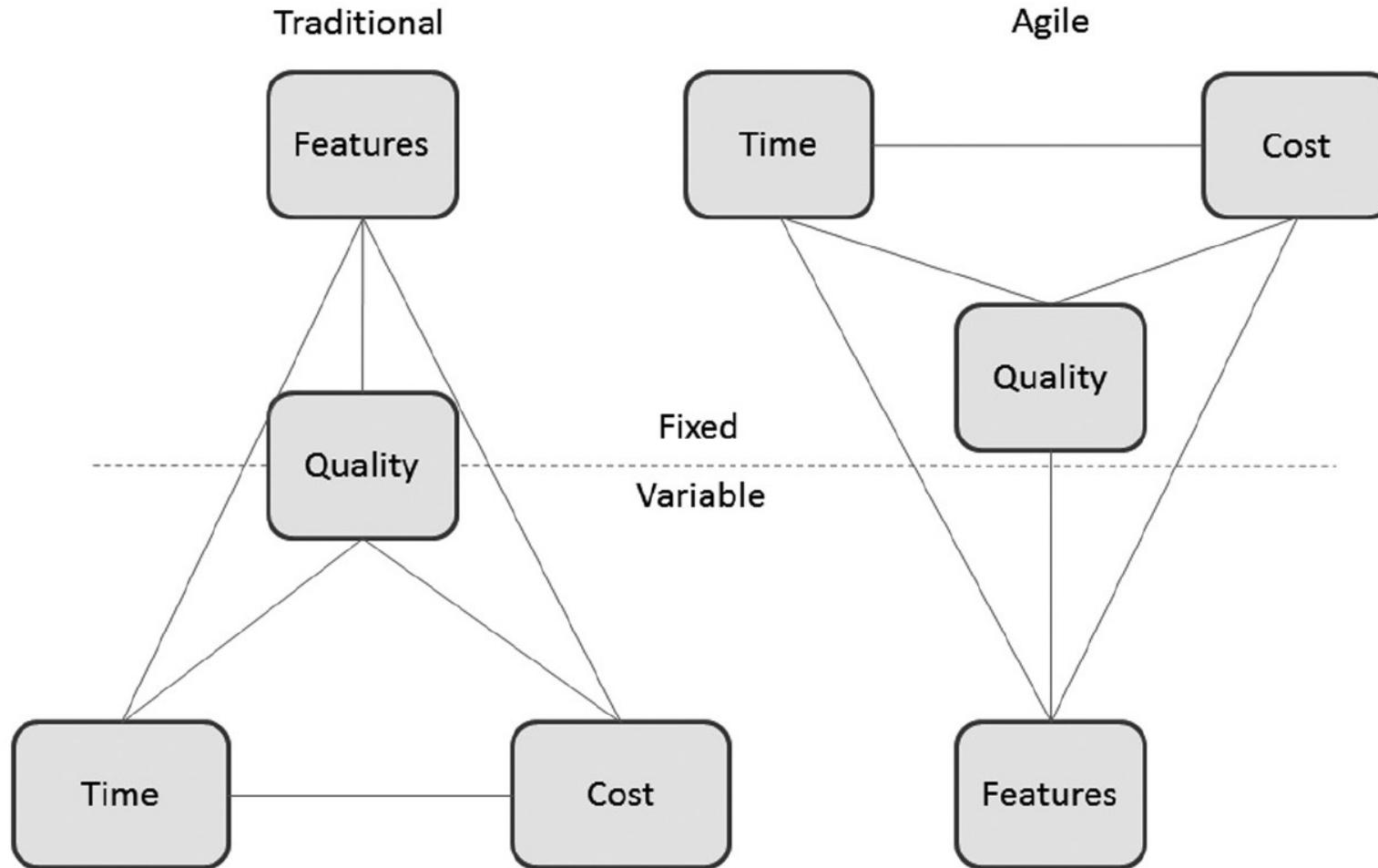


Agile Suitability Filter

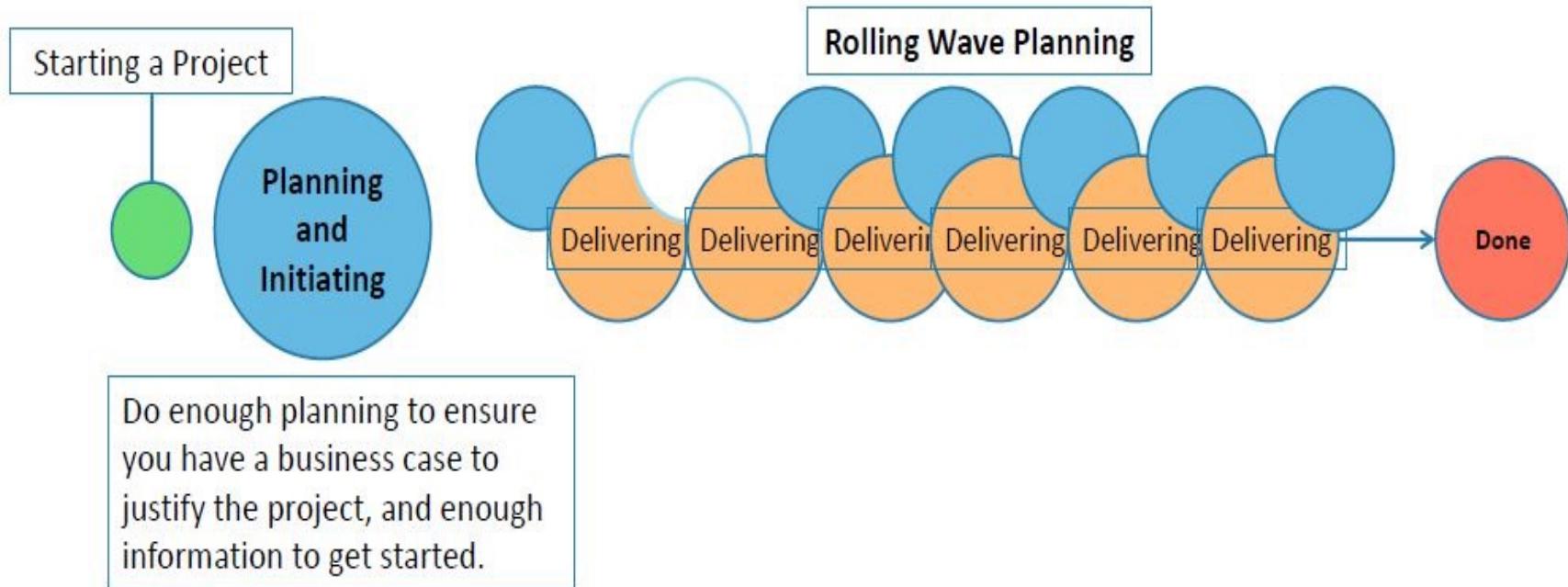


Attributes	Assessment
Buy-In	0-Yes, 5-Partial, 10-No
Trust	0-Yes, 5-Probably, 10-No
Decision Making	0-Yes, 5-Probably, 10-Unlikely
Incremental Delivery	0-Yes, 5-Maybe/Sometimes, 10-Unlikely
Criticality	0-Low, 5-Medium, 10-High
Changes	0-High, 5-Medium, 10-Low
Team Size	1-Small (<10), 5-Medium (>80),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

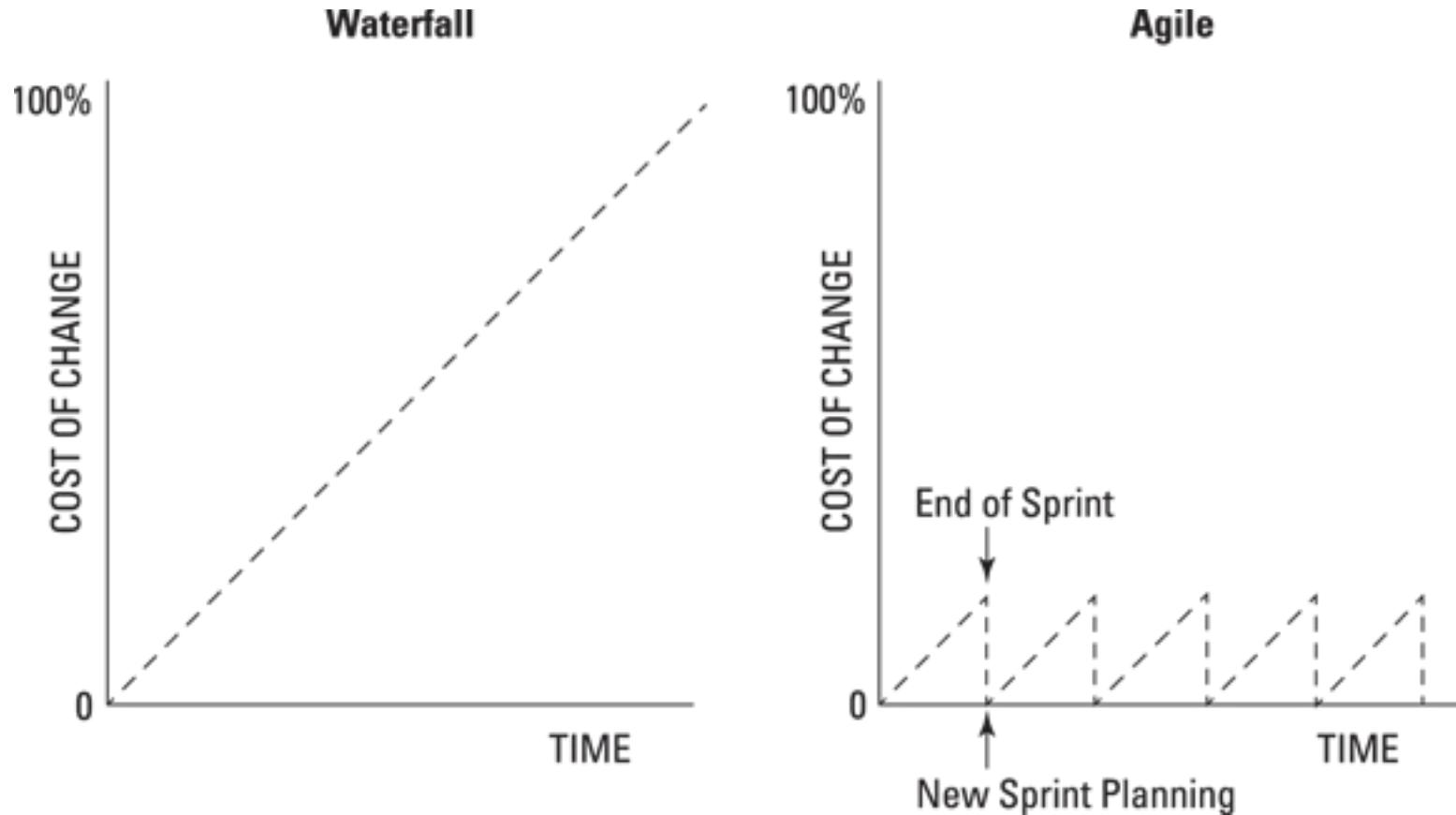


Rolling Wave Planning or Progressive Elaboration



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

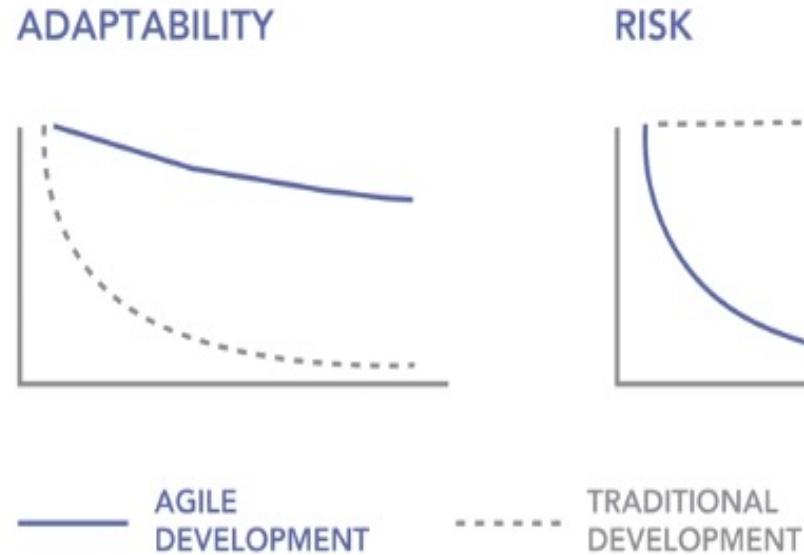
Cost of Changes: Agile vs Traditional Development



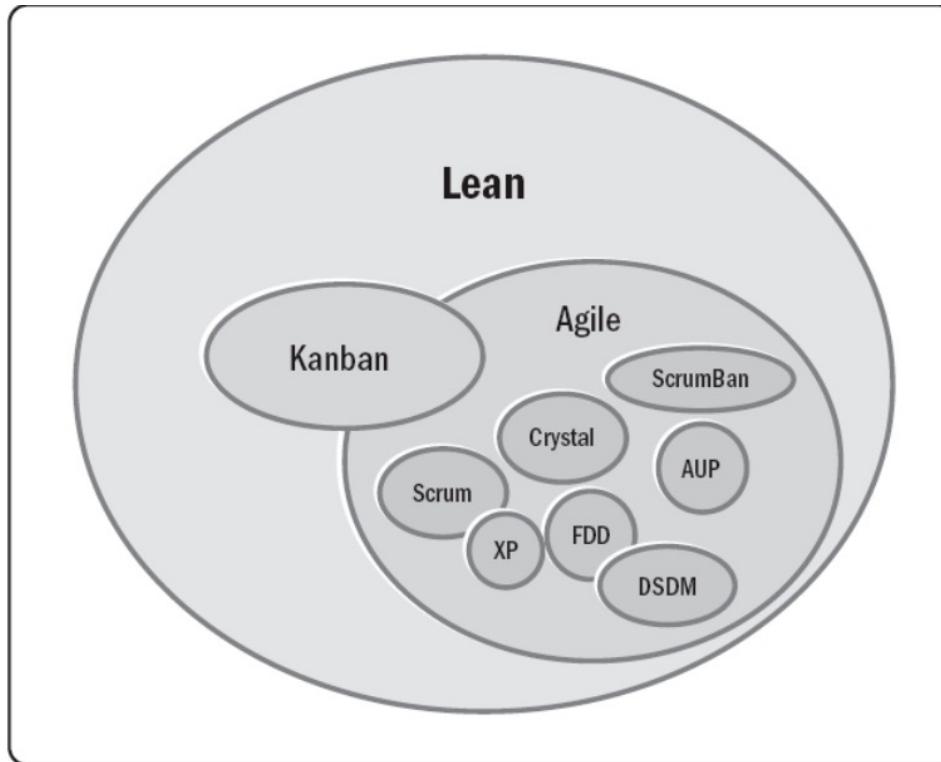
Adaptability and Risk Agile vs Traditional Development



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

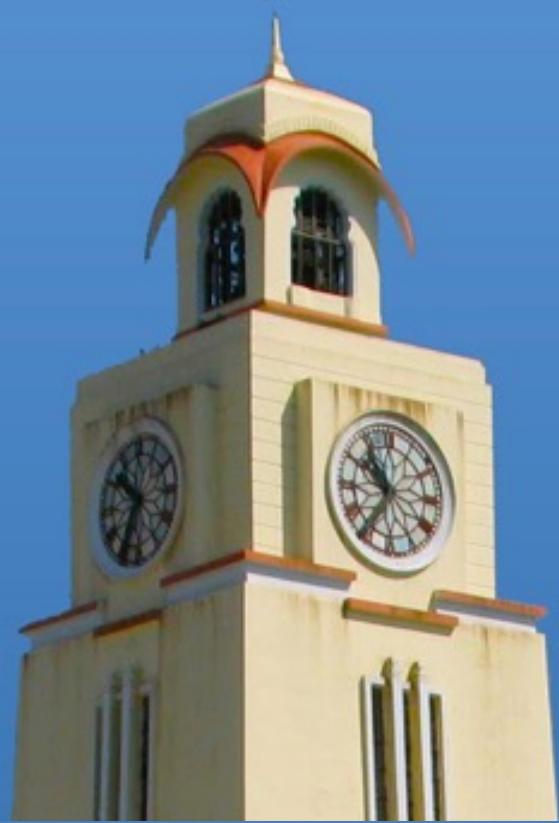
Q6,Q7,Q8

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

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

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

End Contact Session-2



Module-2 Additional Notes

Life Cycle

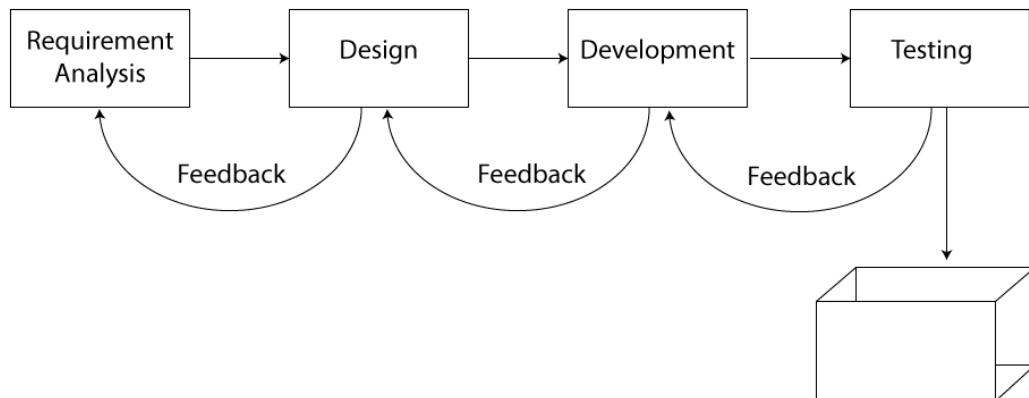
- The **sequence of actions** that must be **performed** in order to **build a software** system
- **Ideally** thought to be a **linear** sequence: **plan, design, build, test, deliver**
 - This is the waterfall model
- **Realistically** an **iterative process**
 - Iterative, Incremental, Agile Process

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



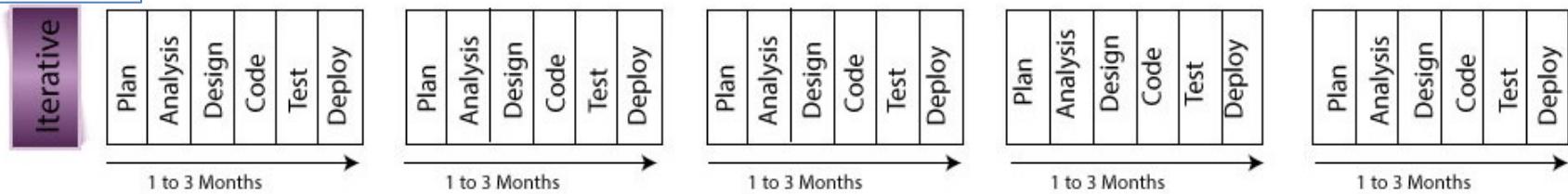
- Requirements/Scope is fixed
- Single delivery
- Goal: Manage Cost
- Minimal feedback changes
- Team is matured in estimation, technology etc..
- Project governance model exists
- **Don't expect long feedback cycle**, If this happens, this lifecycle not suitable for the project

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**

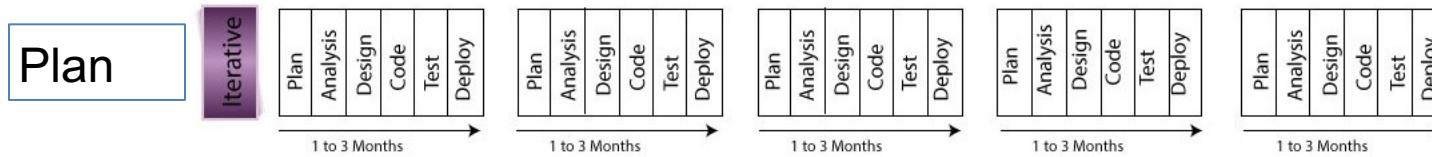
- Deliver result** at the end of each iteration.
- Result may **not be usable**
- E.g. 1 year project divided into 3 to 4 iterations

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

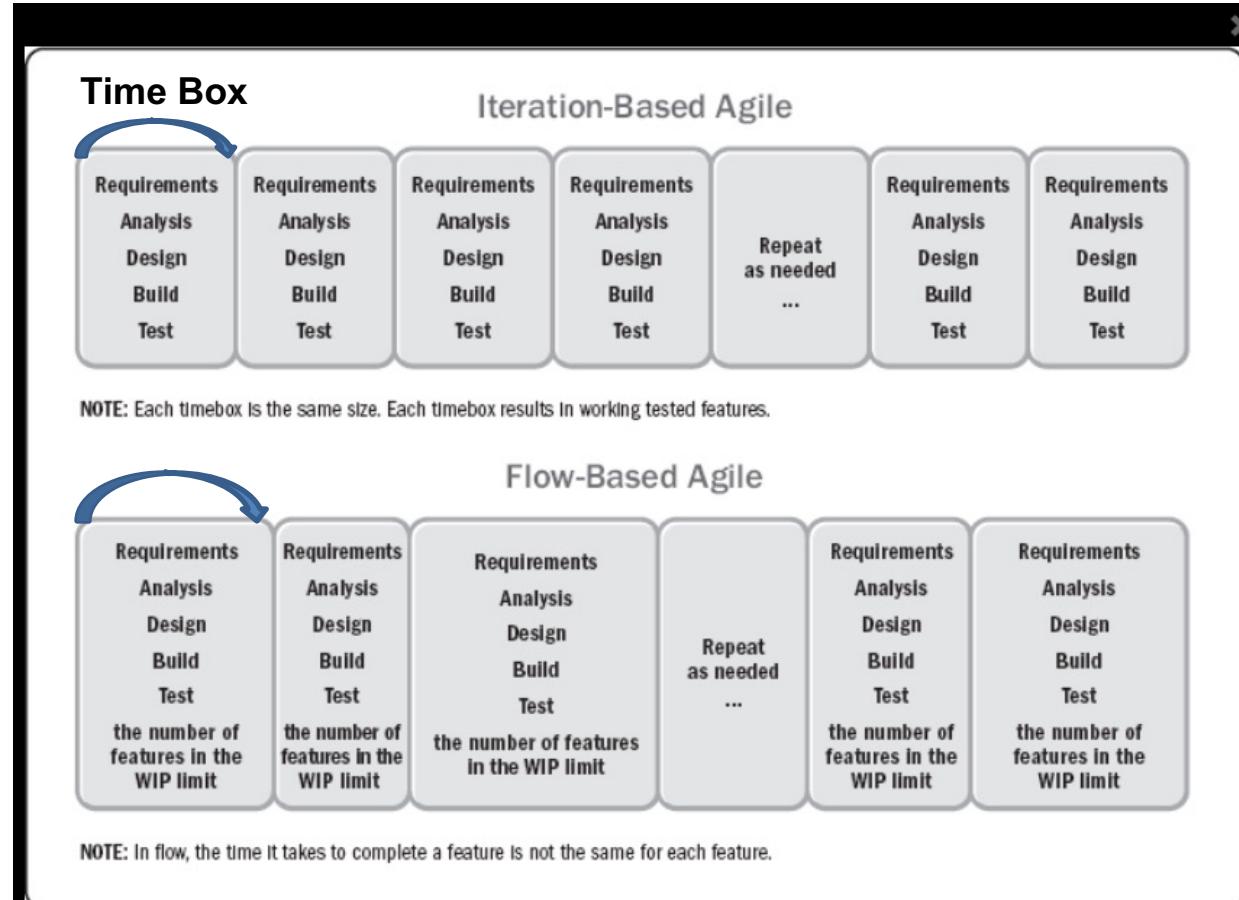
- In an incremental approach, one aims to **build pieces of program/product that is complete in features and richness**. Product **increment** is **usable**.
- In this case, each functionality is built to its fullest and **additional functionalities are added in an incremental fashion**.



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

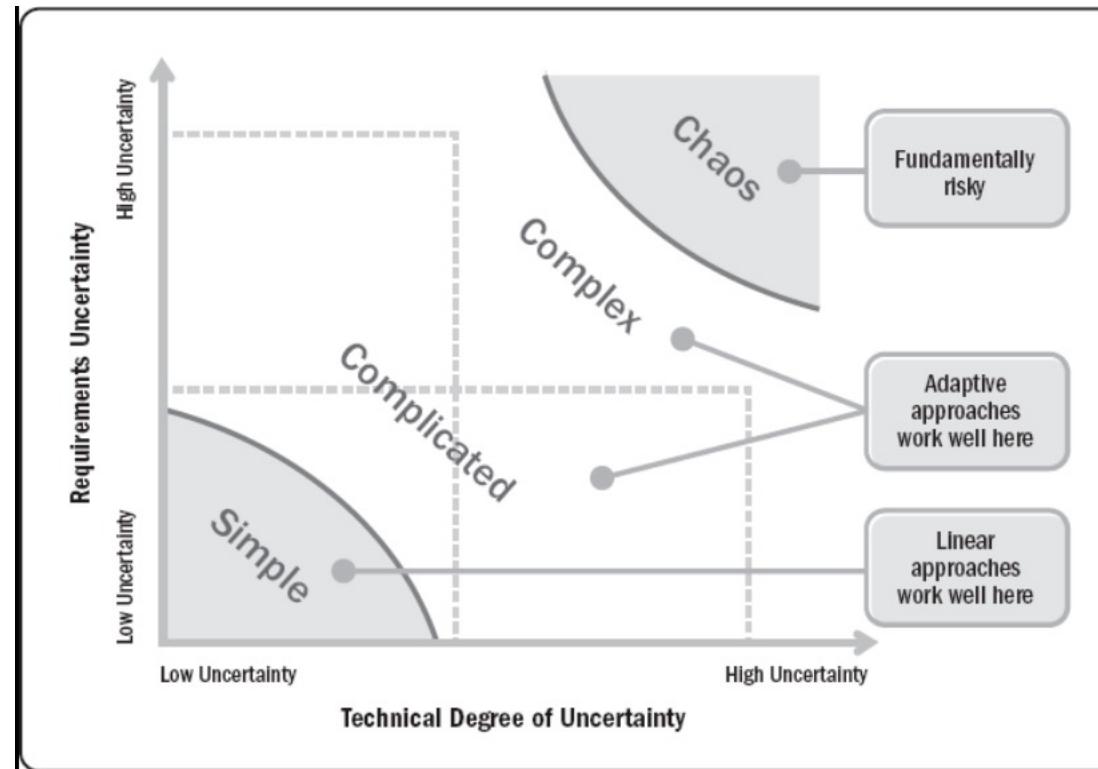


Delivery Environments and Agile Suitability

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



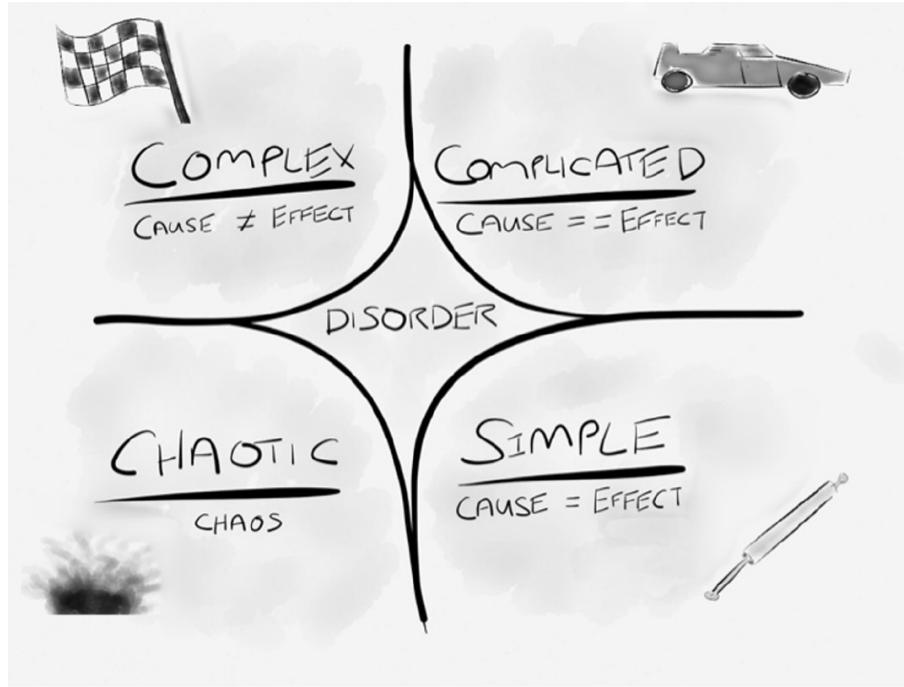
- Simple Environment: Use defined Process like Waterfall
- Complicated/Complex/Anarchy Environment: Use Empirical process like Agile.
Example: New IT product development

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

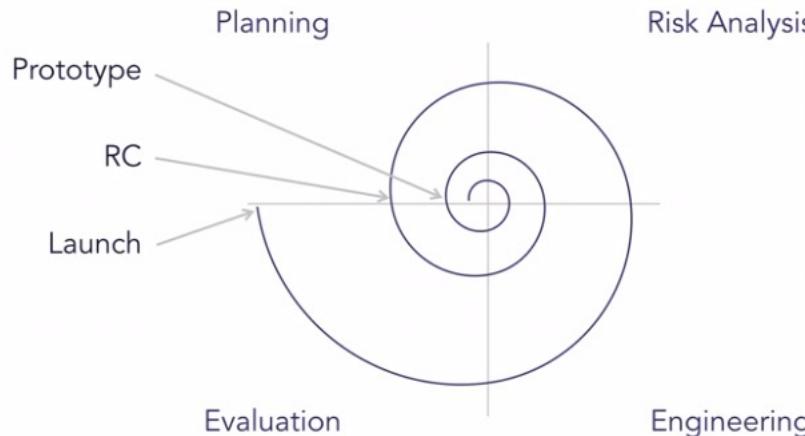


BITS Pilani
Pilani Campus



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

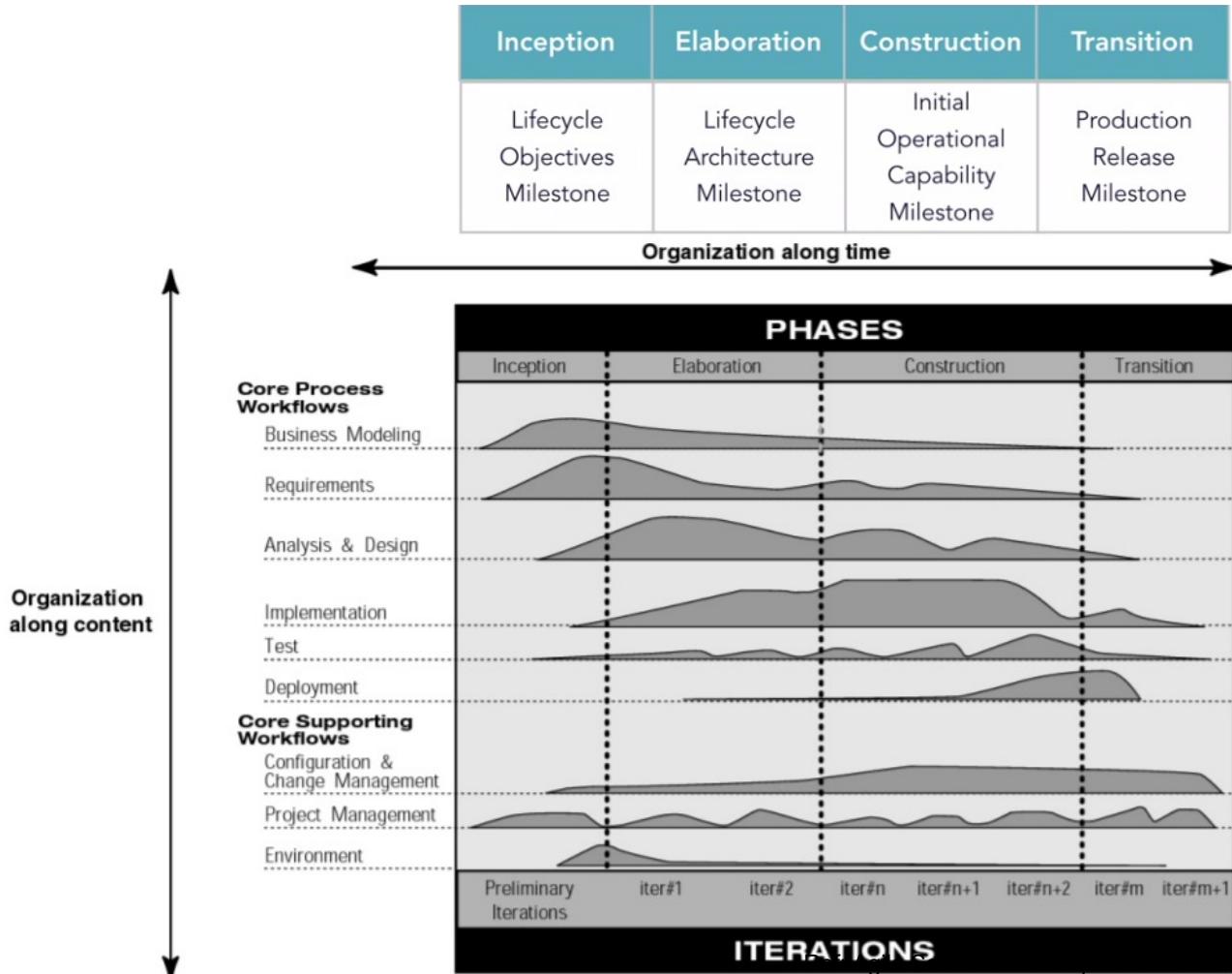
Rational Unified Process (RUP)



- 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

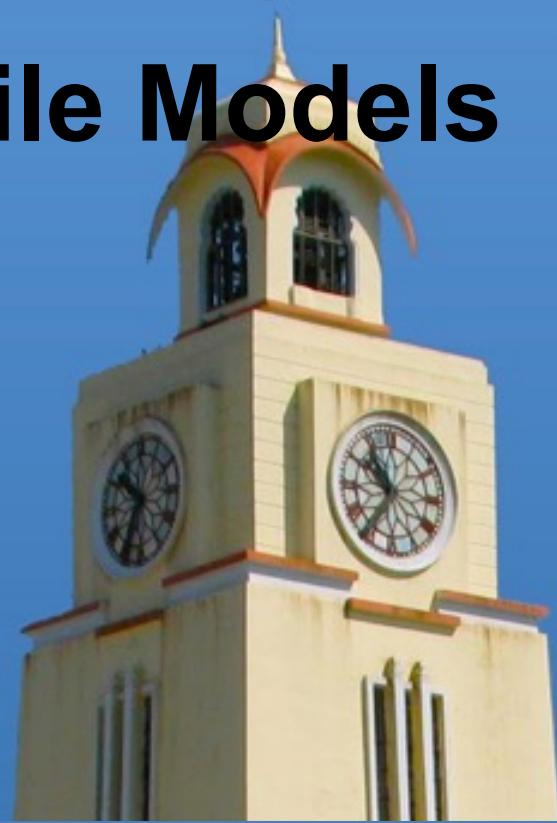


X-Axis: RUP Phases,
Dynamic

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

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

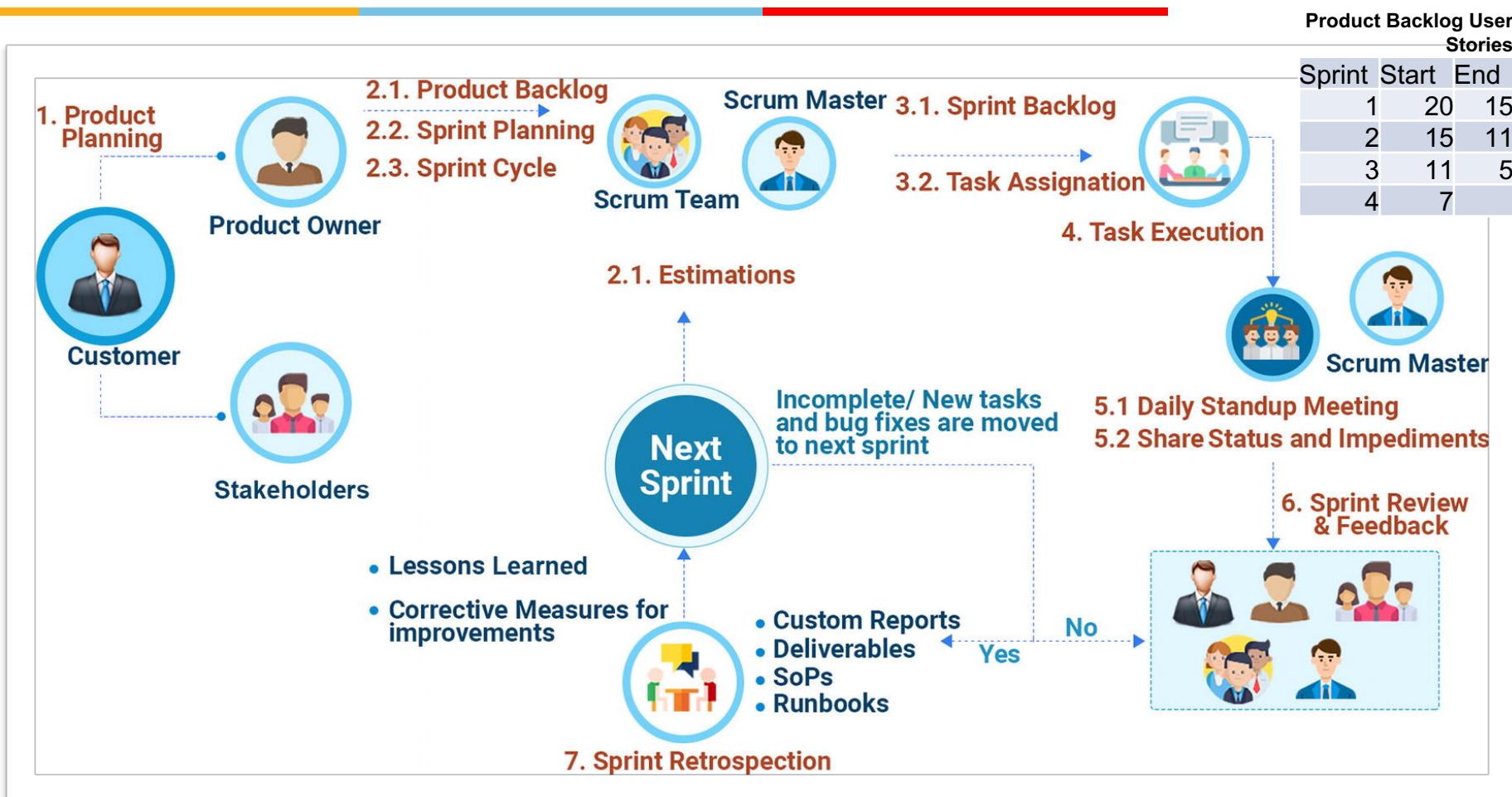


SE ZG544 , Agile Software Process Lecture No. 4 - Agile Methodologies

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			

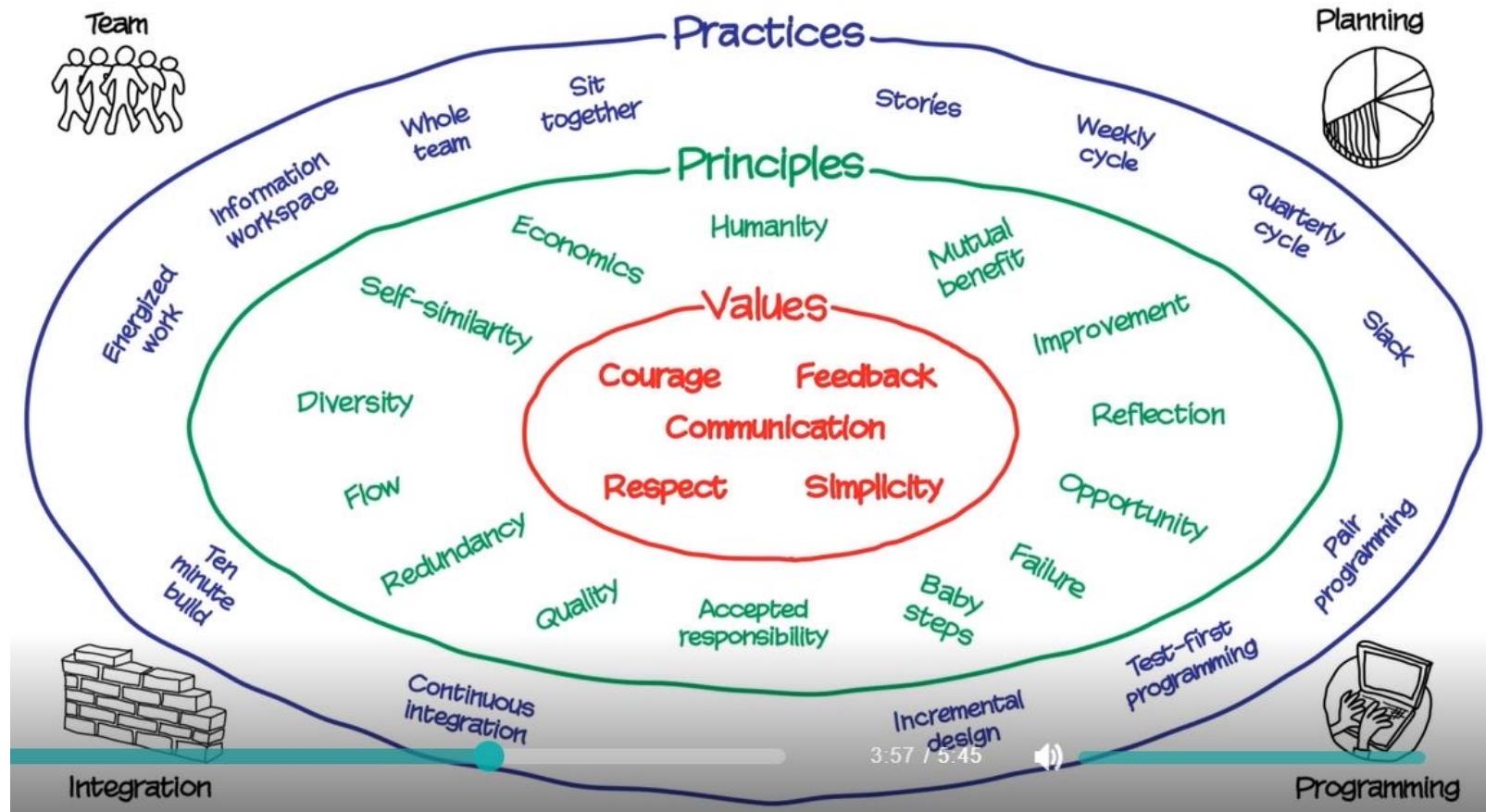
Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 2005

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? &

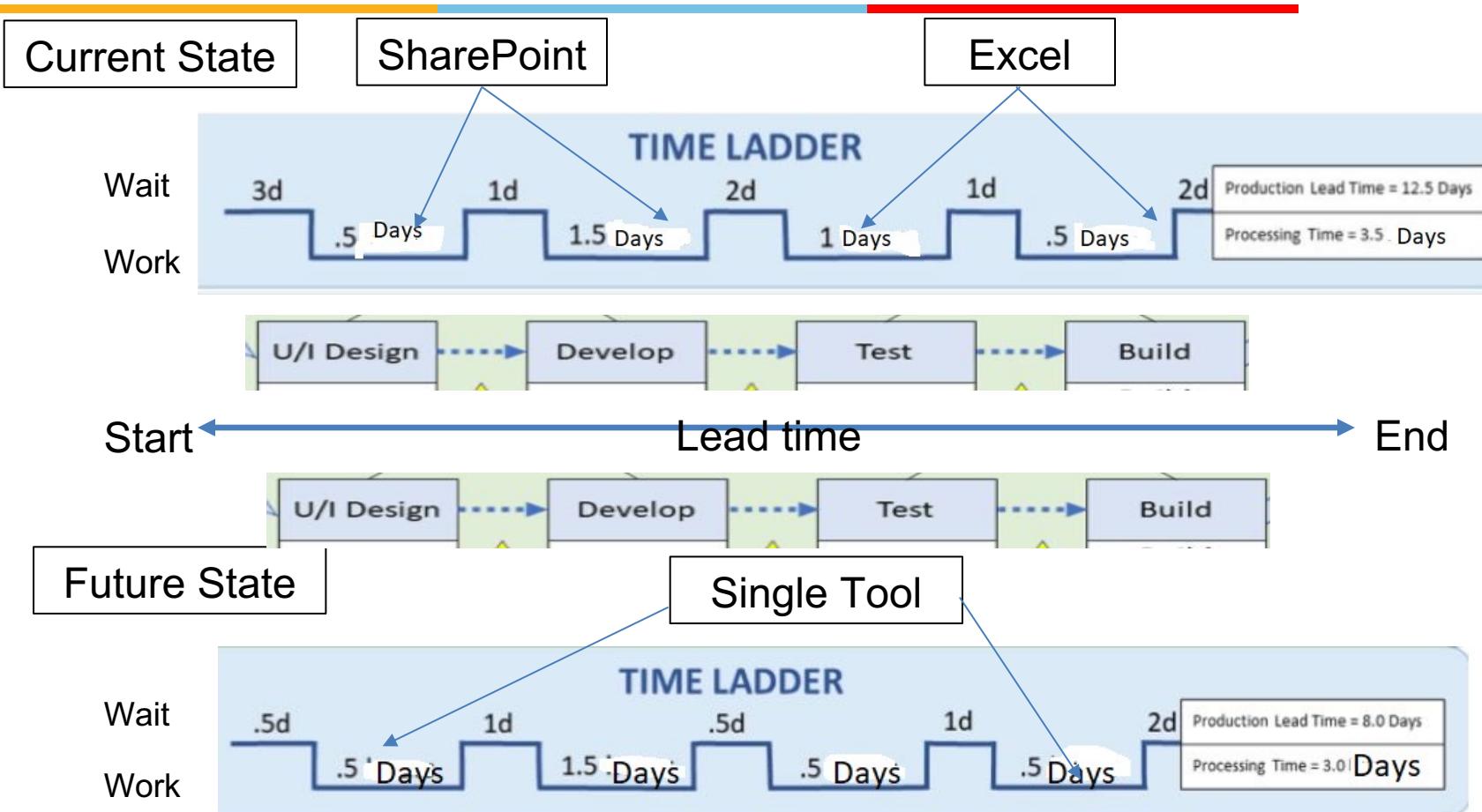
Seven Principles of Lean

- 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

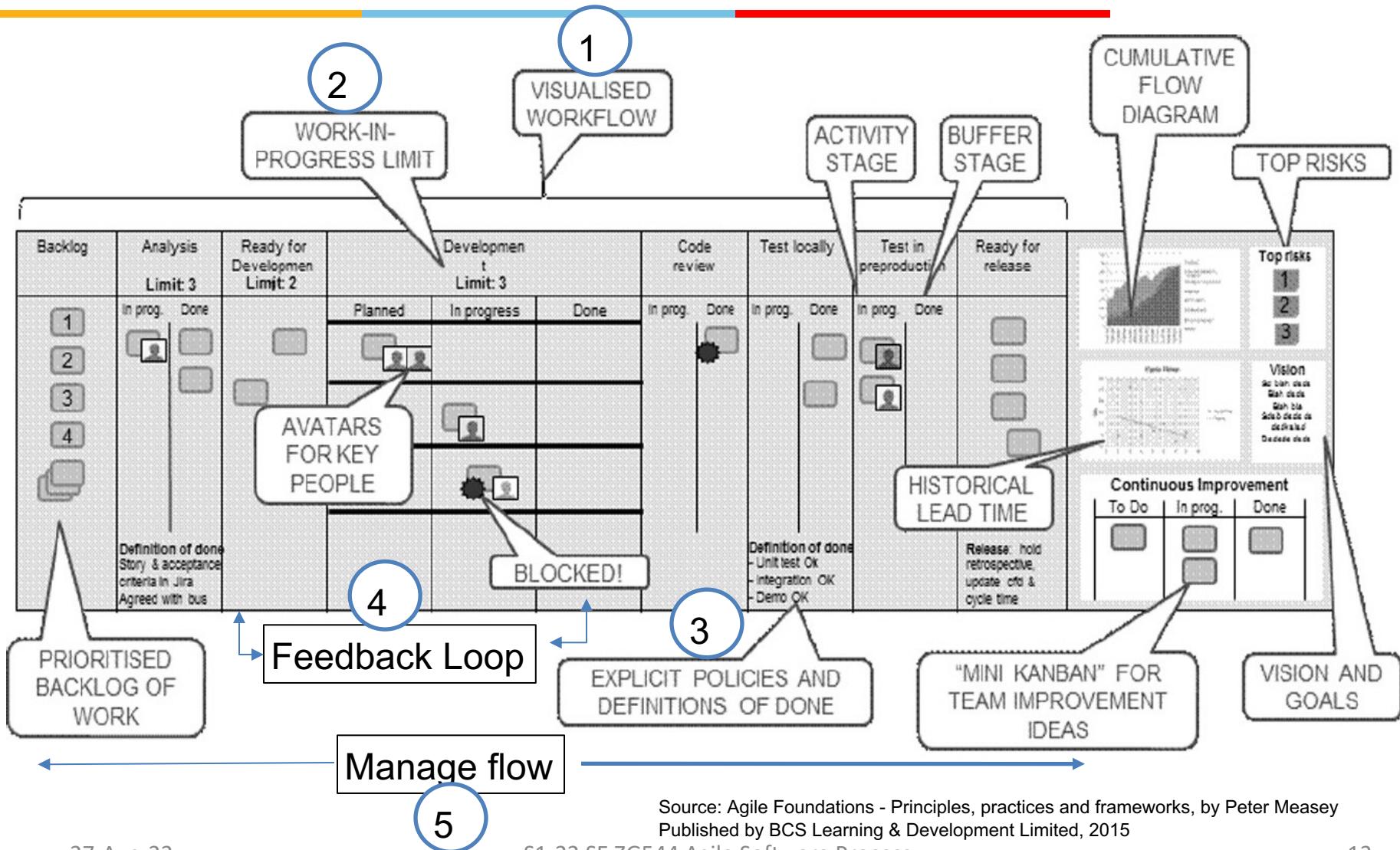
Current State: $3.5/(12.5)*100 = 28\%$; Future state: $3/(8)*100 = 37.5\%$

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.

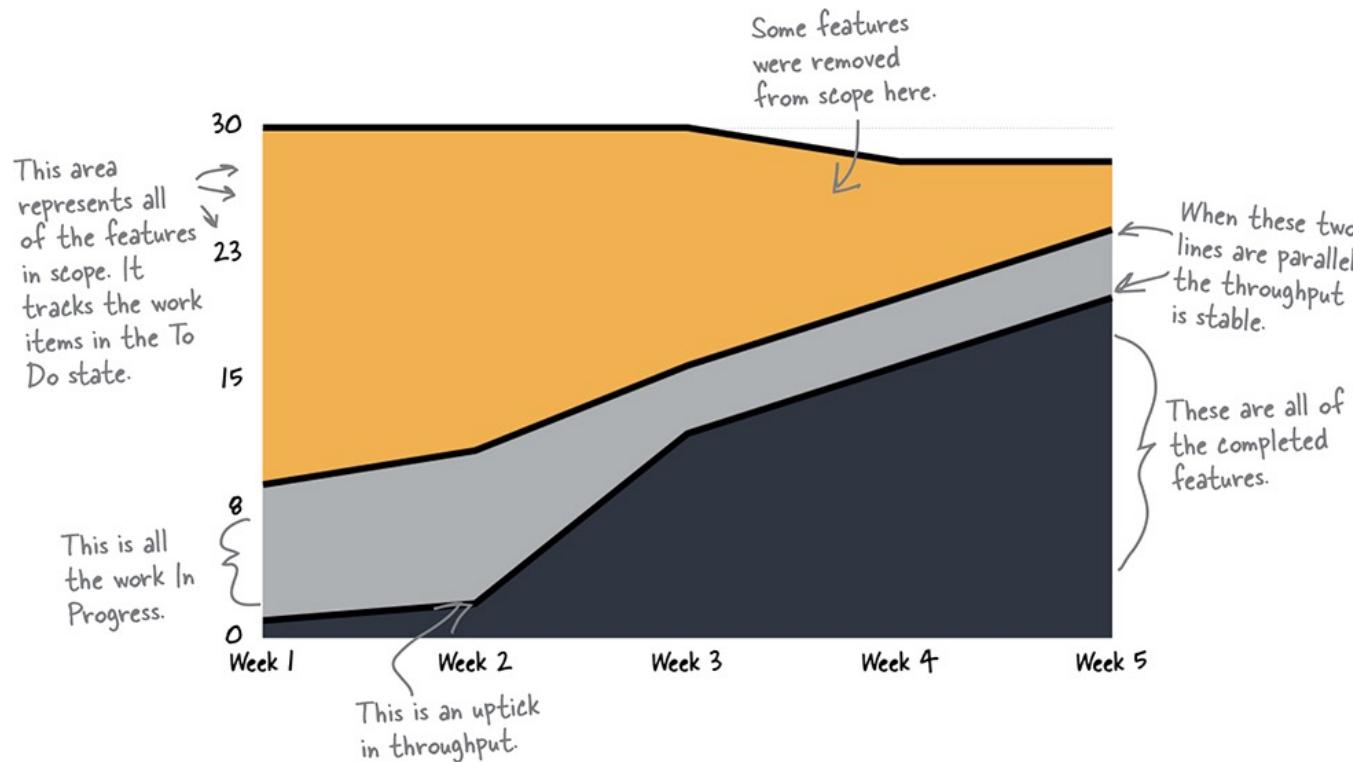
Typical Kanban Board



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

Cumulative flow diagram (Example-1)

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



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

Q&A

- » 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



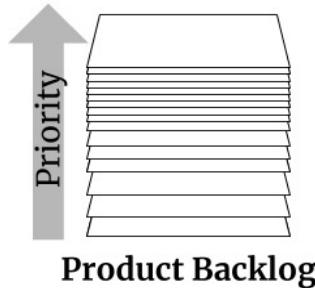
Scrum Master

- 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

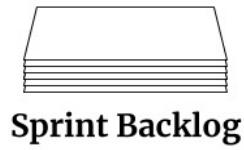
Ref: The Agile Developer's Handbook, by Paul Flewelling, Published by Packt Publishing, 2018

Scrum Artifacts

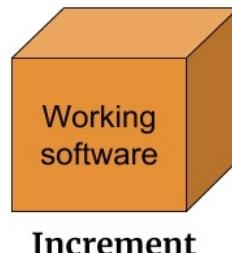
Scrum Artifacts



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**.



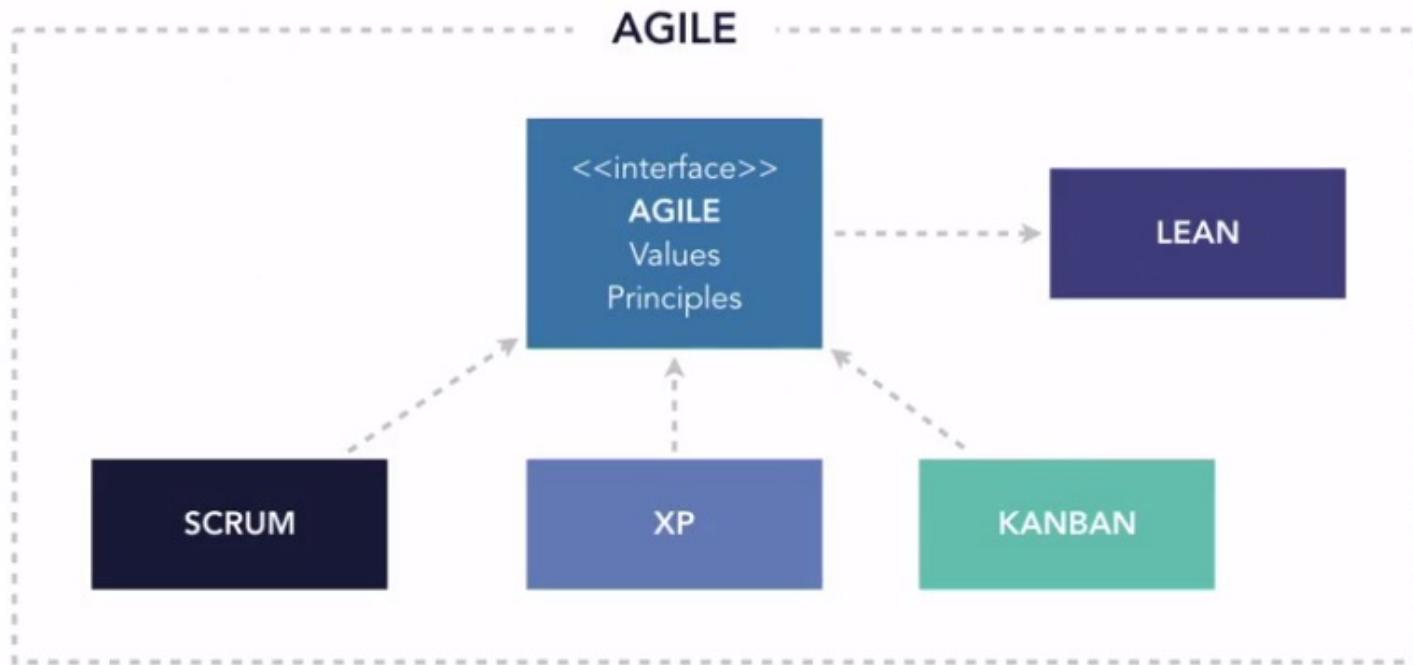
XP- Extreme Programming

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



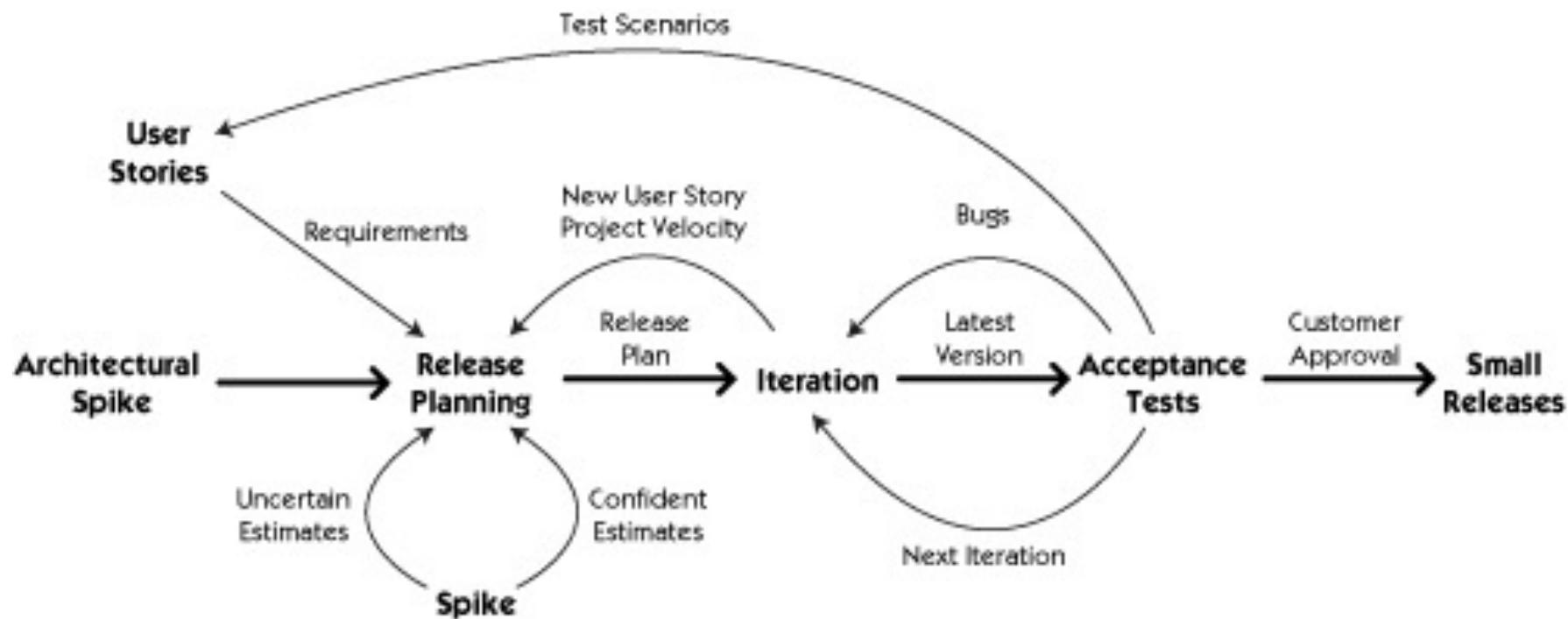
Source :LinkedIn Learning/XP overview

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 ...

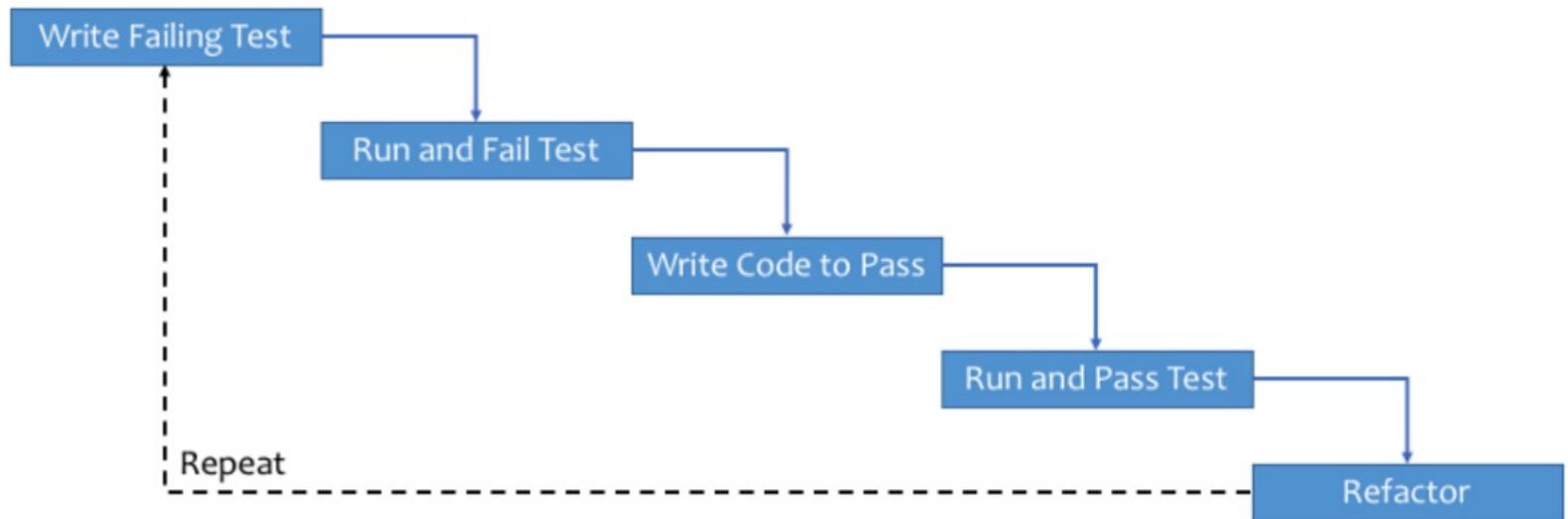
- **Forty-hour week (Energized work)**
 - Teams aim for sustainable phase
 - XP does not forbid overtime, but it has a clear rule – *You can't work a second week of overtime.*
- **On-site customer**
 - A real customer should sit with the team.
 - This person will be someone who will use the system, who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





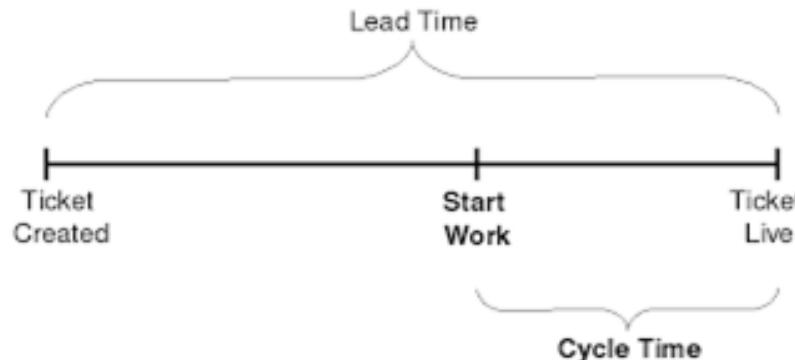
Lean Software Development

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

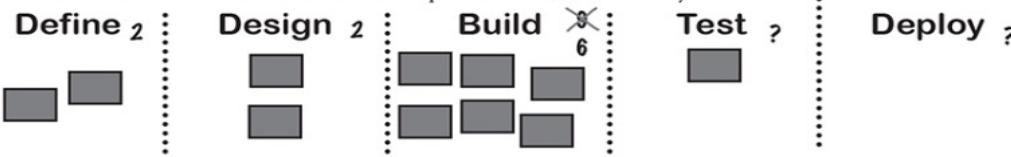
A value stream map is a great way to create this picture! These are the same boxes that you'd see at the top of the map.



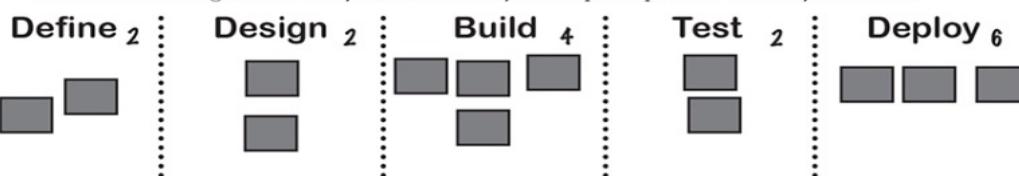
- 1 **Visualize Workflow:** create a picture of the process you're using today.



- 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.

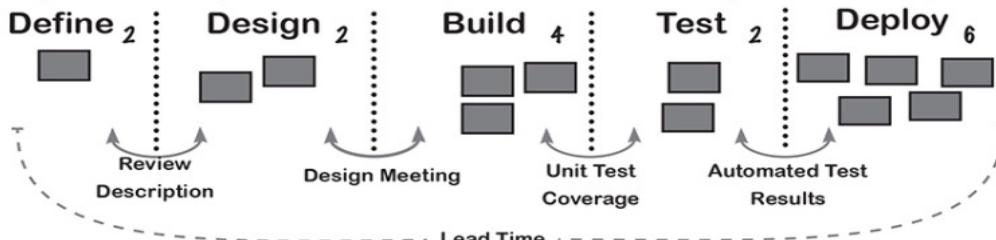


- 3 **Manage Flow:** measure the lead time and see which WIP limits give you the shortest time to delivering features to your clients. Try to keep the pace of delivery constant.



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

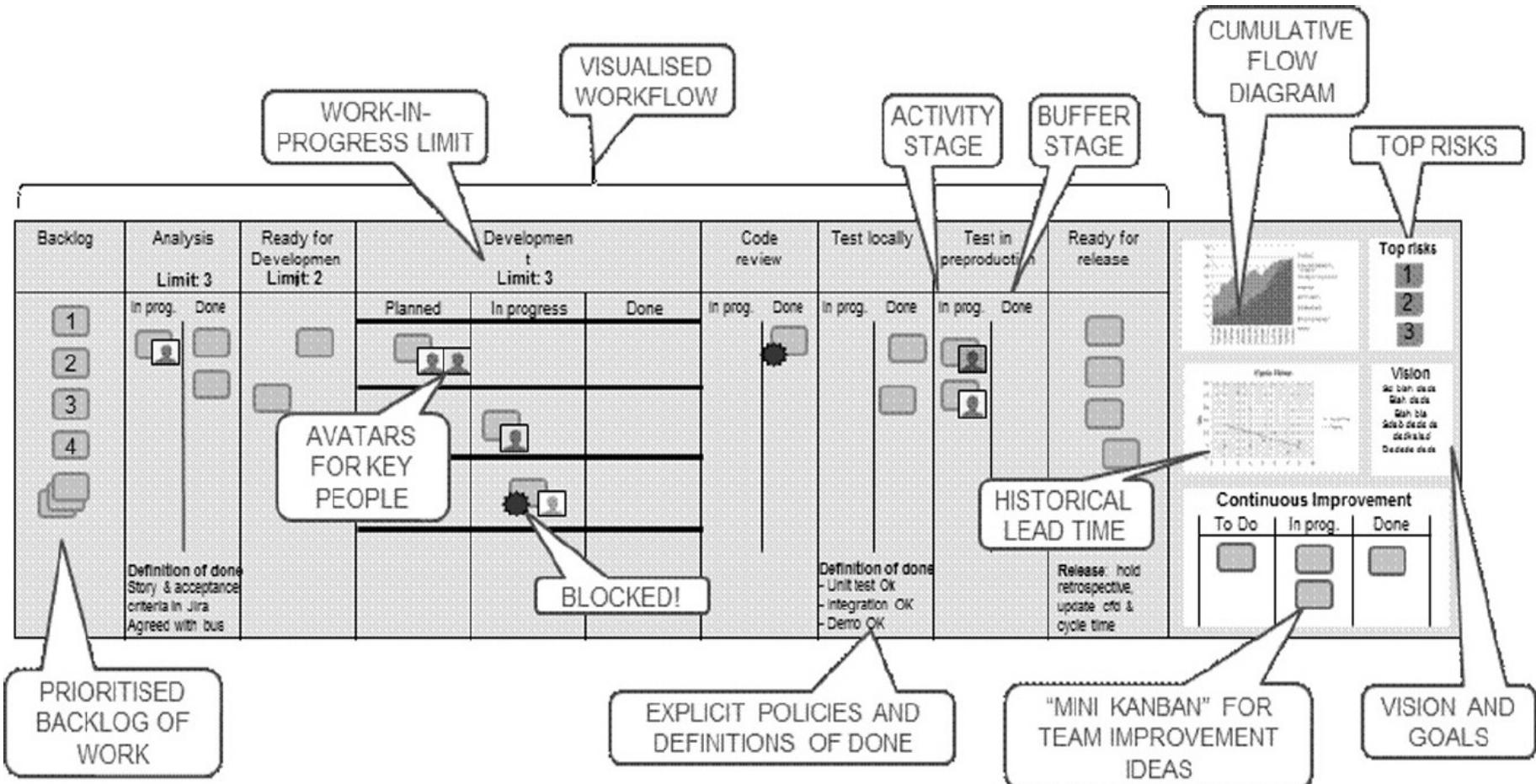
- 5 **Implement Feedback Loops:** for each step in the process create a check to make sure the process is working. Measure lead and cycle time to make sure the process isn't slowing down



- 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

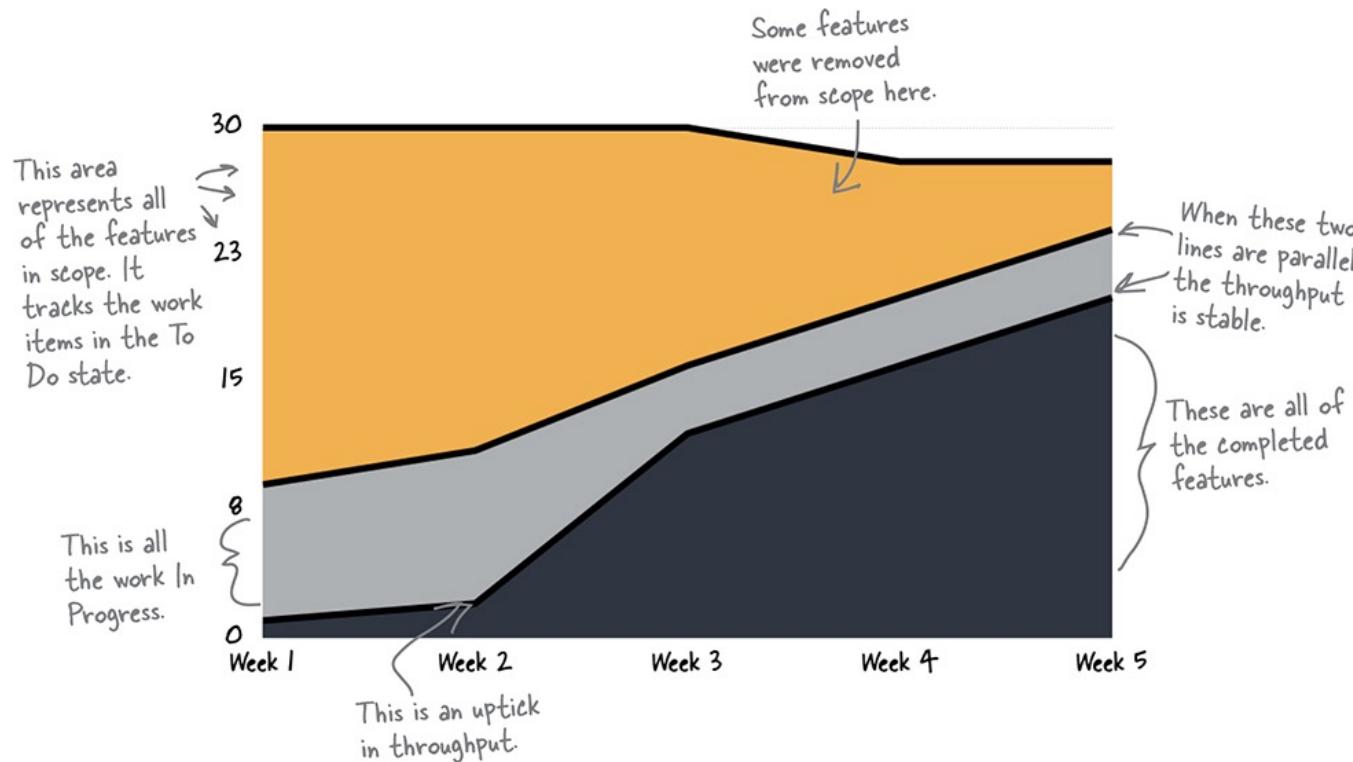
Typical Kanban board



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

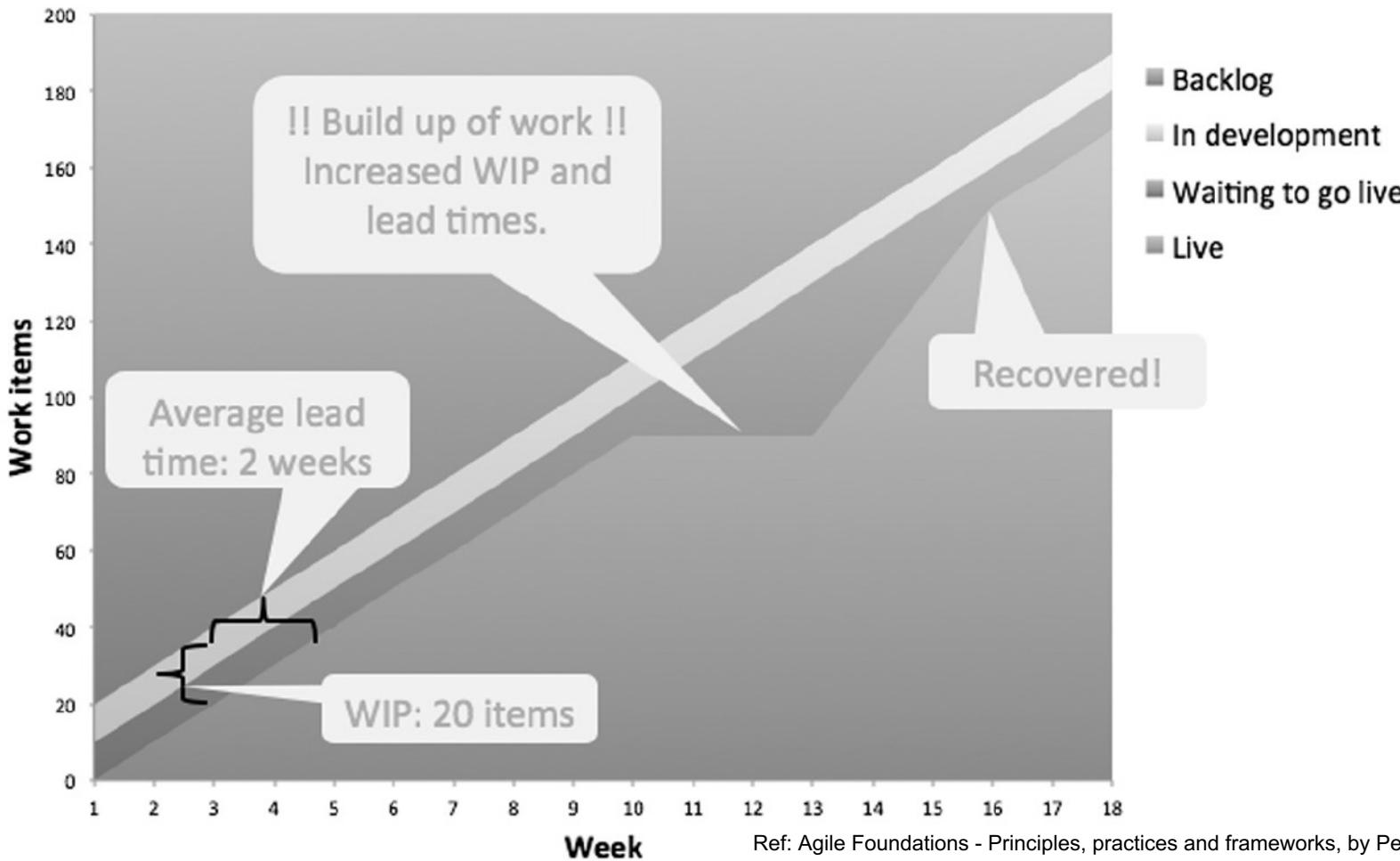
Cumulative flow diagram (Example-1)

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



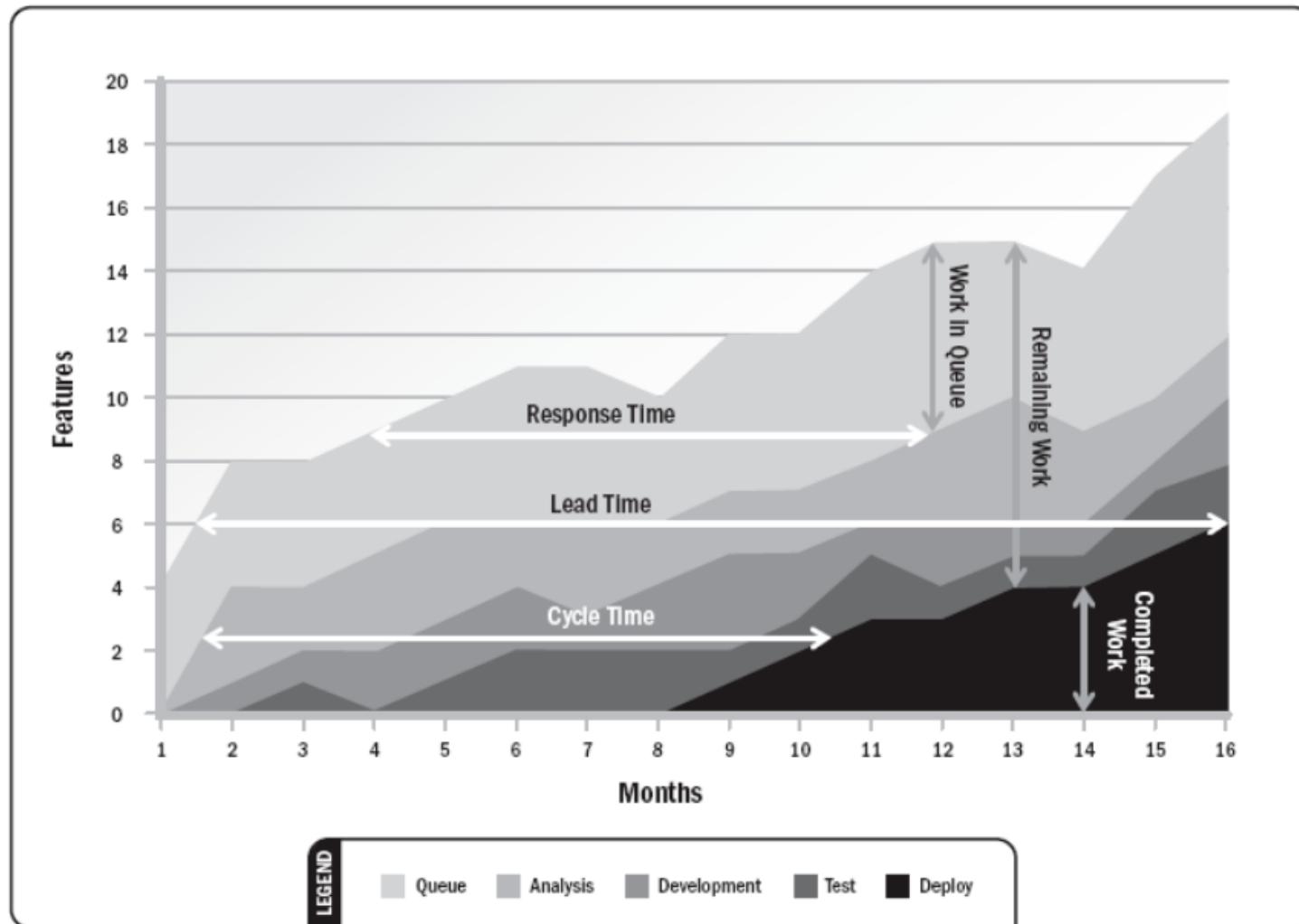
Ref: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

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)





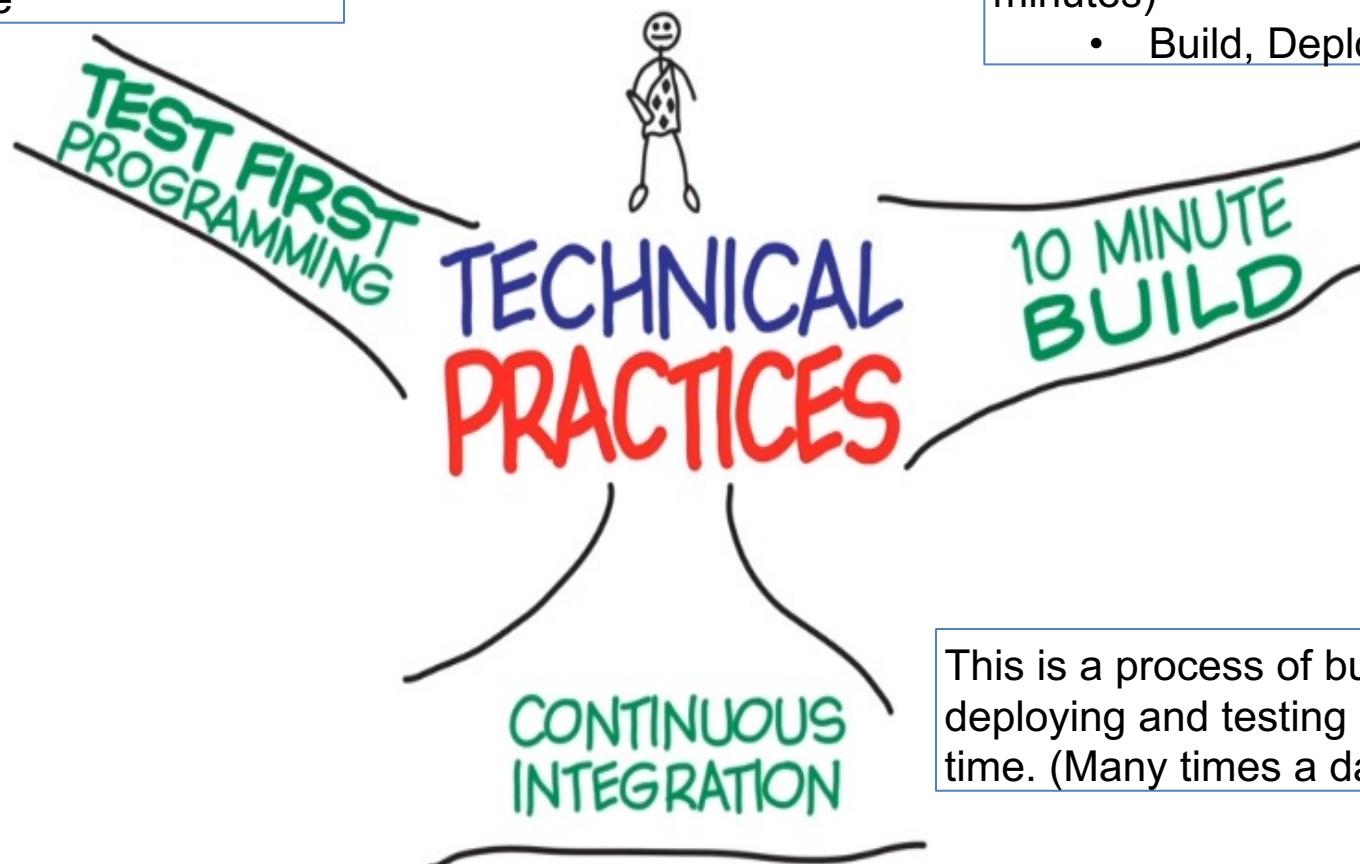
Module-4 XP-Technical Practices

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 presentation

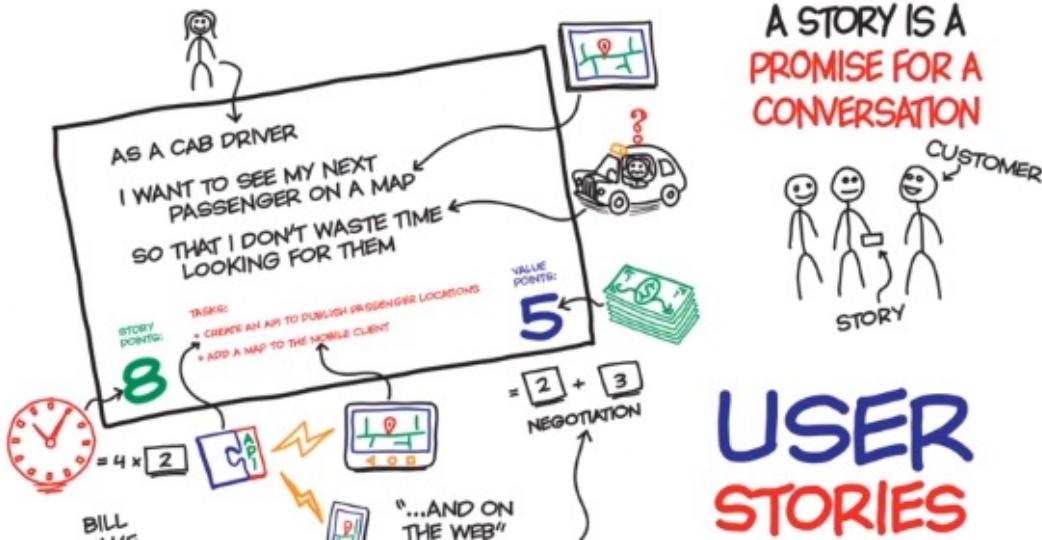
BITS Pilani
Pilani Campus

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

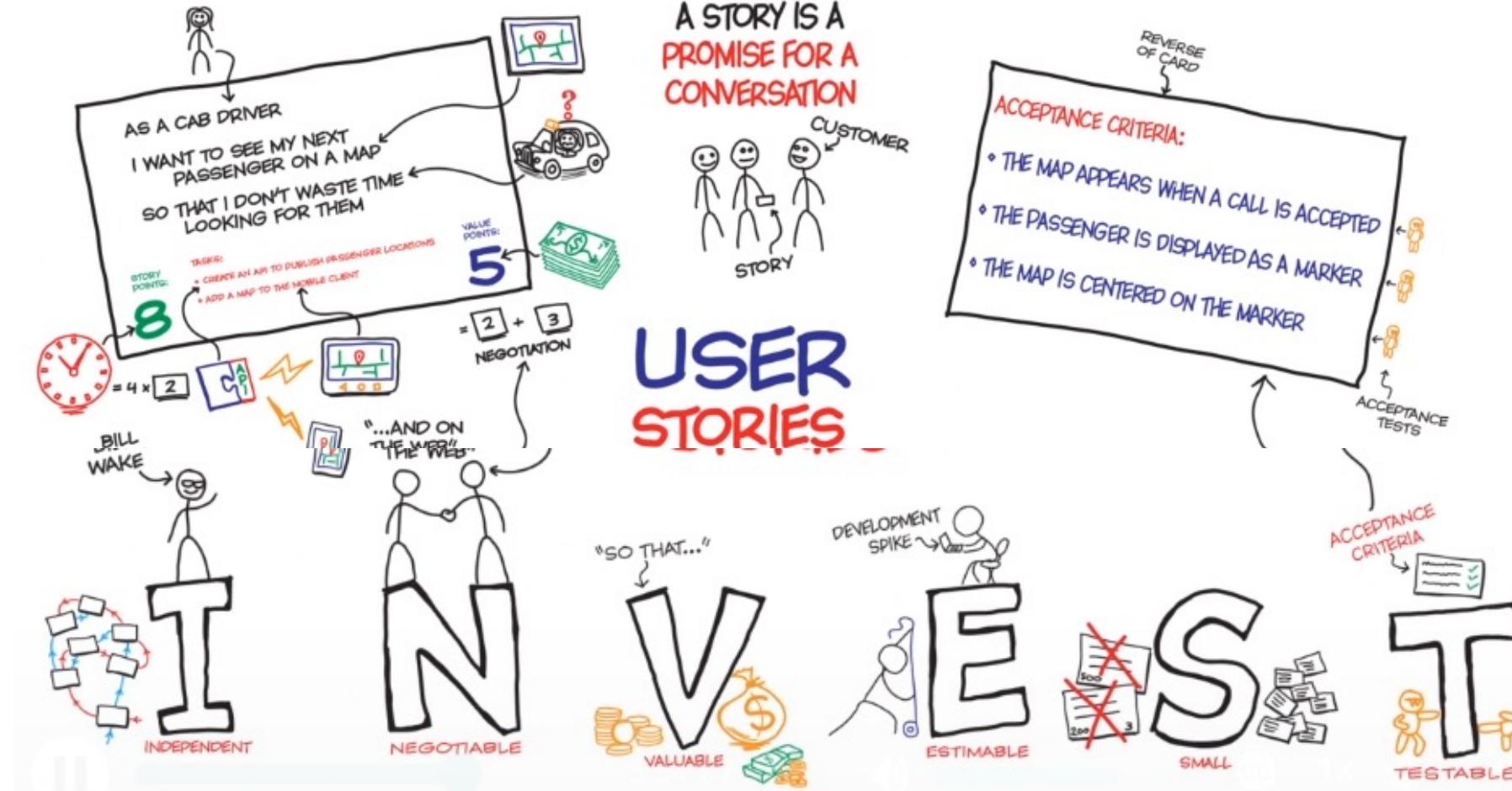


Module-5 Agile Requirements & Agile Estimation

Agile Requirements-1 (User Stories)

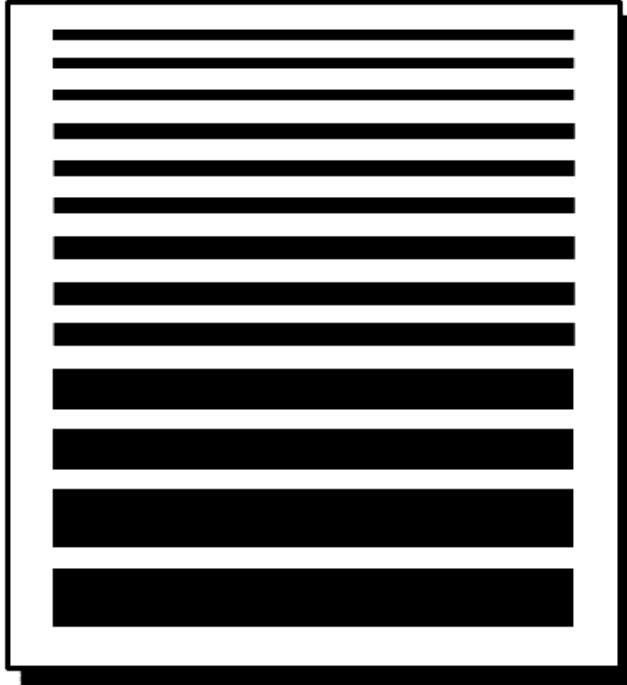


Agile Requirements-2 (INVEST)



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)

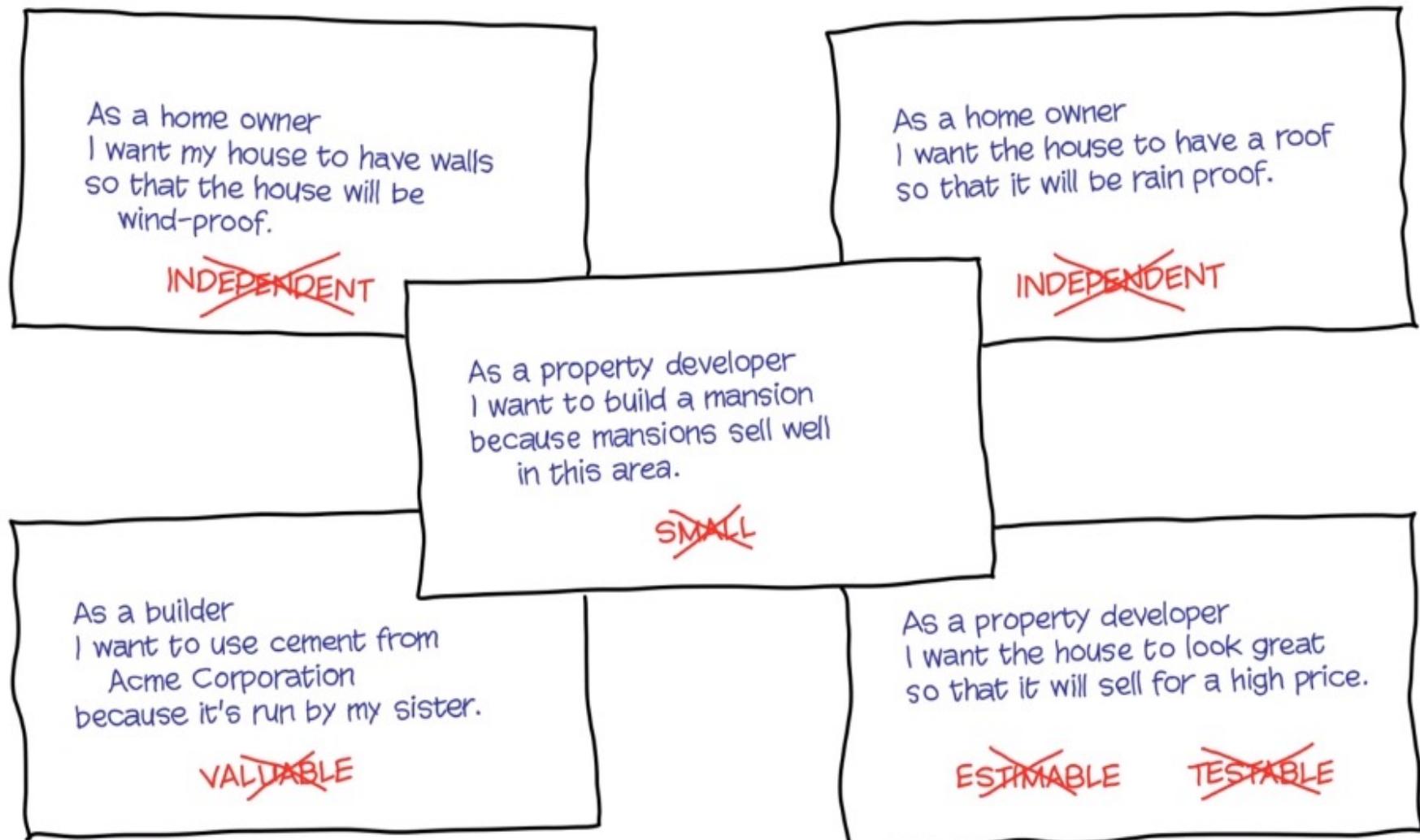


(Real Estate Project - House Construction)

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



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

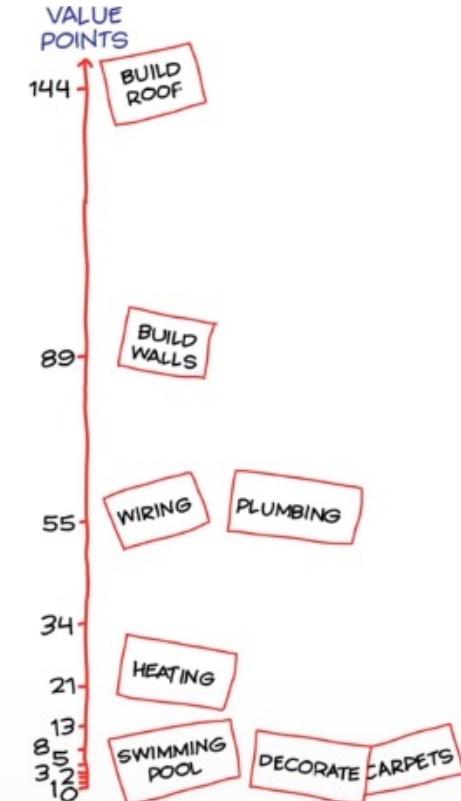
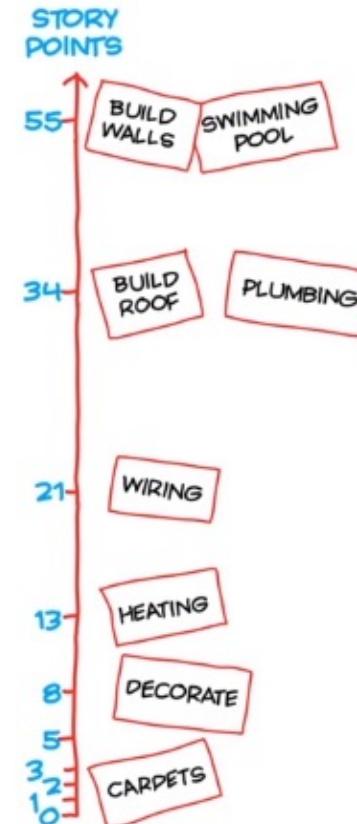
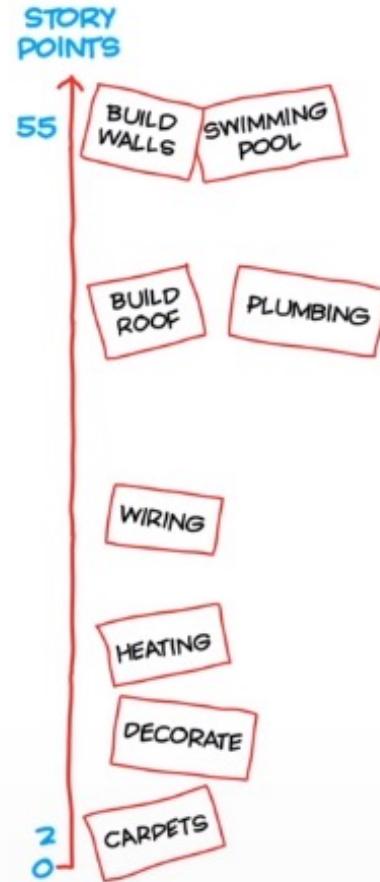
- ✓ Relative Estimation
- ✓ Fibonacci Sequence (1,2,3,5,8,13,21,34,55,)
- ✓ Story Point
- ✓ Value Point
- ✓ Velocity
- ✓ Average Velocity
- ✓ BFTB (Bank For The Buck)= Value Point /StoryPoint

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

↳ ↲

This story needs to go first

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

VELOCITY: 55
VALUE: 89
ITERATION 1

BUILD WALLS

Cost: \$20000,
Cost/SP \$225

VELOCITY: 55
VALUE: 199
ITERATION 2

BUILD ROOF
WIRING

Cost:
 $199*225=\$44000$

VELOCITY: 47
VALUE: 76
ITERATION 3

HEATING
PLUMBING

Cost: $76*225$
=\$17000

VELOCITY: 10
VALUE: 4
ITERATION 4

CARPETS
DECORATE

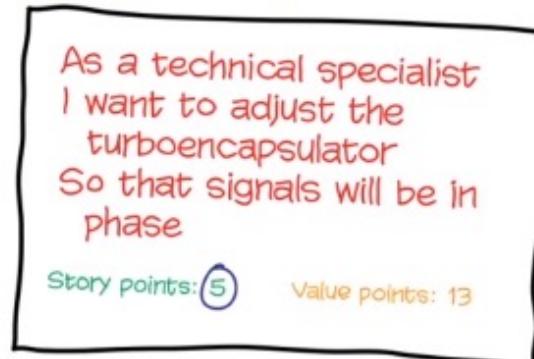
Cost:
 $4*225=900$

Iteration-5
Swimming Pool

Estimation Exercise

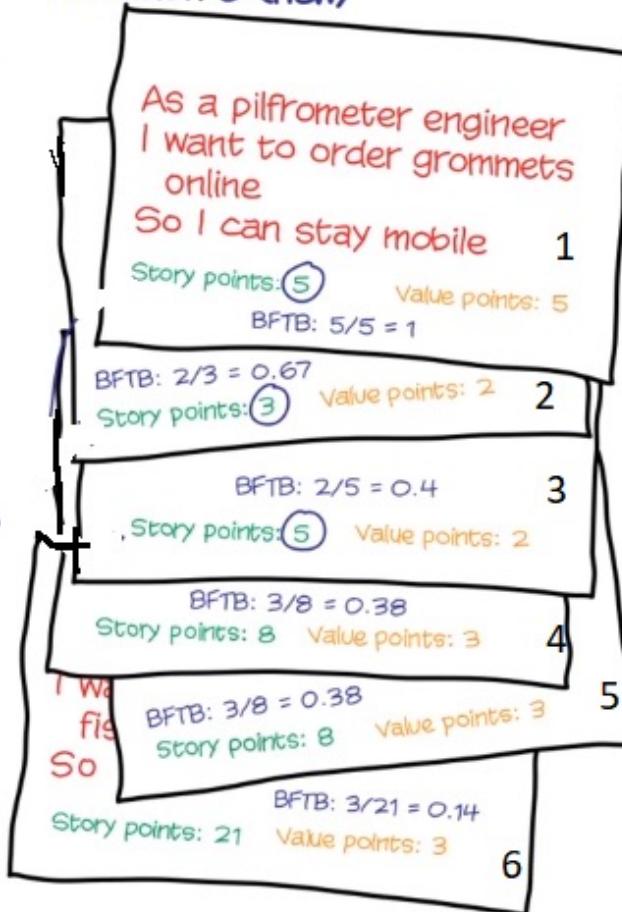
(Iteration Length = 2 weeks (10 Days, 8hrs Per day)

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?

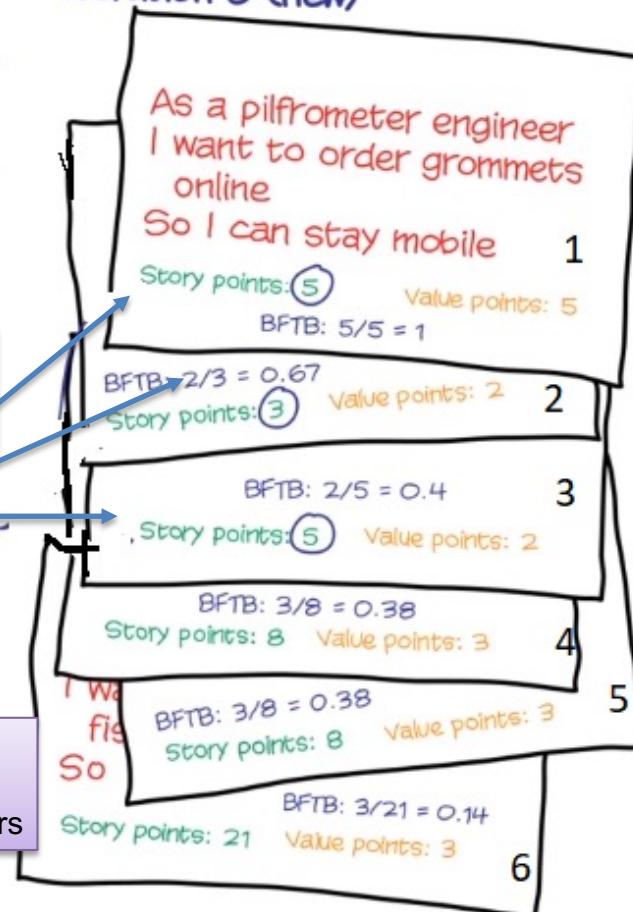
80 hrs = 13 Story Points
1 Story Point = $80/13$
= 6.15hrs

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 = $50*6.5\text{hrs}=307.5\text{ hrs}$

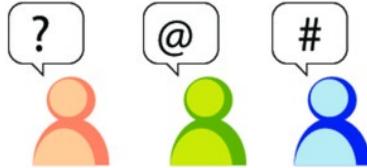
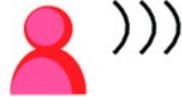
Iteration 8 (new)



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

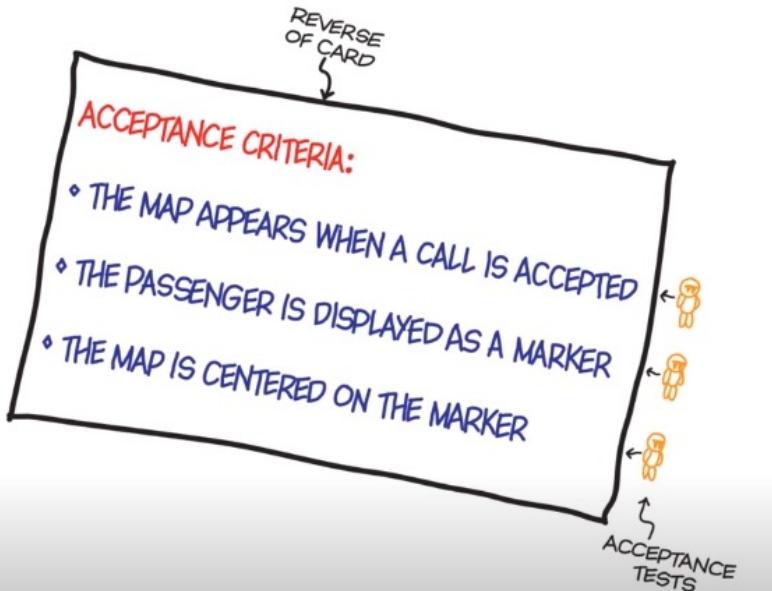
Acceptance Criteria/Conditions of Satisfaction



- 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:

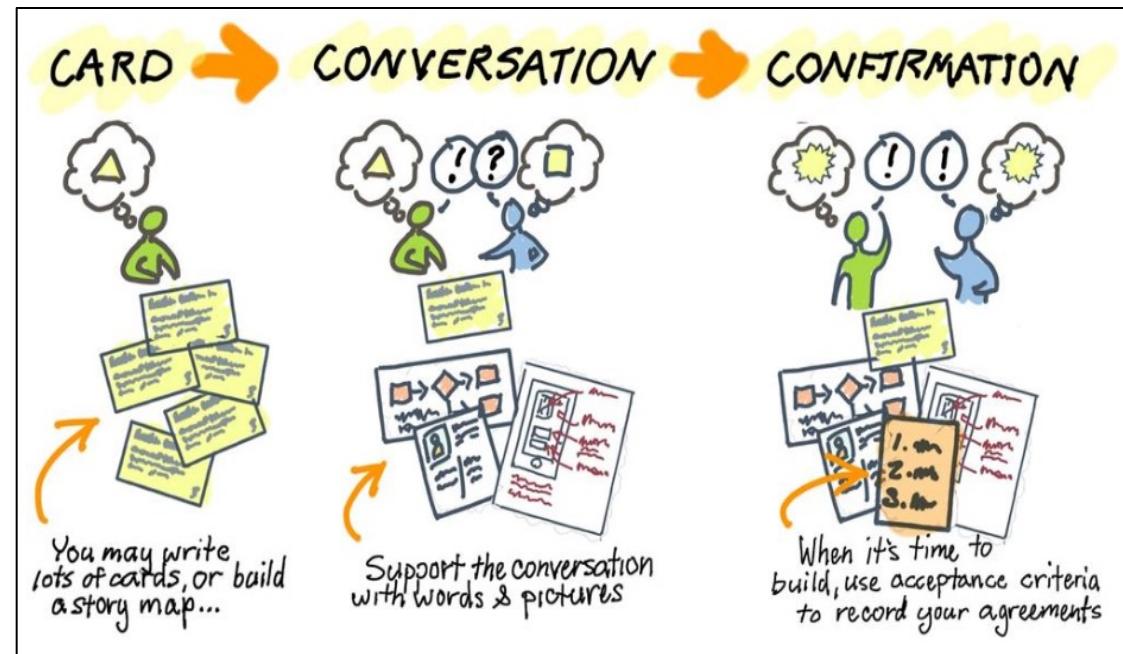
- Write what you'd like to see in the software on a bunch of index cards.

• 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

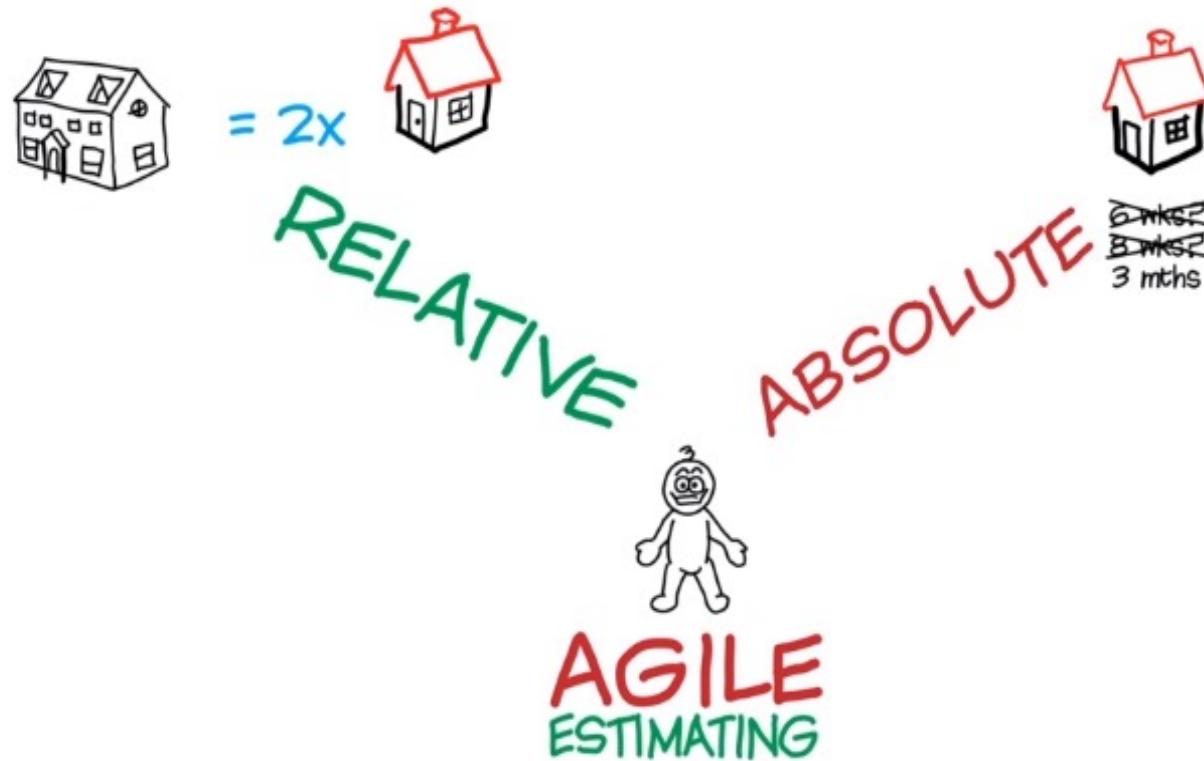
Absolute Estimation

- 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.

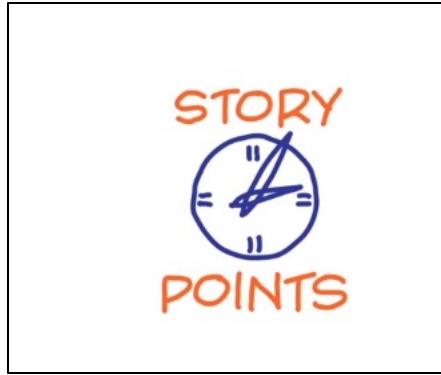


Story Point Estimation

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

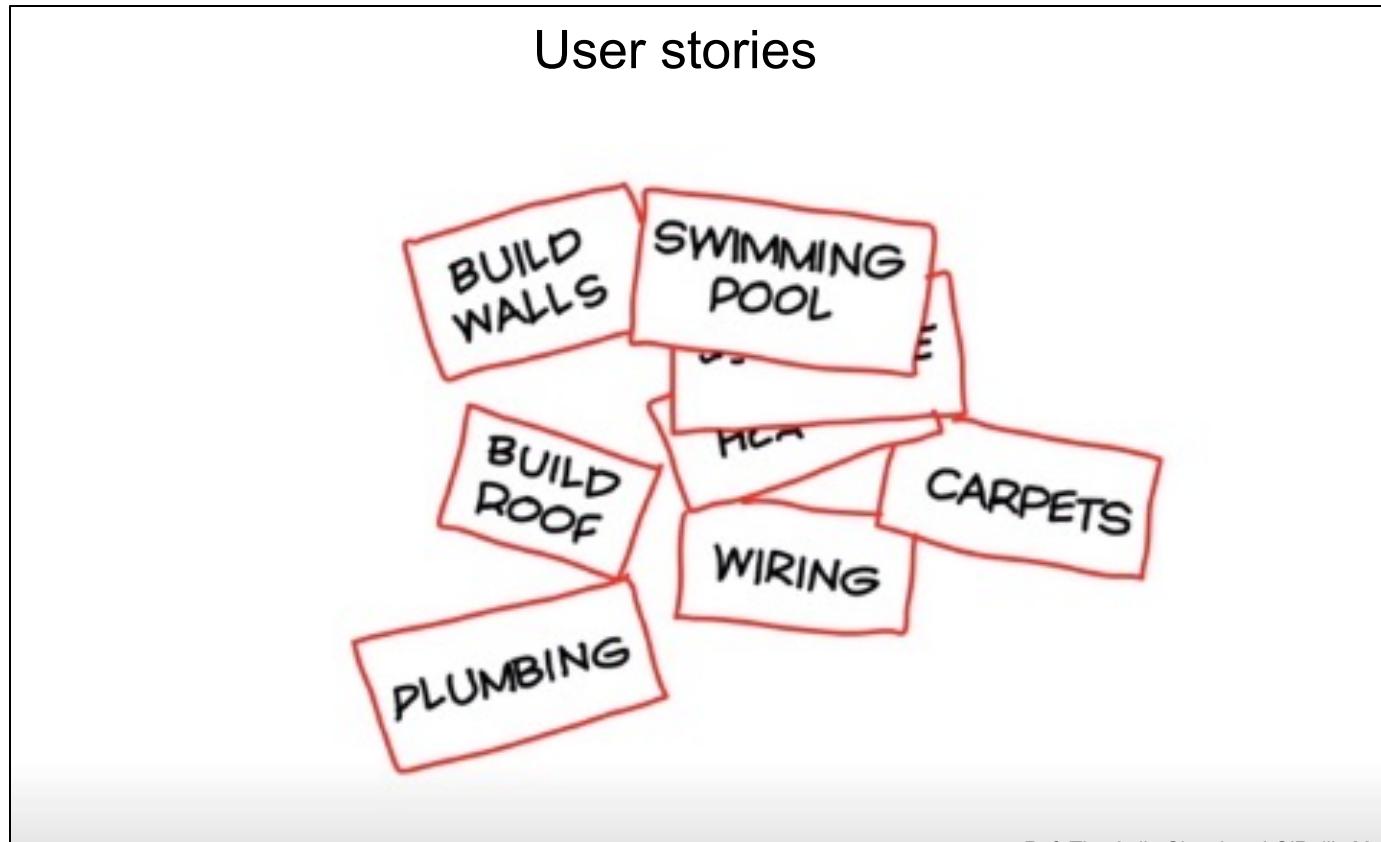
Example: Migrate application from an outdated data store to new DB technology and ORM

How User Stories are estimated by the team?

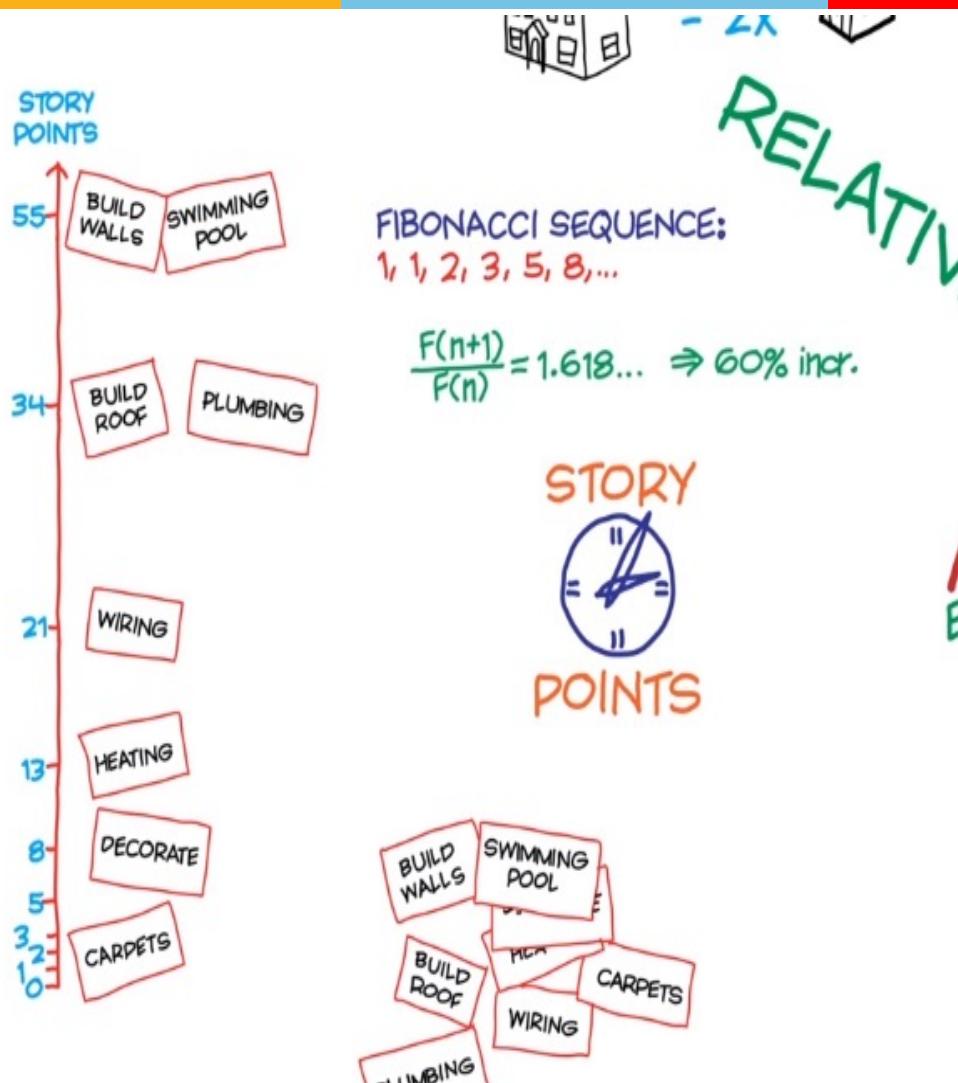
- One method is to play **Planning poker** game.
 - Planning poker helps give everyone a voice.
 - Combining of individual estimates through group discussion leads to better estimates
 - Combat Groupthink-meaning, the way that people tend to agree with the most popular idea.
- Planning poker is a game 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....

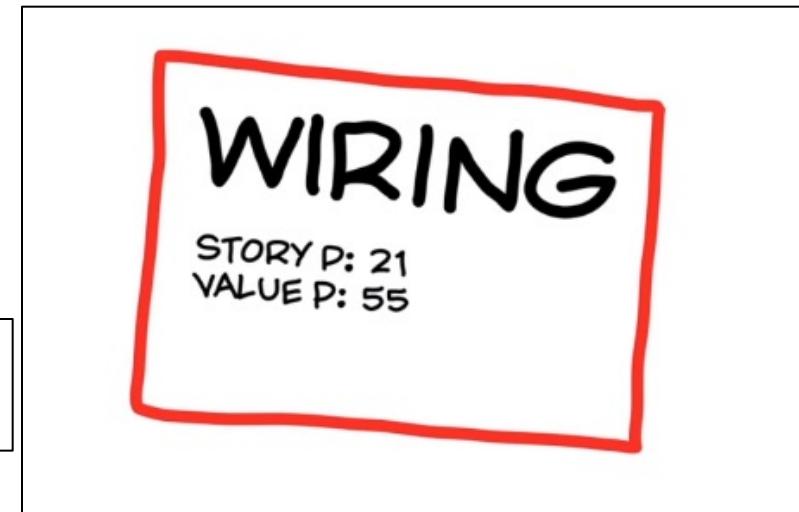
At this point, We do not know the effort of each story

Value Point

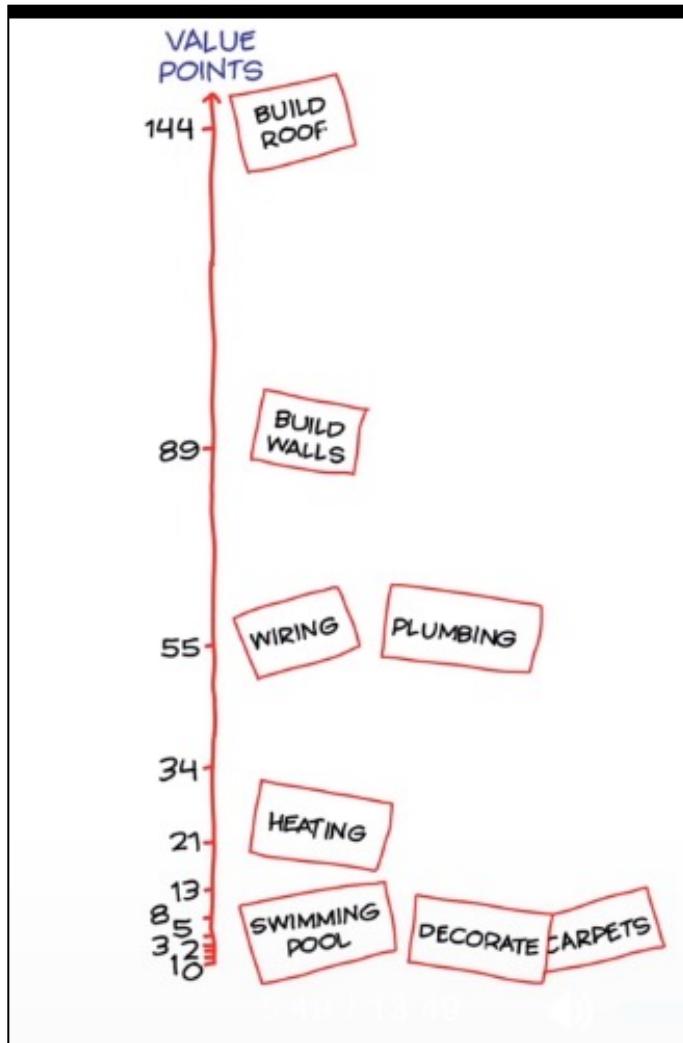
- A user Story has two estimates.
 - Story point – estimate of time
 - Value Point – estimate of value
- Developers , the people who do the work, estimate User Stories in Story points,
- Customer / Product owner estimates User Stories in Value Point, in the same way.

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

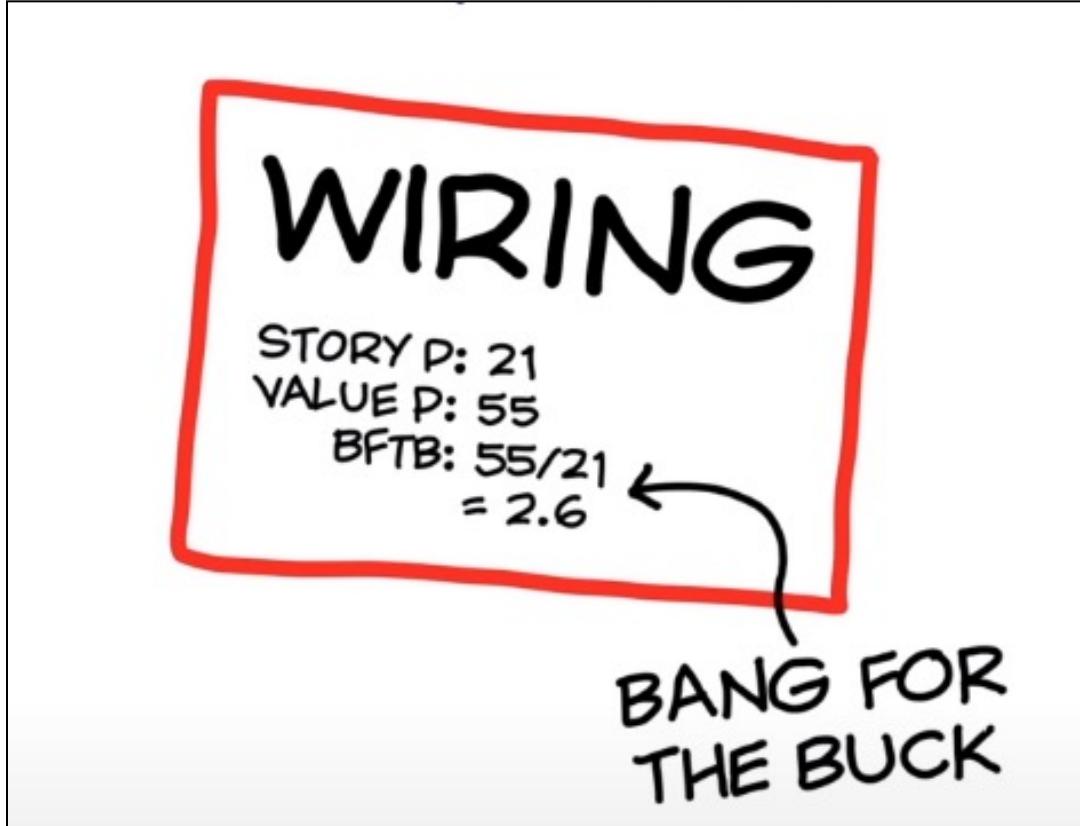


Highest Valued Stories at the top of Product backlog

Relatively Valued Stories

Lowest Valued Stories at the bottom

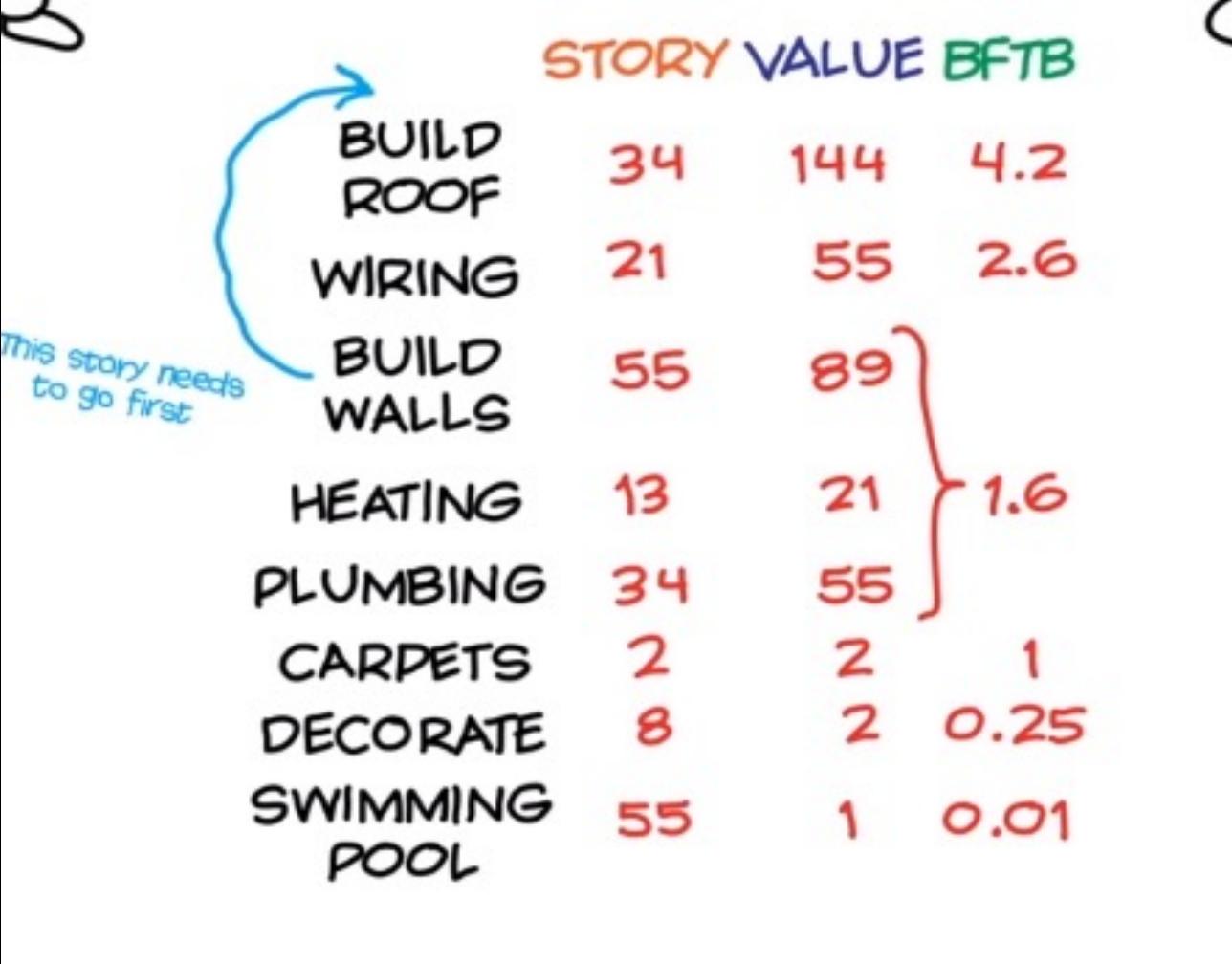
Bang For the Buck (BFTB) (OR) Priority



BFTB = Value Point divided by Story Point for each story

How stories are prioritized for each Iteration – by BFTB

Product Backlog

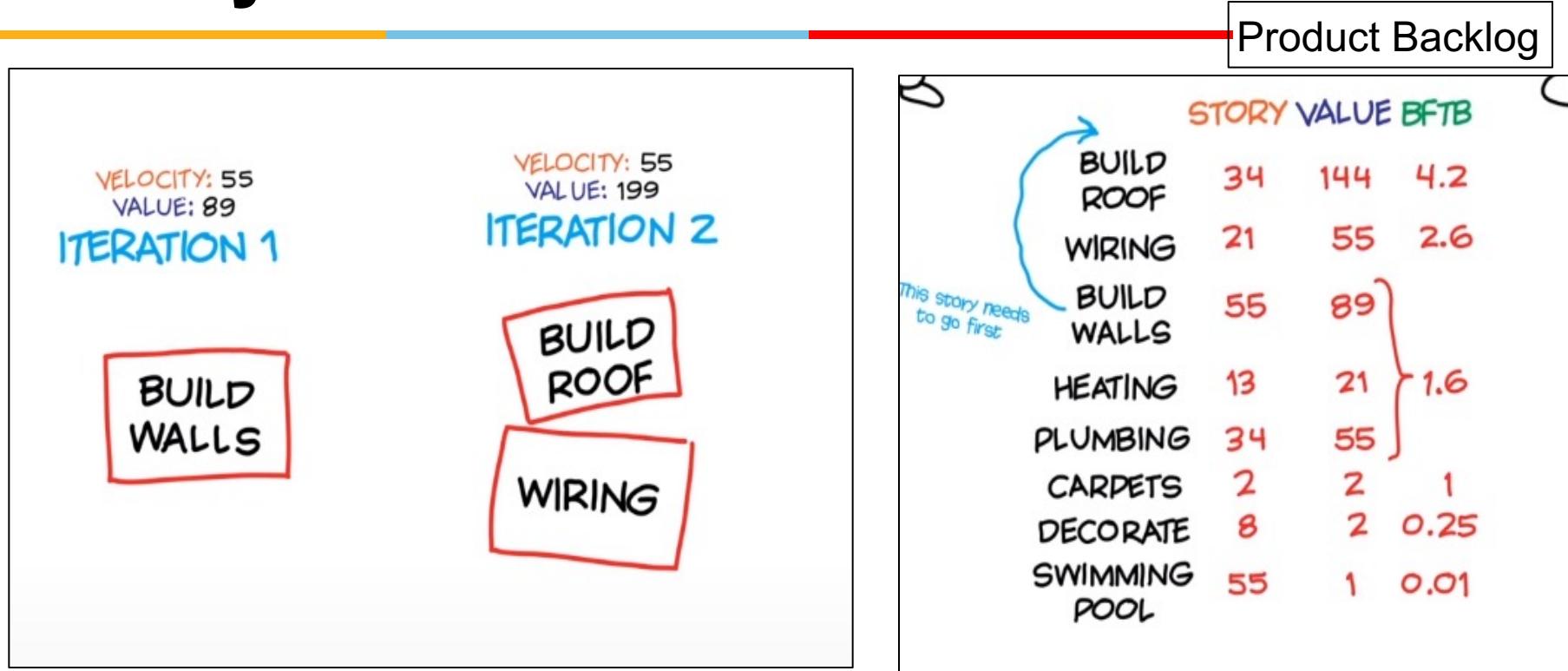


STORY VALUE BFTB

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

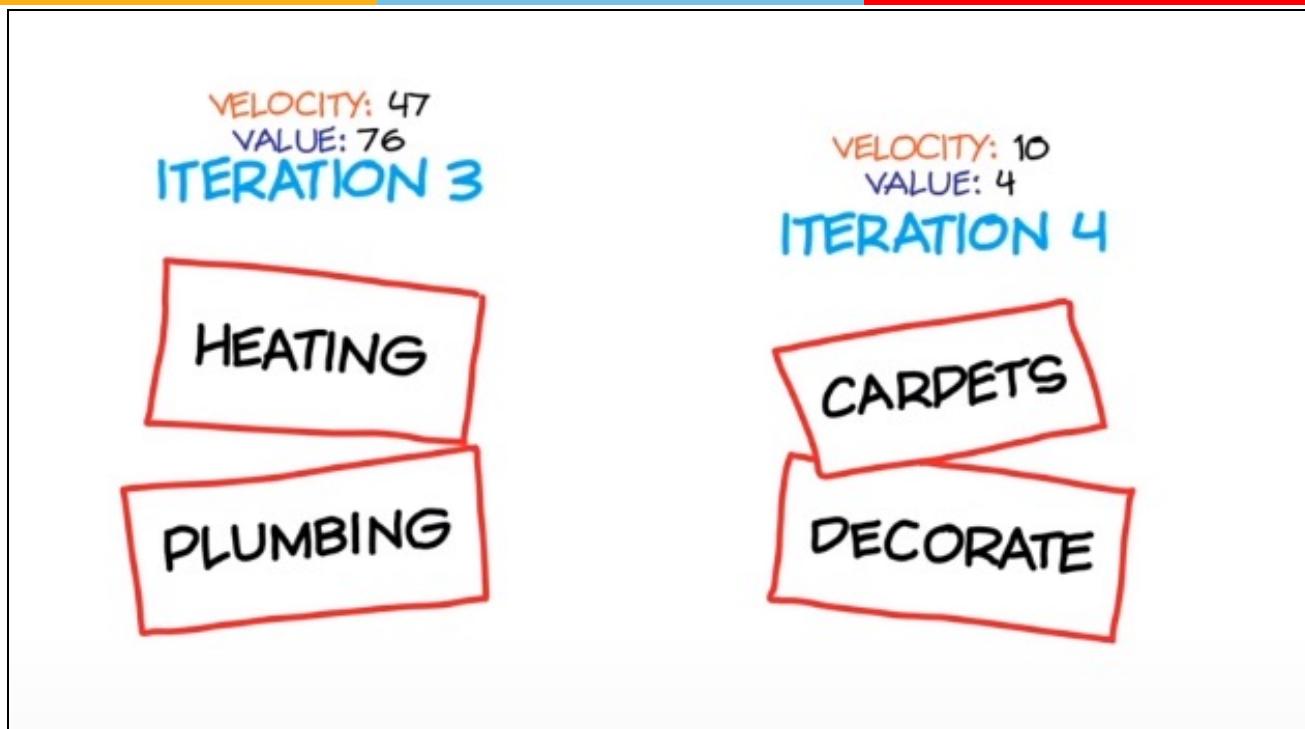
This story needs to go first

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

W

fis

So

BFTB: $3/8 = 0.38$

Story points: 8

Value points: 3

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

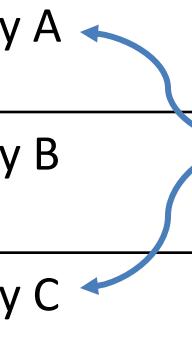
Estimate by Analogy

- 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

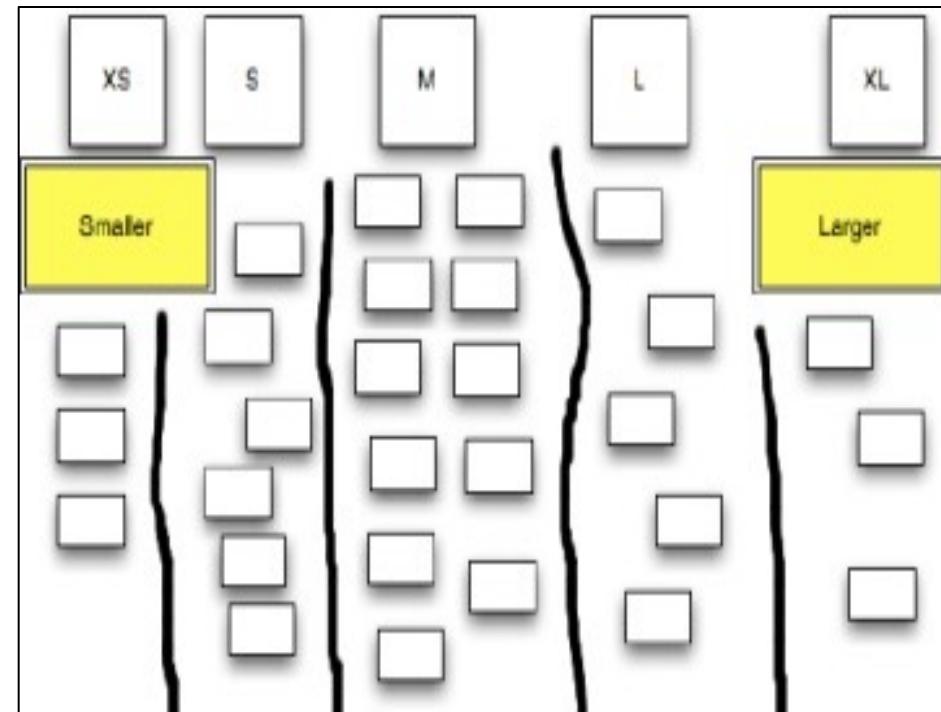
- **Level of Effort (LOE) or T-Shirt Sizing**
 - T-shirt size,” “level of effort” (LOE), or “small, medium, large.” (Easy, but lack precision, inability to add up several stories into a meaningful measure.)

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

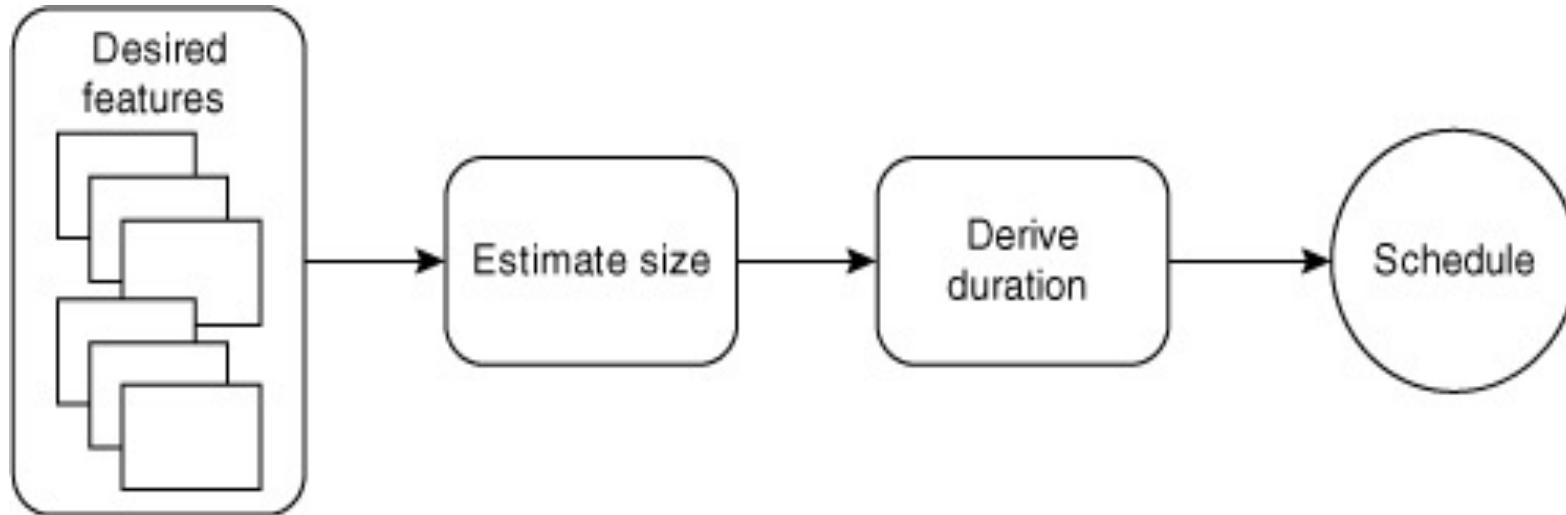
- **Disaggregation**
 - Breaking a big story into smaller stories ,we know how long the smaller stories take, So, disaggregating to something we know lets us estimate something bigger we don't know

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

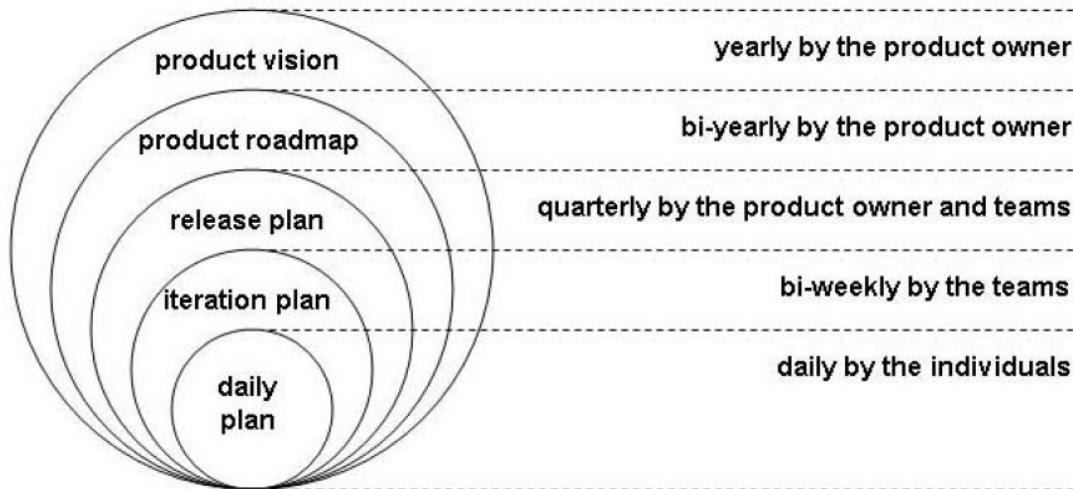
MANAGING ORGANIZATIONS

Bring Agile Planning to the Whole Organization

by Jeff Gothelf

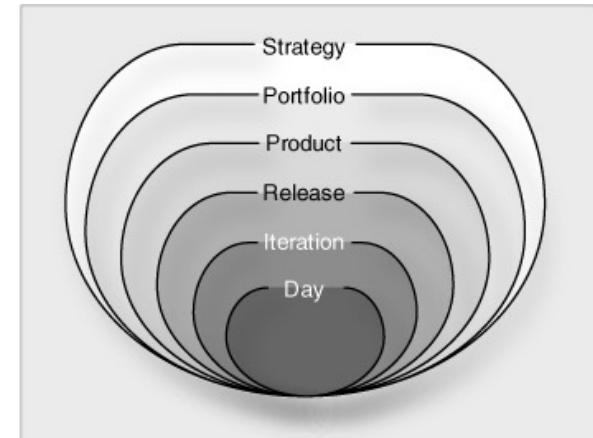
<https://hbr.org/webinar/2015/05/bring-agile-planning-to-the-whole-organization>

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)

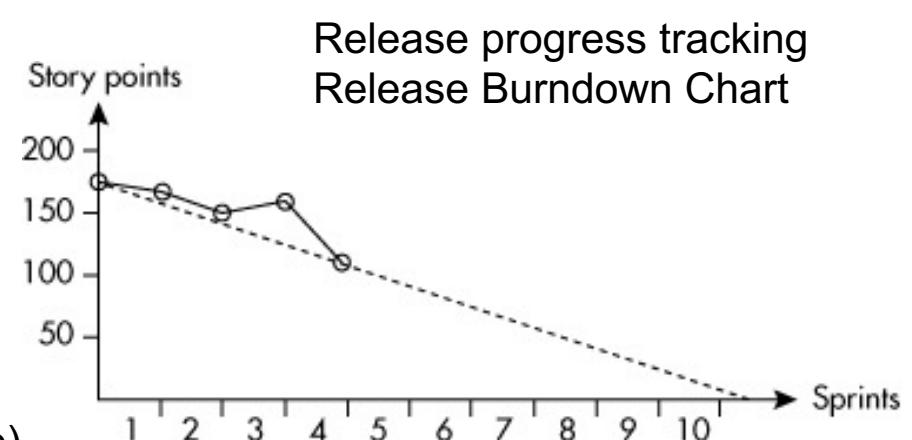
Release Planning



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)

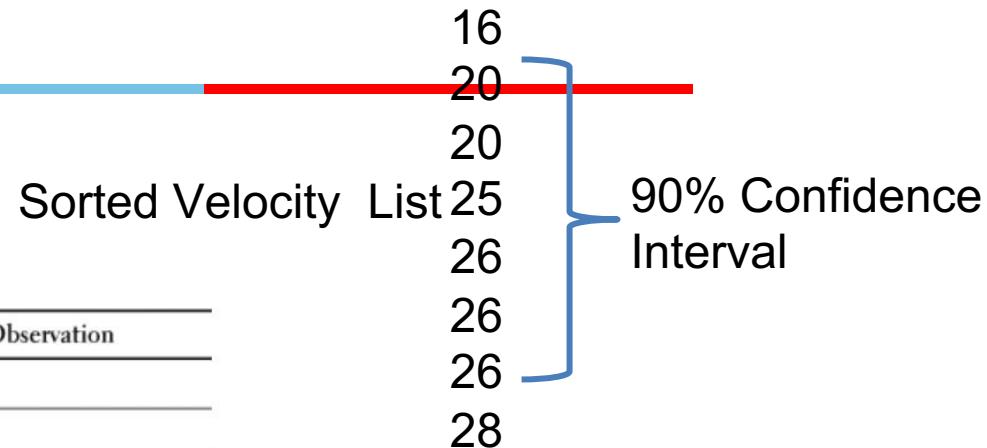
Estimating Velocity

- 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?

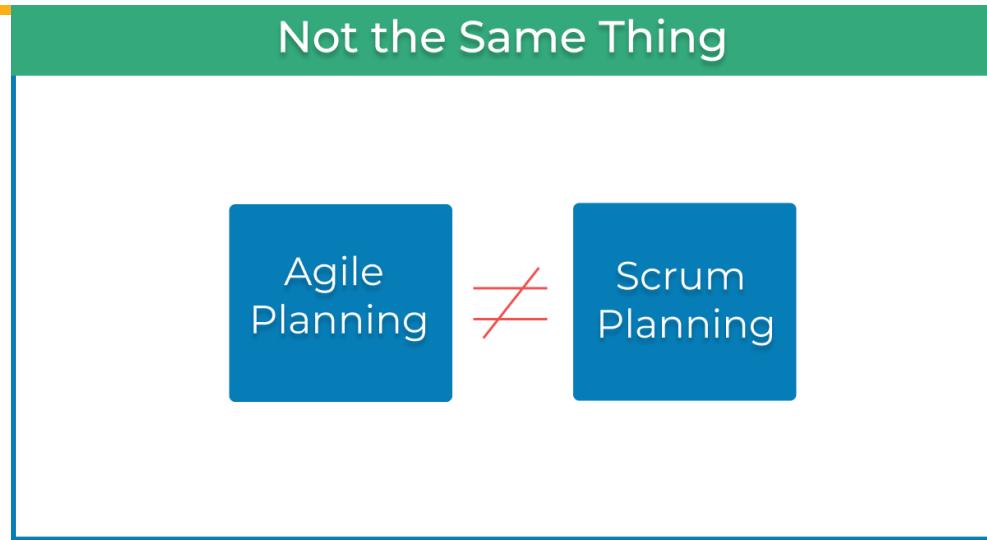
- Available budget is \$140000, Each Iteration cost = \$20000; Only 7 Iterations is possible.
- Max and Min Schedule is same = D0 +14 Weeks
- Velocity High =20; Number of Iteration Required = $140/20 = 7$
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number 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

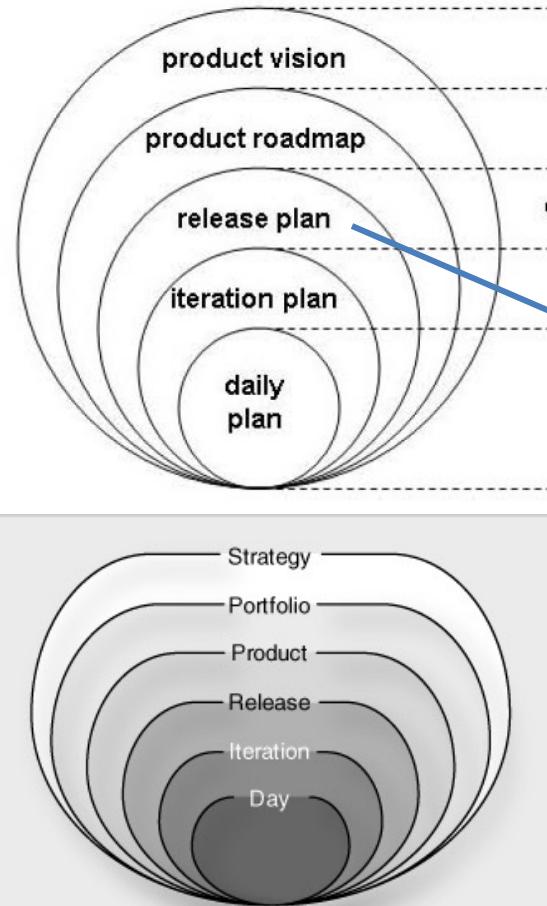
Agile Planning



- 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)



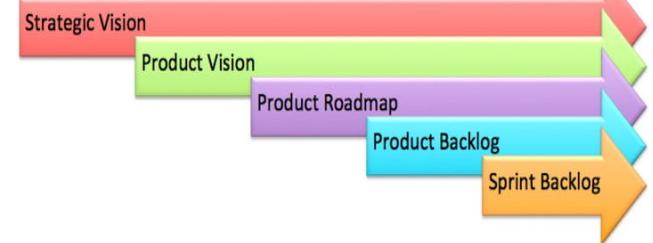
Ref: 5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up by Hubert Smits

10/9/22

SE ZG544 S1-22 Agile SW Process

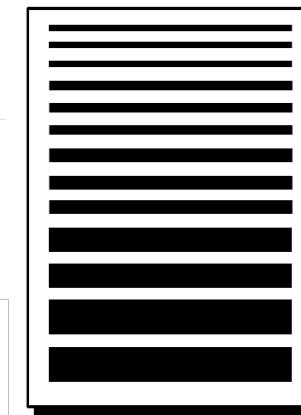
14

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

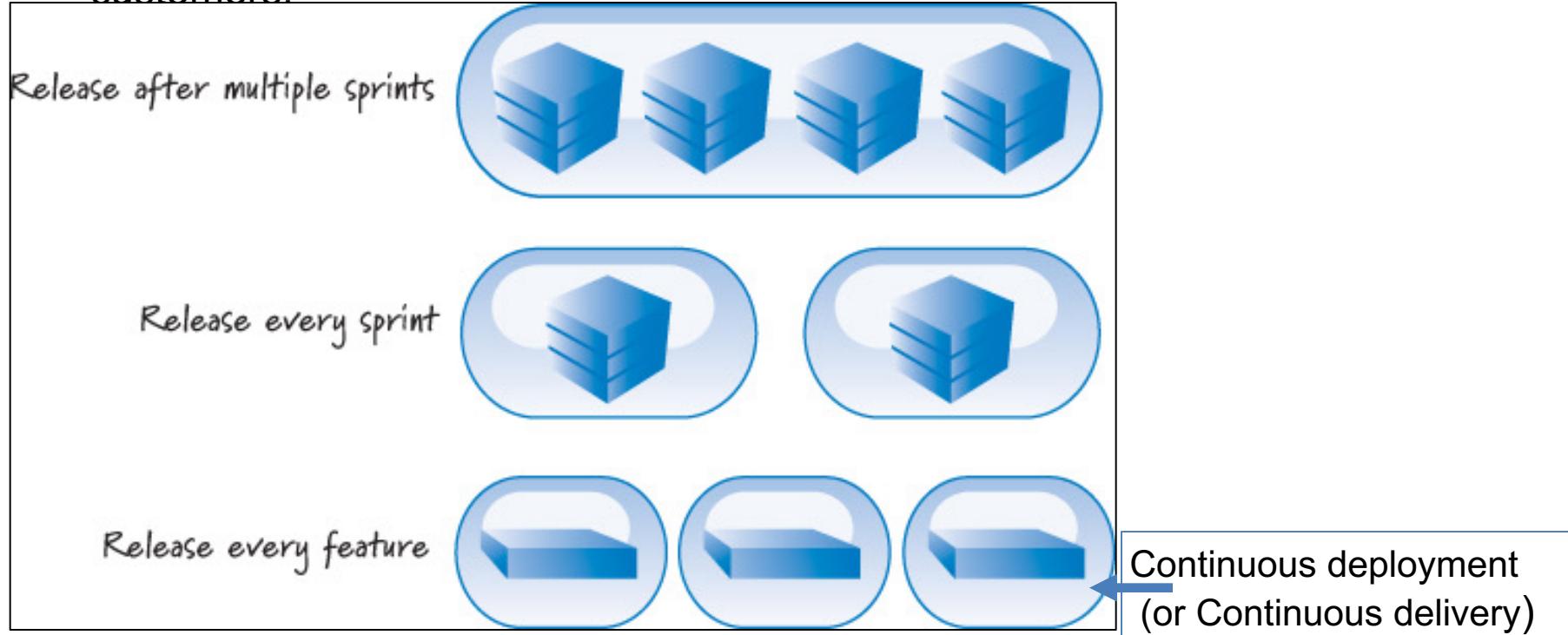
5-Levels of Agile Planning (Product Planning)



- 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

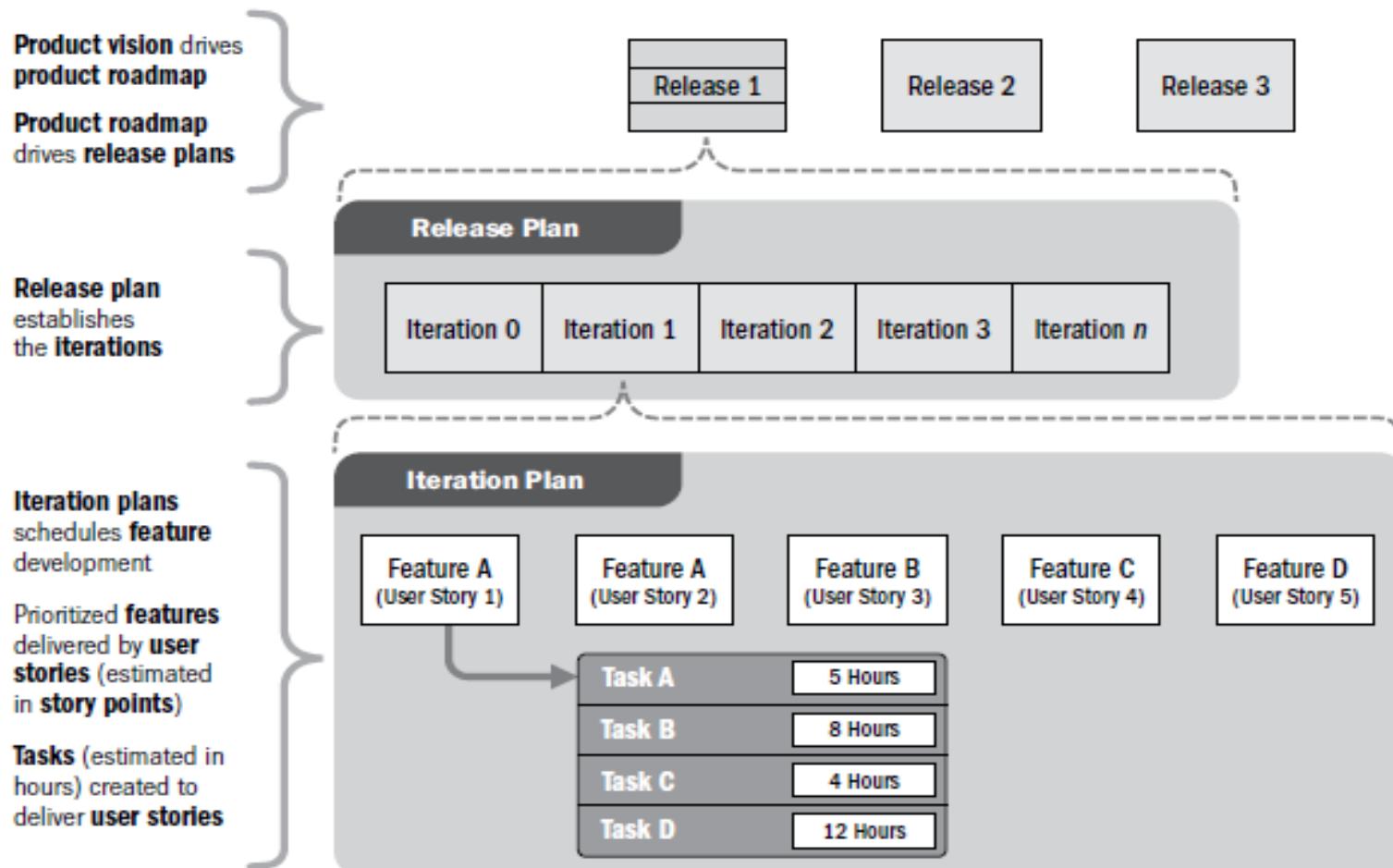
- Release planning is best done as a collaborative effort by involving the stakeholders and the development team



- 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.

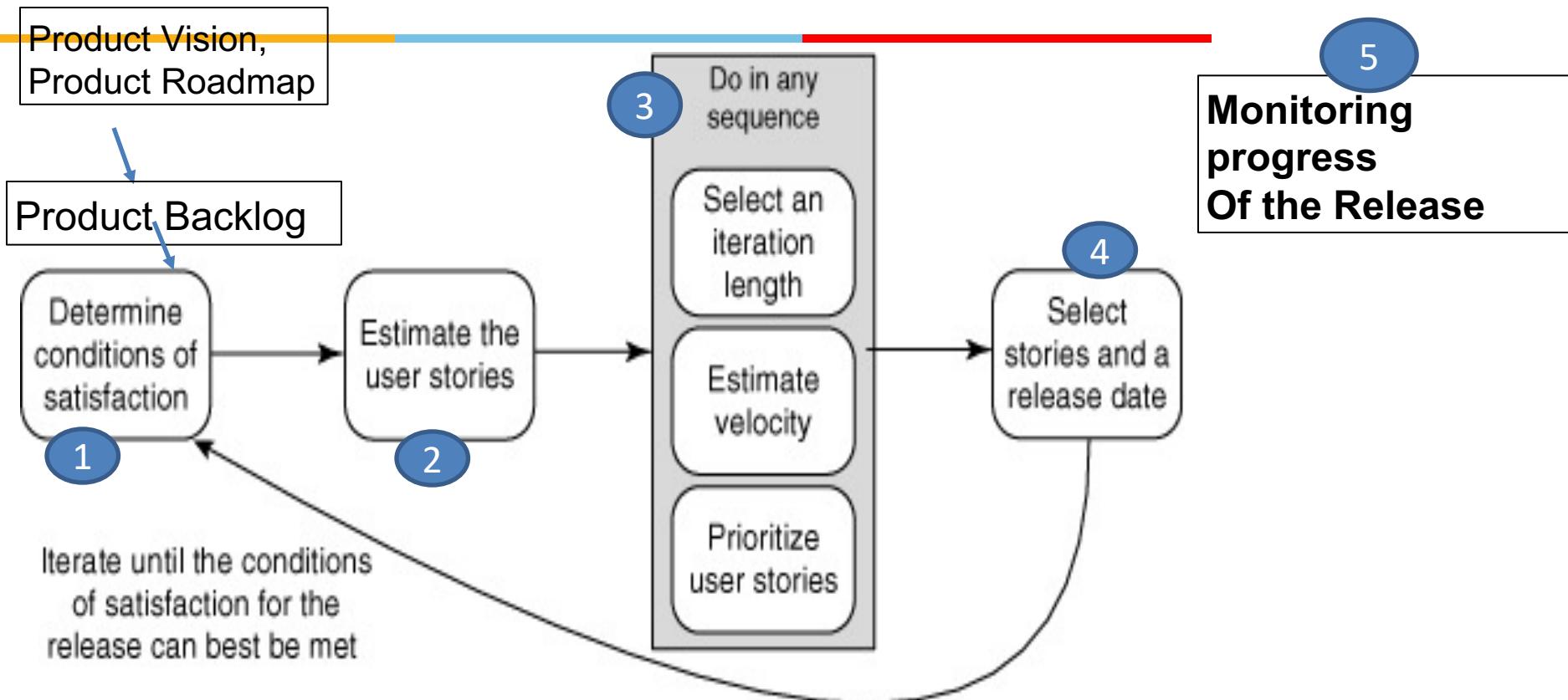
Source: <https://www.romanpichler.com/blog/release-planning-advice/>

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.

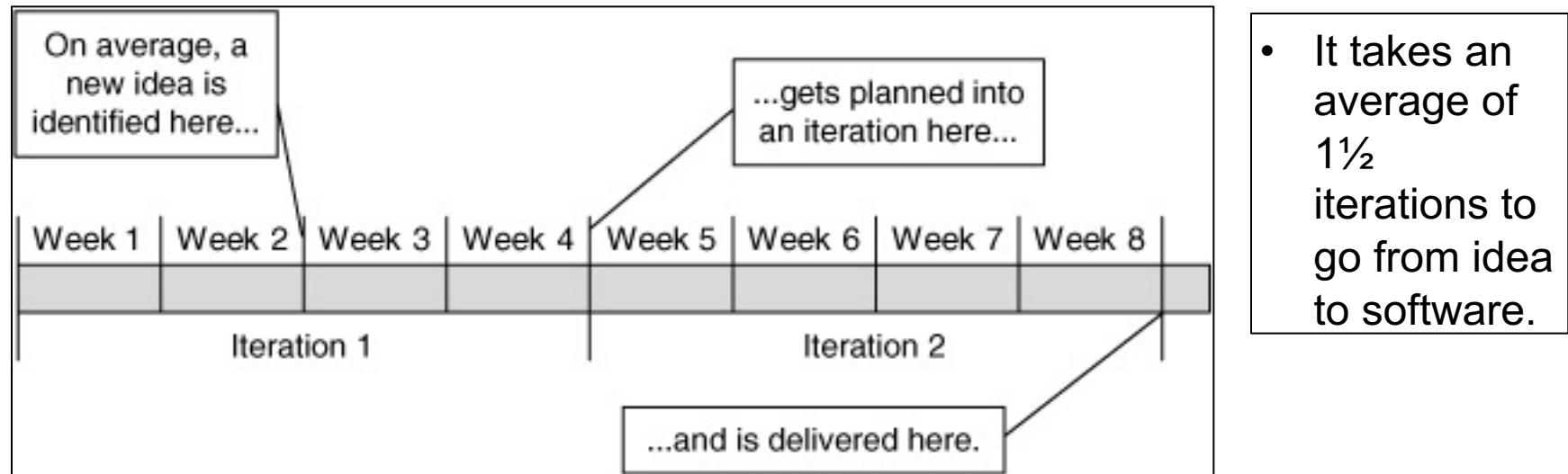
- User need
- Technical aspects
- Team Velocity
- The more uncertainty of any type there is, the shorter the iterations should be.
 - Shorter iterations allow more frequent opportunities for the team to measure its progress through its velocity and more opportunities to get feedback from stakeholders,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

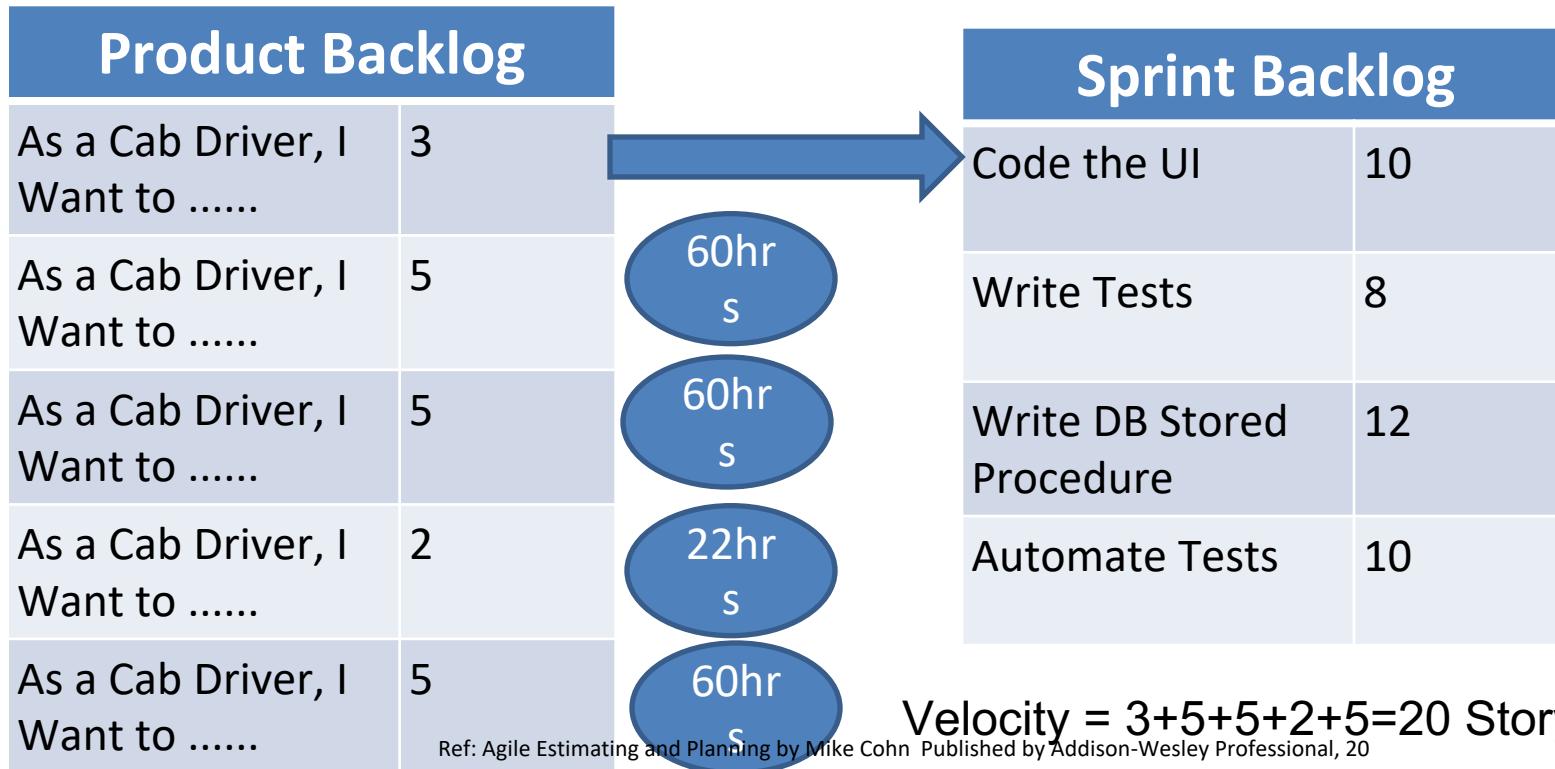
Ref: Agile Estimating and Planning by Mike Cohn Published by Addison-Wesley Professional, 20

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					

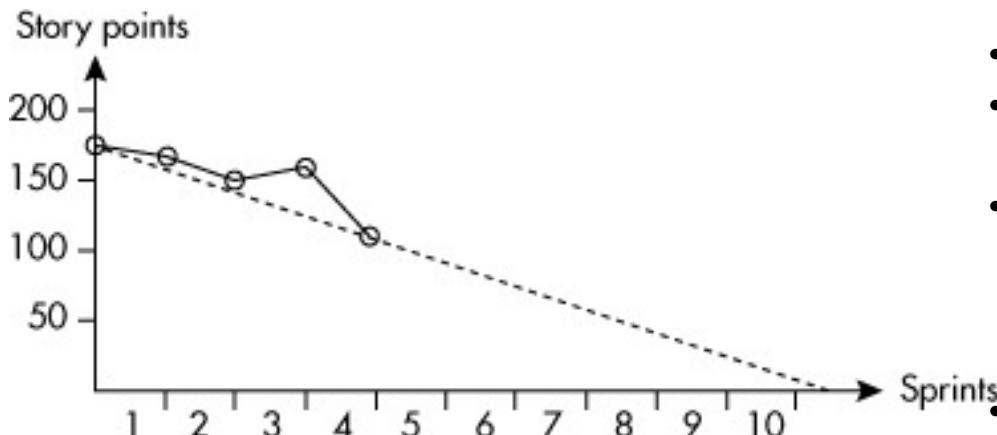
Iterations Completed	Low Multiplier	High Multiplier
1	0.6	1.60
2	0.8	1.25
3	0.85	1.15
4 or more	0.90	1.10

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	<i>Date or timeframe</i>	<i>Date or timeframe</i>	<i>Date or timeframe</i>	<i>Date or timeframe</i>	When will the release be available?
 NAME The name of the new release	<i>Name/version</i>	<i>Name/version</i>	<i>Name/version</i>	<i>Name/version</i>	What is it called?
 GOAL The reason for creating the new release	<i>Goal</i>	<i>Goal</i>	<i>Goal</i>	<i>Goal</i>	Why is it developed? Which benefit does it offer?
 FEATURES The high-level features necessary to meet the goal	<i>Features</i>	<i>Features</i>	<i>Features</i>	<i>Features</i>	What are the 3-5 key features?
 METRICS The metrics to determine if the goal has been met	<i>Metrics</i>	<i>Metrics</i>	<i>Metrics</i>	<i>Metrics</i>	How do we know that the goal is met?

Goal Oriented Roadmap – An Example



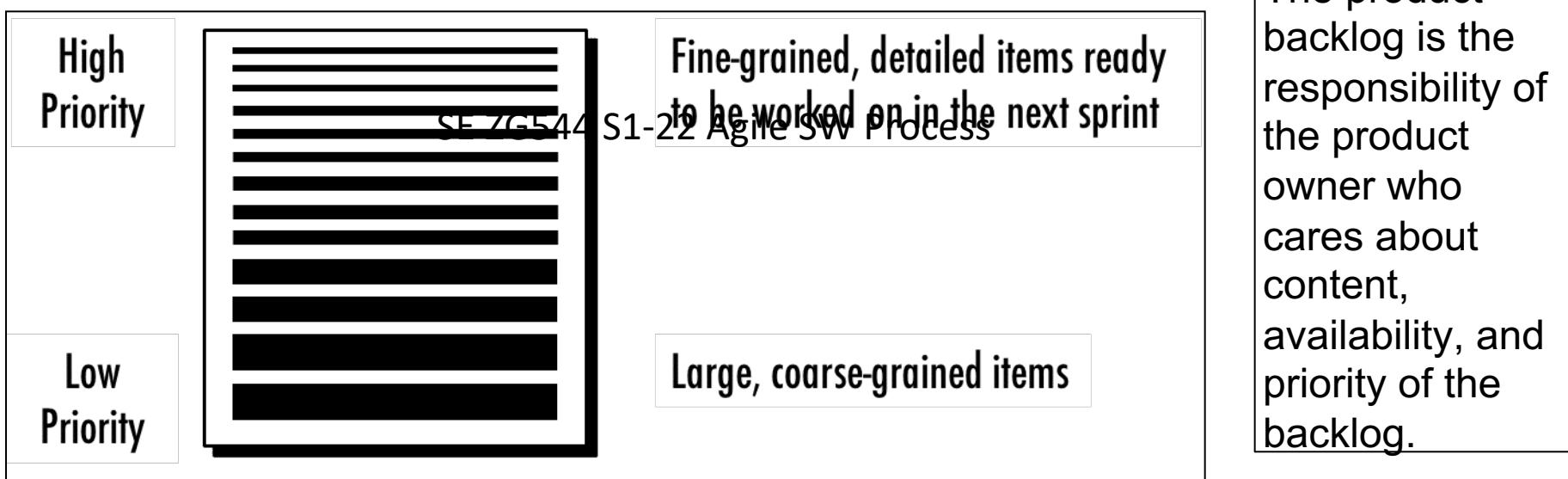
Develop a new dance game for girls aged eight to 12 years. The app should be fun and educational allowing the players to modify the characters, change the music, dance with remote players, and choreograph new dances.

	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/>

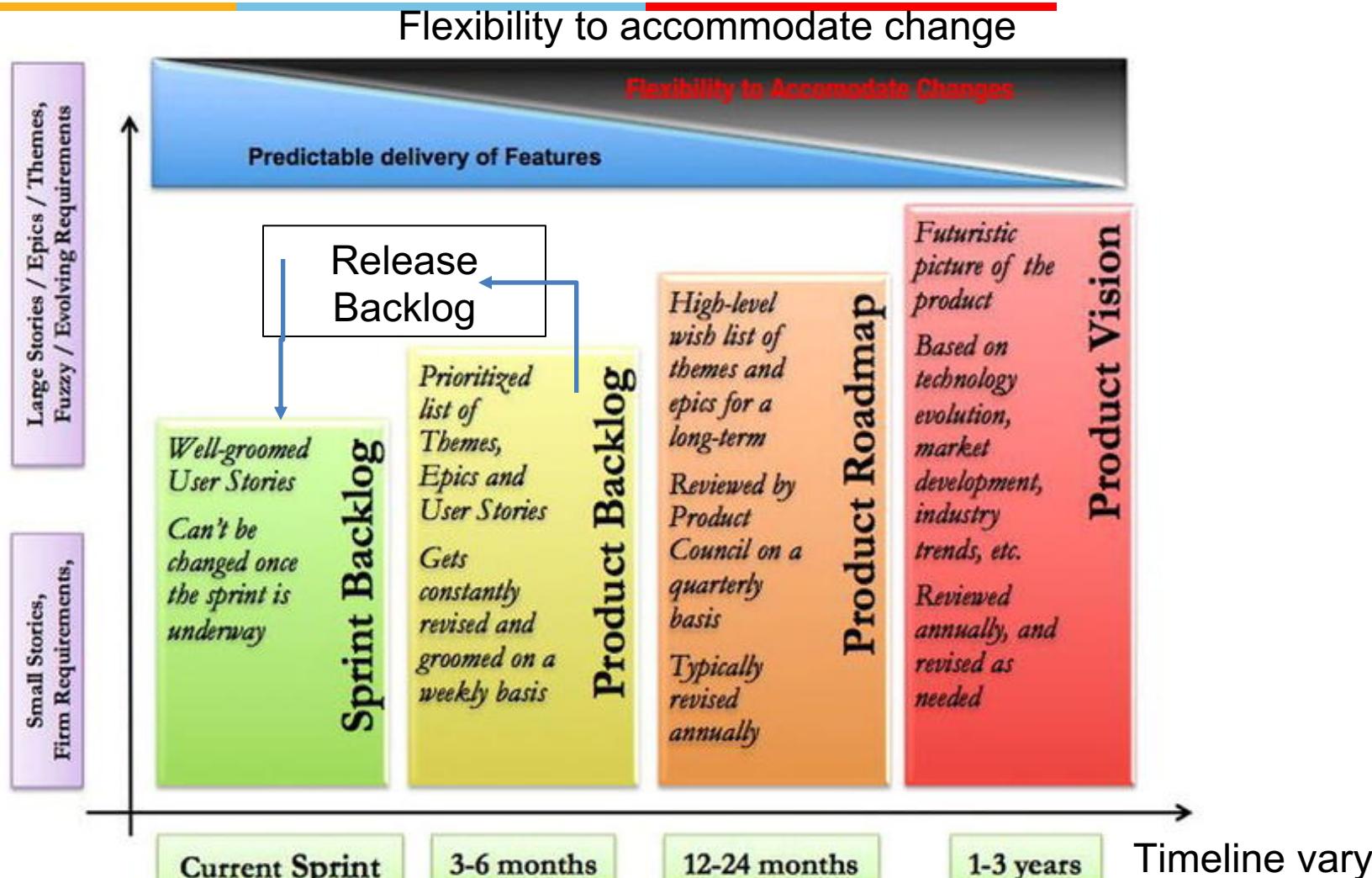
Characteristics of a Product Backlog



- 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.

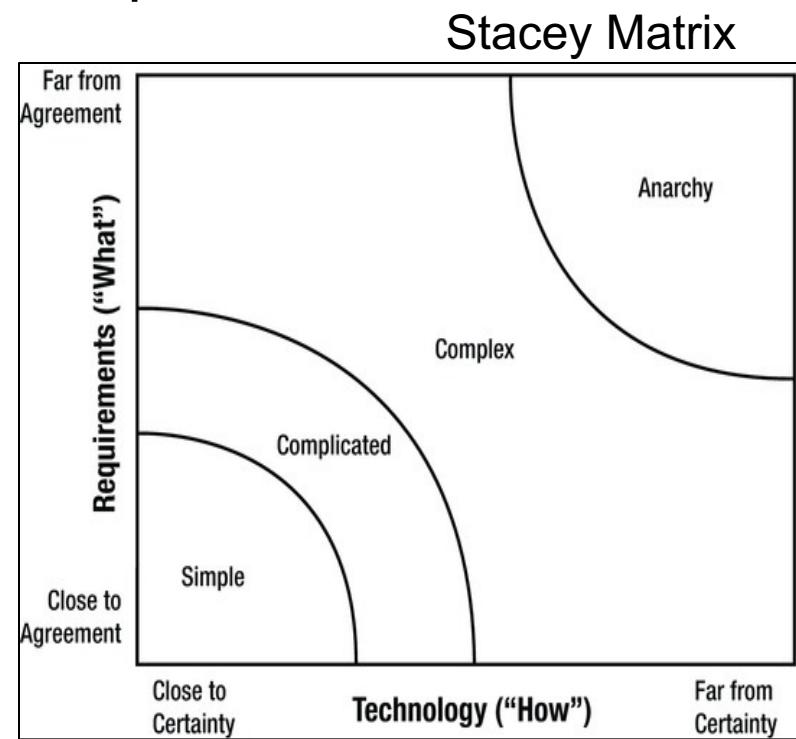
Ref: Agile Project Management: Creating Innovative Products, Second Edition by Jim Highsmith Published by Addison-Wesley Professional, 2009

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

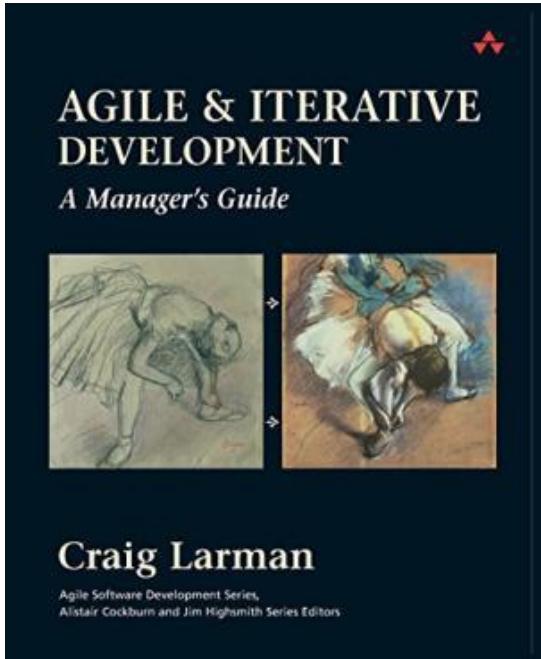


Agile Methods – An Introduction

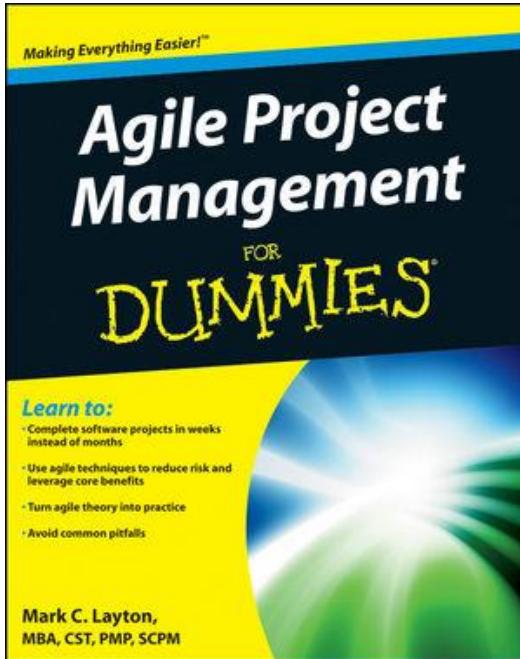
- Prof K G Krishna

Text/Reference Books

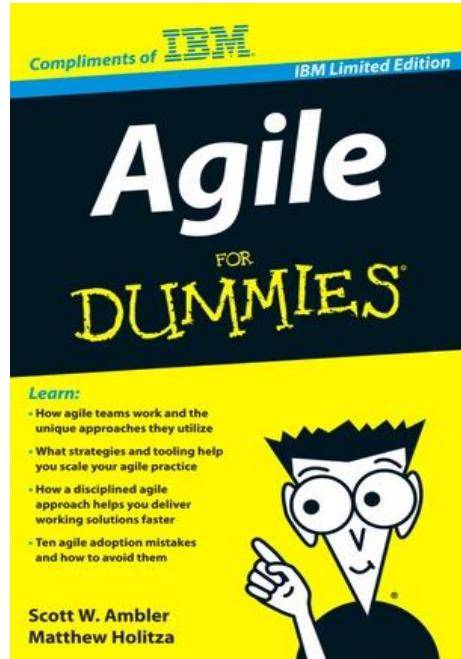
T1



T2



Compliments
of IBM



➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

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



Traditional Software Development – The Backstory

- Large Software Projects (Mainframe era)
- Development Models based on Linear Models (Sequential/Waterfall)
- No Time-to-Market Constraints (Software is Expensive)
- Mostly Custom/Bespoke Software Development
- Programmer-Intensive Manual Activities

Notes SPE

- ① Following ENTER SPE as directive by *
- ② Incorporating current line pointer concept
- ③ in A25 mode display line numbers
- ④ Switching back to edit mode by 'm'
- A line can be added after line no 'm' by
- ① D25 m
- ② DSP m
- ③ Any line not ending in termination line is 'm'
- Ex: DSP m.m
- SEL m.m
- CR 1 / 1.m
- DEL m.m
- REP m.m
- PRT m.m *Since preceding "NL"
- ④ ADD m at any pos in SPE
- ⑤ *m.m: position the line pointer at 'm' displaying the line
- ⑥ Default Warning: *SEL (only current line) line
- *SEL m (following preceding line)
- ⑦ Specifying an option for no line
- Ex: SEL m@m .m@m, m@m
or SEL m@m
- ⑧ X PND string/m.m
- ⑨ Specifying default 'A' in *SEL A (selectable) and 'L' selection CTG/SEL/SEL

K G Krishna



"A temporary endeavour undertaken to create a unique product, service, or result." -
A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 4th Edition
(Project Management Institute, 2008)

Traditional Software Projects (circa 1990 ~ 2000)

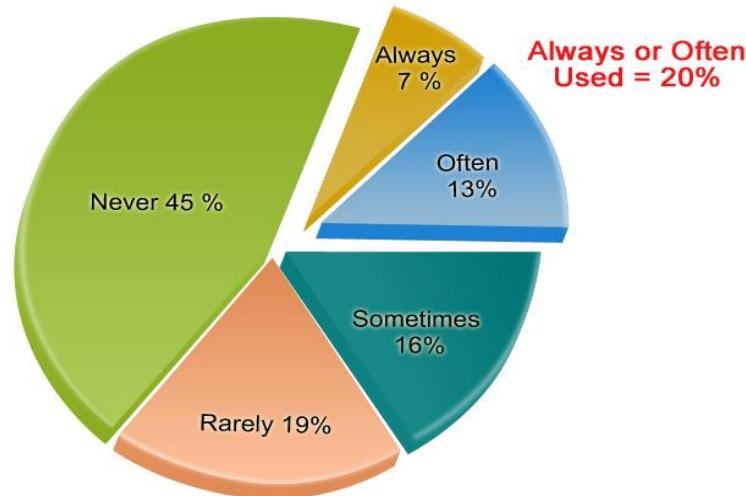
46 % Challenged

- functionality & quality
- over budget
- time overruns



Source: Standish Group Chaos Report [2006]

Features and Functions Used in a Typical System



Standish Group Study - Reported at XP2002 by Jim Johnson, Chairman

Copyright © 2011 luuduong.com

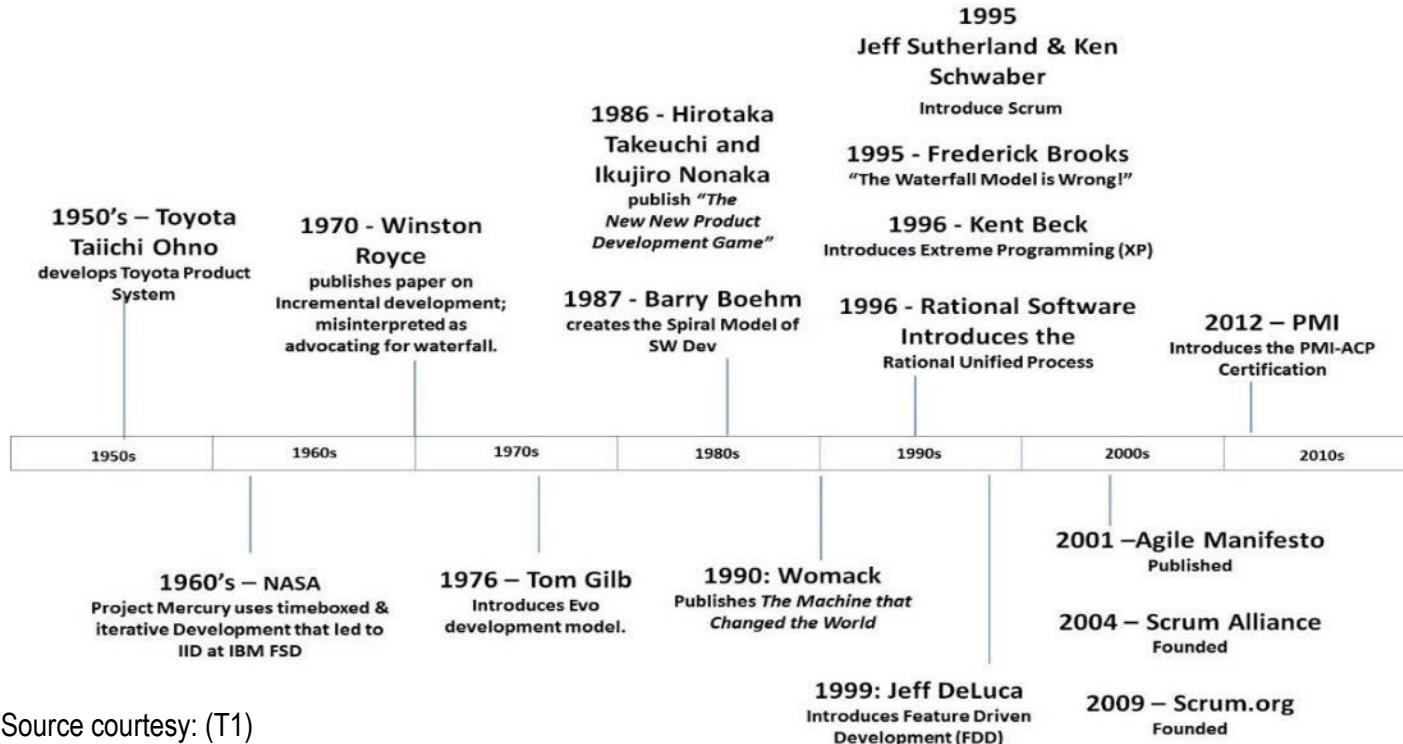
In 2009, companies and organizations in the U.S. spent \$491.2billion on application development. That means that more than \$103billion was wasted on failed projects.

Agile Turns Upside-down Project Constraints



Production-Line → Agile System – An Evolution

A Brief History of Agile



Being Agile Means...

agile

/'adʒəl/ 

adjective

1. able to move quickly and easily.

"Ruth was as agile as a monkey"

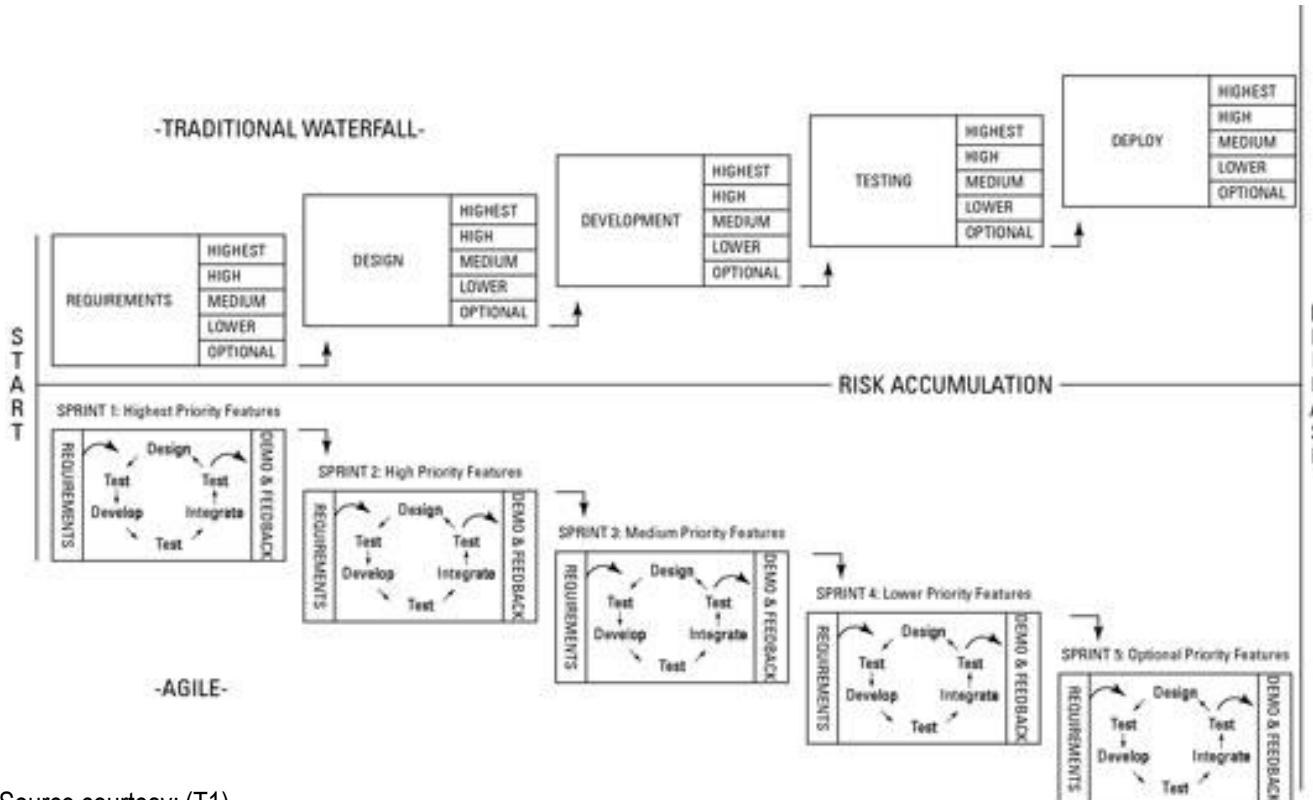
synonyms: nimble, lithe, spry, supple, limber, sprightly, acrobatic, dexterous, deft, willowy, graceful, light-footed, nimble-footed, light on one's feet, fleet-footed; More

- **Agility**

The ability to both create and respond to change in order to profit in a turbulent business environment

- Rigid Processes vs. Agile Frameworks
- Being Agile = Competitive, Responsive, Flexible,...

Agile Development – A Series of Short Sprints



Source courtesy: (T1)

Agile Development = Iterative Releases!

- **Agile software development** is a conceptual framework for software engineering that promotes development **iterations** throughout the life-cycle of the project.
- Software developed during one unit of time is referred to as an iteration (**sprint**), which may last from one to four weeks.
- Agile methods also emphasize **working software** as the primary measure of progress



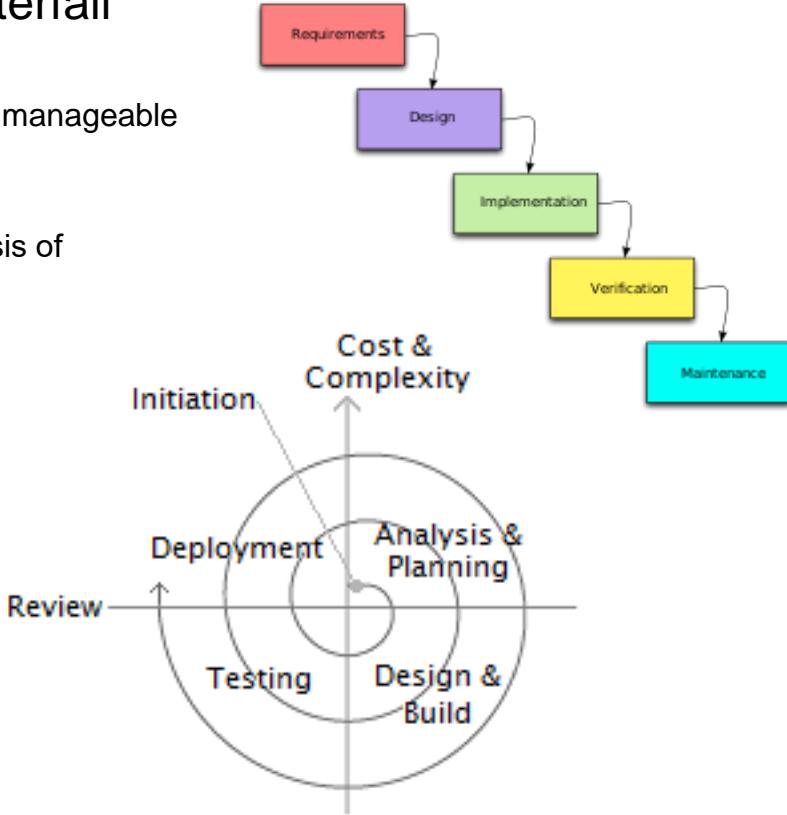
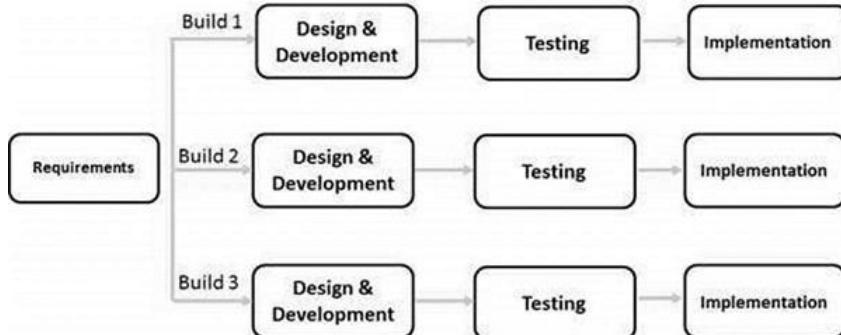
Need for Agile Methods →

Software Systems Today...

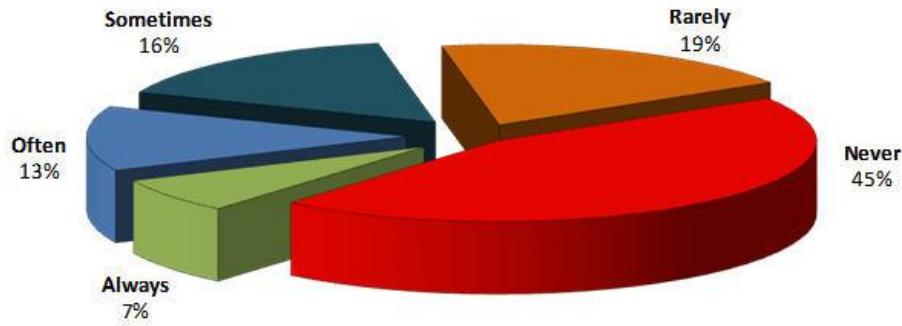
- Changing ('Unclear') Requirements / Customer Demands
"Requirements will be clear only at the end of the Project"
- Shrinking Development Cycles (Time-to-Market Releases)
- Entrepreneurial, Innovative Apps
- Shorter Shelf-life

Addressing Challenges of Waterfall Model

- Traditional Waterfall → Adaptations of Waterfall (Incremental, Iterative Prototyping,...)
 - Focus on Requirements Management (Breaking down into manageable 'Increments')
 - De-risking Large Development Effort
 - Capturing Requirements ('show-and-tell', 'prototype' as basis of discussion around Requirements)
 - "Quick-and-dirty" Working Prototype to Start with



Software Feature Overload

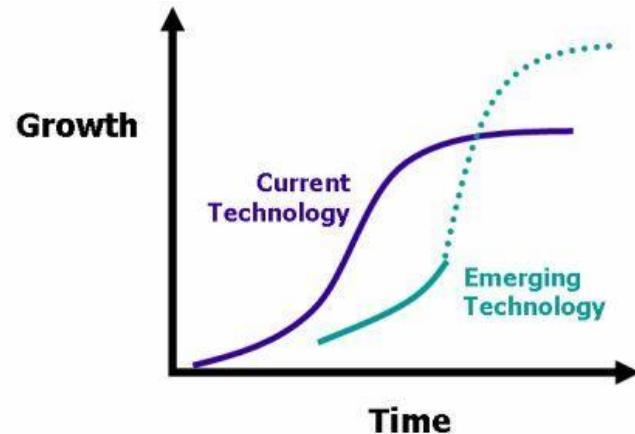


Source: Standish Group Study, 2002



Pace of Technology vs Project Duration

- Technology Life-cycles have shrunk (decade → 1..2..3 years)
- Bespoke Software → Product Customization
- Line-by-line Coding → Integrated Development Frameworks



De-risking Future Investments Upfront...

- “Fail-Safe” early rather than Failure at the end
- “Proof-of-concept” when adopting new Technologies
- “Testing the waters” before launching Big in the Marketplace
- Today’s Software Products are strategic levers--“Idea-driven” rather than mere automation of manual process



Benefits of Agile Methods →

Controlled Development

- Agile Products are based on empirical control method – decisions based on reality
- Adjustments on-the-go by Frequent Inspections
- **Transparency:** Everyone involved knows what is going in the project
- **Frequent Inspection:** Regular evaluation of the Product
- **Adaptation:** Make quick adjustments to minimize problems later

Agile Development → Agile Project Management

- Traditional Project Management Turned Upside-down!
- *“Let’s wait till the Project completes to see the Product” →
“Several Min-projects with little visible successes”*
- Transformation of Project Management by actively involving
ALL the Stakeholders; Organization Structures &
Communication; Time-Boxing of Deliveries

“Standish Group Study on Software project success and failure: In 2009, 26 percent of projects failed outright — but in 2011, that number fell by 5 percent. The decrease in failure has, in part, been attributed to wider adoption of agile approaches

Benefits of Agile Project Management

- *Almost Zero Risk of Catastrophic Project Failure*
- Prioritization of Business Value over 'Good or Nice-to-have' features
- Agile Testing (Continuous Testing) ensures Problems are discovered early
- Down-plays 'Scope-creep' as Requirement Changes are managed throughout Product Development Life-cycle
 - Prioritizing Features in early Iterations
 - Managing Evolving Requirements
- Continuous Inspection and Adaptation: Improvement of Processes and Products based on Prior Experience of the 'Completed' Product

Agile Methods - Summary

- Traditional Software Development Methods involving Large One-time Projects are yielding to Low-risk High-turnaround Incremental Deliverables in Agile Methods
- Involvement of All Stakeholders (including Customer) early on in the Process enhances Collaboration and minimizes Scope-creep
- Full Transparency and High-visibility of Deliverables in Short Iterations (Sprints)
- Continuous Quality Monitoring with embedded Agile Testing across all Iterations

“Agile is the Great Leap Forward in Software Development Methodology with its associated Transformation in Agile Project Management”

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

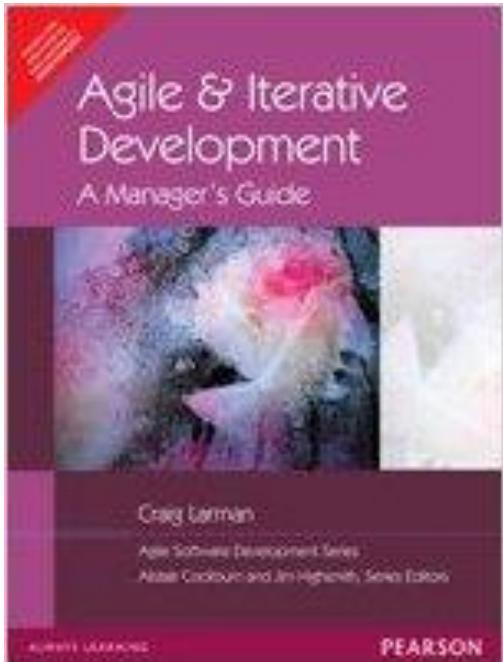


Agile Software Development

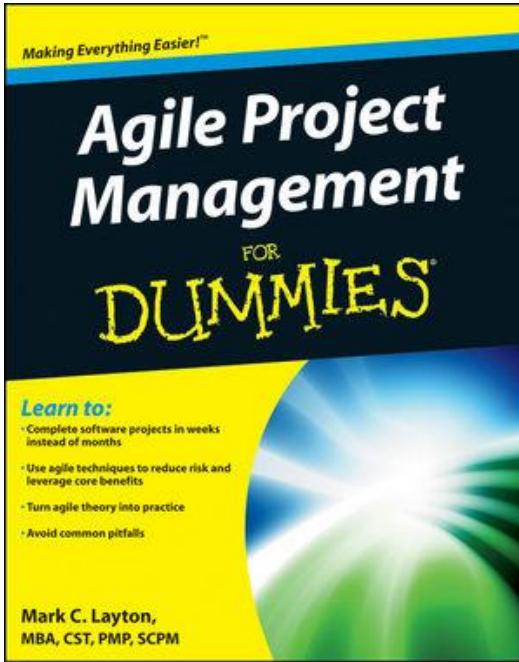
- Prof K G Krishna

Text/Reference Books

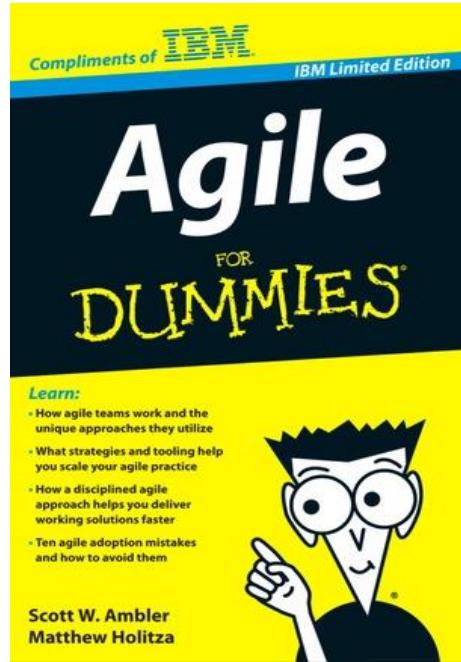
T1



T2



Compliments
of IBM

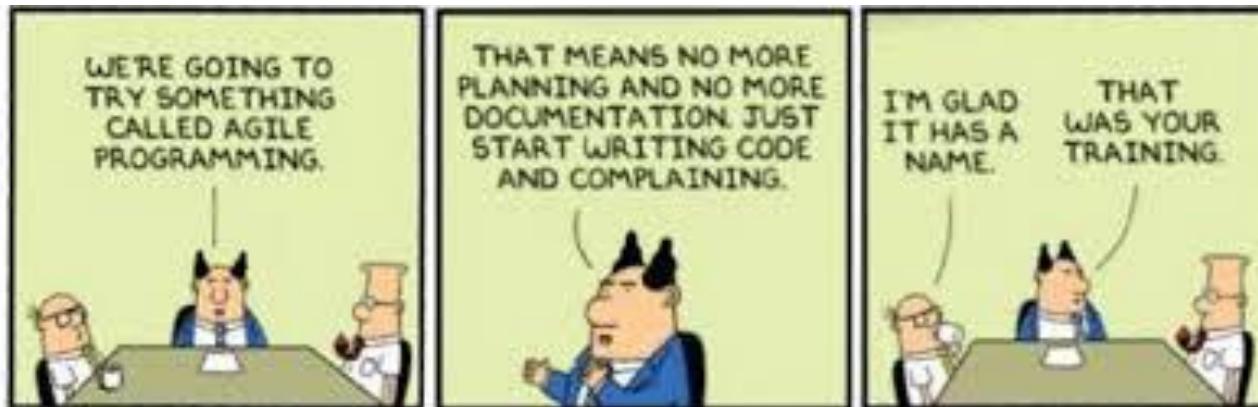


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Basics of Agile Software Development

- Iterative and Incremental Approaches
- Risk driven and client driven development
- Time-boxed development
- Adaptive and Evolutionary development



© Scott Adams

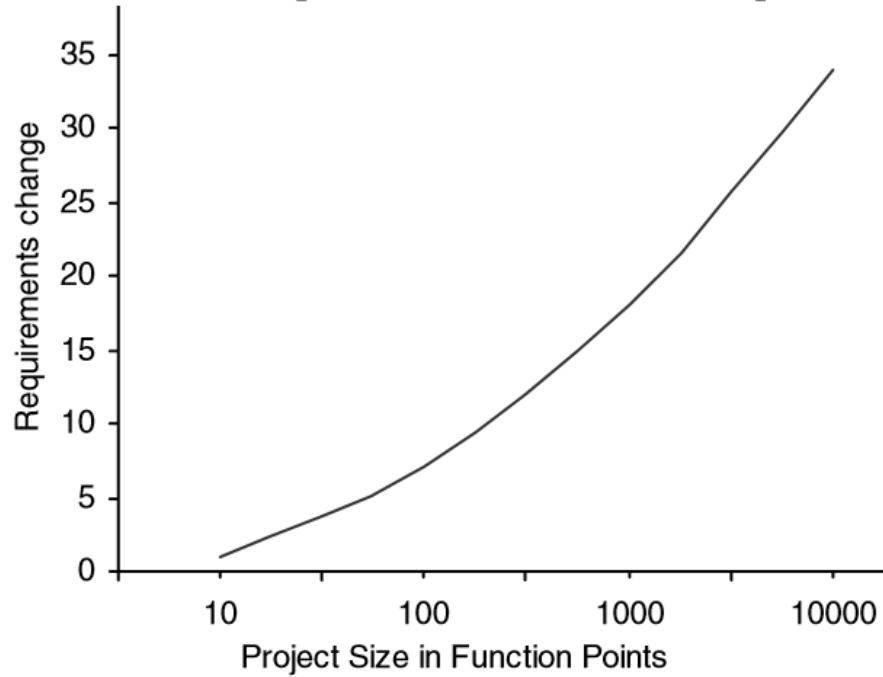
Customer Requirements? Hard To Get!

“Ours is a world where people don't know what they want and are willing to go through hell to get it.”

—Don Marquis

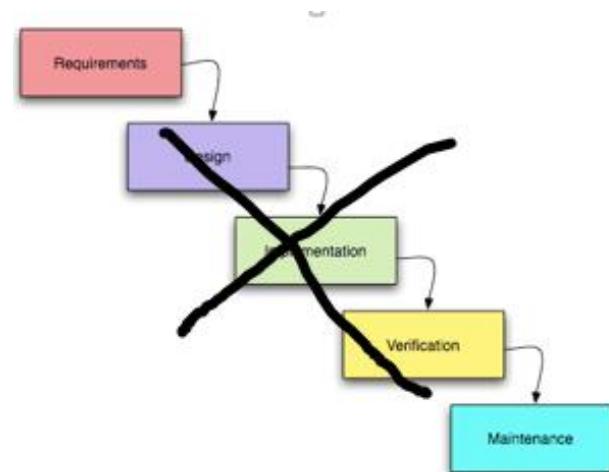
- “Requirements are capabilities and conditions to which the system—and more broadly, the project—must conform”
- “A prime challenge of requirements analysis is to find, communicate, and remember (that usually means write down) what is really needed, in a form that clearly speaks to the client and development team members.”
- More than 50% of Requirements keep changing through the Development cycle (particularly true while working with Oriental Customers like Japanese)

Waterfall is nightmare...Don't Go For It!



The ills of Waterfall...

- Originated in the Mainframe era (suited for Large Enterprise Projects)
- Freezing Requirements Upfront (*Reality is different*)
- Long Project Life-cycles (>>2 years)
- Lack of Transparency and Visibility to Customer
- “Last-minute surprises to Customer upon Delivery”
- Documentation overhead (“non-value-adding?”)
- “Work fills available schedule”
- Hierarchical Team Structures
- ...



Iterations in Waterfall? No Good Either...



Iterative & Incremental...is the Way To Go!

You should use iterative development only on projects that you want to succeed.

—Martin Fowler

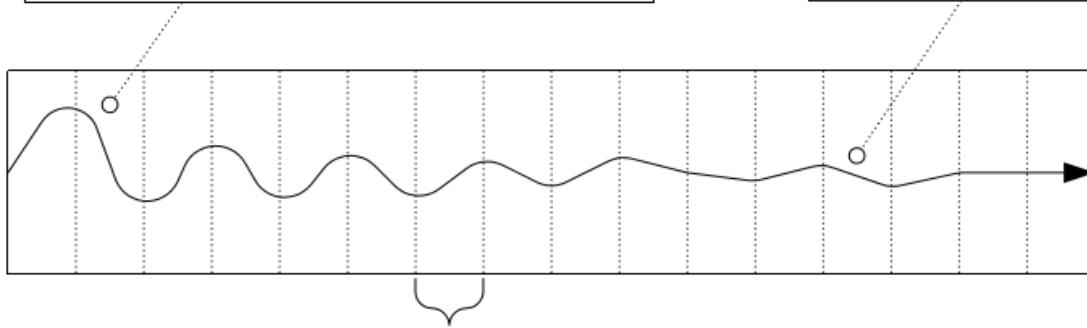
- Early Programming & Testing of Partial System, in Repeating Cycles in Agile vs. Early Upfront Speculative Requirements Freeze before Programming in Waterfall Models
- Refinement of the System through Successive Fixed-length Iterations (mini-projects or Sprints)
- Common **Agile Processes**: Scrum, Lean Development, Unified Process, Test-Driven Development (TDD), Feature-Driven Development (FDD), Adaptive Software Development, etc.



Iterations Converge to True Requirements!

Early iterations are farther from the "true path" of the system. Via feedback and adaptation, the system converges towards the most appropriate requirements and design.

In late iterations, a significant change in requirements is rare, but can occur. Such late changes may give an organization a competitive business advantage.



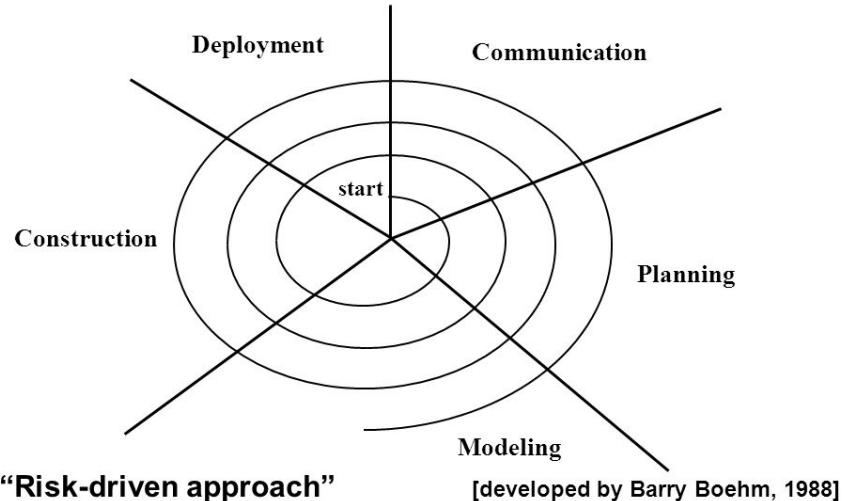
one iteration of design,
implement, integrate, and test

NO 'Waterfall Thinking' in Iterative Development! "...on average 45% of the features in waterfall requirements are never used, and early waterfall schedules and estimates vary up to 400% from the final actuals..."

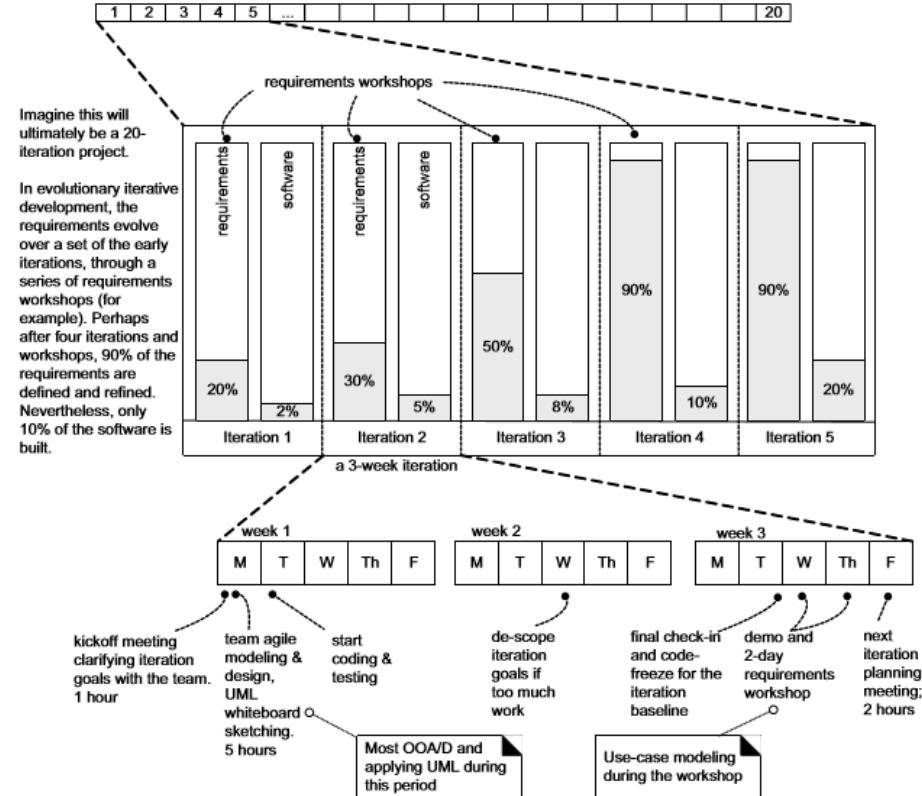
Risk-Driven, Customer-Driven Iterative Planning

- Early Iterations with Highest Risk
- Ensure Early Visibility of Key Features
- Focus on Stabilizing Architectural Choices

Spiral model



Evolutionary Analysis & Design in Early Iterations



Typical *Iteration* includes...

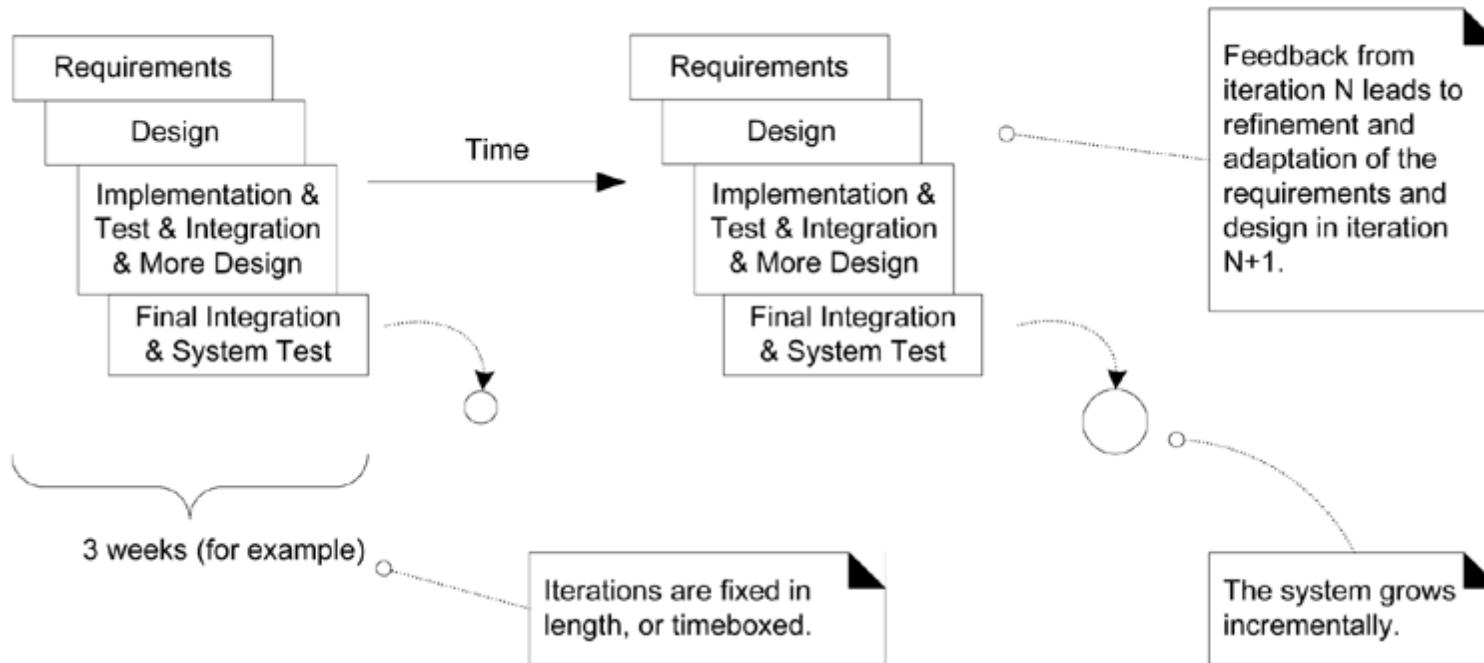
- Well-defined, **Prioritized** Set of Requirements
- **Time-boxed** Schedule (Deadline = 'Dead'line!)
- Output Deliverable: Tested, Integrated and *Partial* Usable System
- Each Iteration includes its own **Requirements Analysis** → *Design* → *Programming* → ... → *Testing* Cycles (mini-waterfall)
- Incorporates **Feedback** (from Customer and other Key Stakeholders) after every Iteration

Evolutionary

Incremental

Iterative

Time-Boxed Iteration with Feedback



Let's Review Few **Popular Agile Methods:**
(Lean, Extreme Programming, SCRUM) →

Agile Frameworks

- Common Frameworks (Methods & Techniques in **Practice**) that embrace characteristics of *Agile* :
 - Lean Software Development (Kanban)
 - Extreme Programming (XP)
 - SCRUM (widely adopted today)
- Common **Principles** that Govern The Agile Frameworks
 - Iterative Development: by Multiple Iterations
 - Simplicity, Transparency and Situational-strategies (being ‘street-smart’ for ‘rubber-meets-the-road’ challenges)
 - Cross-functional, Self-organizing Teams
 - Visible Progress: measured by Working Software at any instant

Projects with Agile Frameworks: Defining Structure vs. Being Prescriptive

- Projects adopting **Lean Methods** & **SCRUM** focus on well-defined Structure and Roles
- Extreme Programming (**XP**) Techniques like *Pair-programming, Release Planning Game, Test-driven Development (TDD)*, etc., are more prescriptive in the nature of project activities
- While SCRUM is the most popular Methods practiced in organizations, individual developers/entrepreneurial setups adopt a creative combination of the methods to meet the project challenges

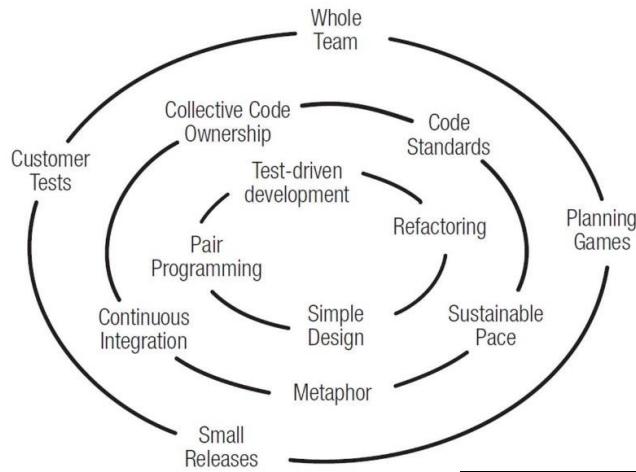
Lean Methods – Inspiration for Agile

- Originated in Japanese Manufacturing Organizations (Toyota's *Kanban* system), Lean Methods **Focus** on:
 - Eliminating Wastage in mass-manufacturing processes (Just-In-Time Production System)
 - Focus on Humans in the Decision-making in the Production Process (vs. Expensive Machines)
 - Ownership by Shop-floor Workers (vs. Supervisors/Managers)
- **Principles** of Lean:
 - Optimize the Whole, Build Quality, Learn Constantly, Deliver Fast, Engage Everyone, Continuous Improvement (*Kaizen*)
- Lean applied to **Software Product Development**:
 - Avoid '**Unnecessary**' features
 - **People** (not Machines) are central to Project – they add real value
 - Involve Customers early-on and **Prioritize** Requirements
 - Constant **Communication** among All Stakeholders (using Tools)

Extreme Programming (XP)

- Guiding Spirit: *Extreme Focus on Customer and Projects are like War-rooms*
 - Features to be developed when Customer needs them
 - New Requests (or Change-requests) accepted as part of daily routine
 - Dynamic Self-organization of Teams around Customer Problems or Issues as and when they surface
- Principles of XP:
 - Coding is the Language of the Product and Communication
 - Extensive Testing: Coding doesn't start unless Success-criteria is defined;
“A bug is not a failure of code, it's a failure to define the right test”
 - Direct Communication between Programmer and Customer
 - Design during *Refactoring* to reduce Complexity and Maintainability

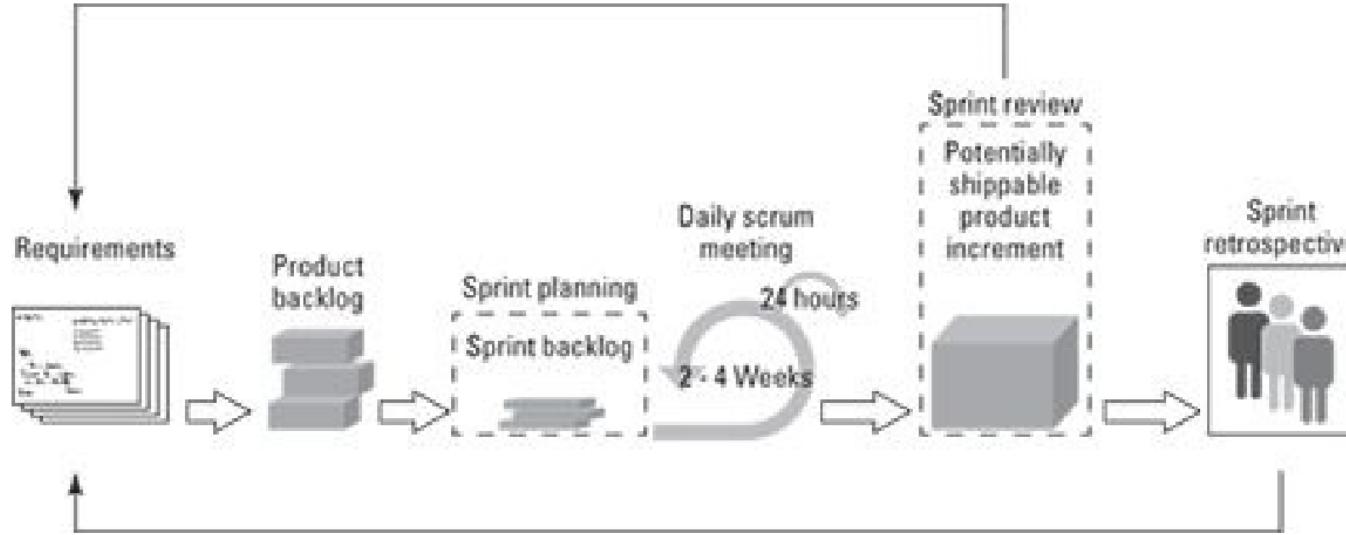
XP – Key Practices



Planning Game: All Members of the Team Should Participate in Planning – No Disconnect between Business and Technical People



SCRUM – The Common Agile Project Management Framework



Product Owner: Responsible for End-to-end Product Development

SCRUM Master: Manages the SCRUM Process (Not a *Manager* of Teams)

Cross-functional Teams: Involving Developers, Designers, Testers, and Operations Teams

Agile Software Development - Summary

- Customer and the Developer (Programmer) is at the Centre of any Agile Project Management Framework
- Core Characteristics of Agile: Time-Boxed and Iterative Development (via Short Iterations or Sprints); Continuous Feedback; Direct Involvement of all Key Stakeholders; Constant Communication; Transparency; Cross-functional and Self-organizing Teams,..
- Agile Methods: Lean (Kanban), XP and SCRUM
- SCRUM is the Common Agile Framework adopted in most Software Product Organizations

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

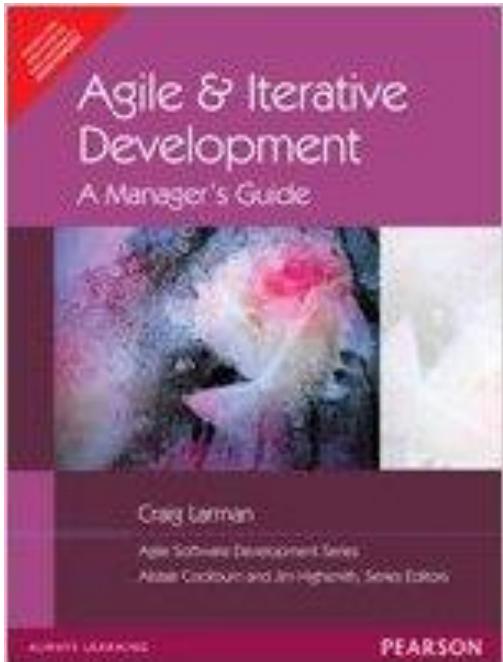


Agile Principles & Manifesto

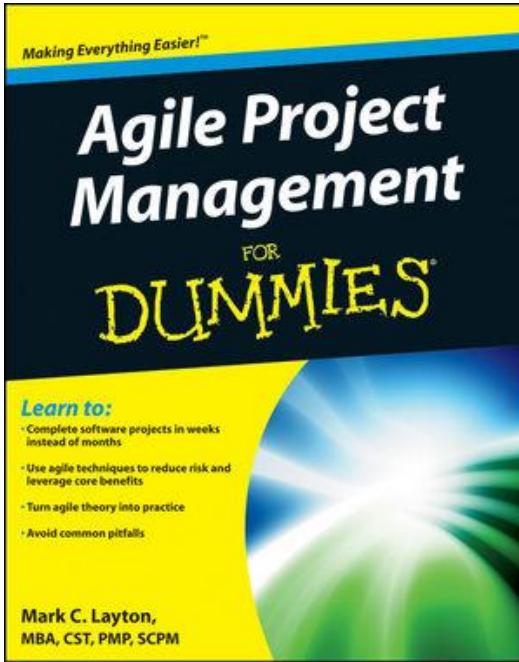
- Prof K G Krishna

Text/Reference Books

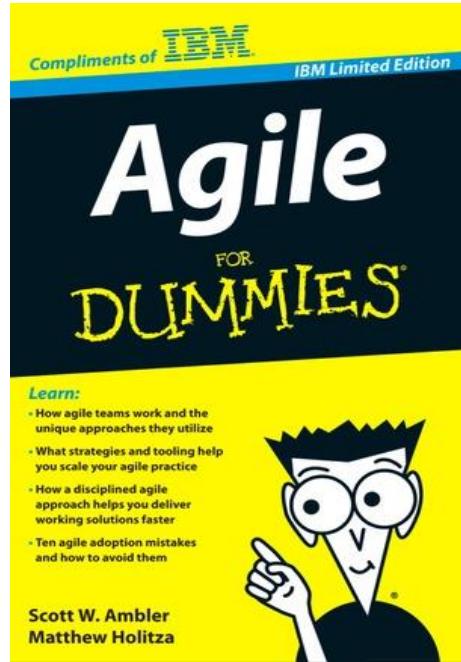
T1



T2



Compliments
of IBM



➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

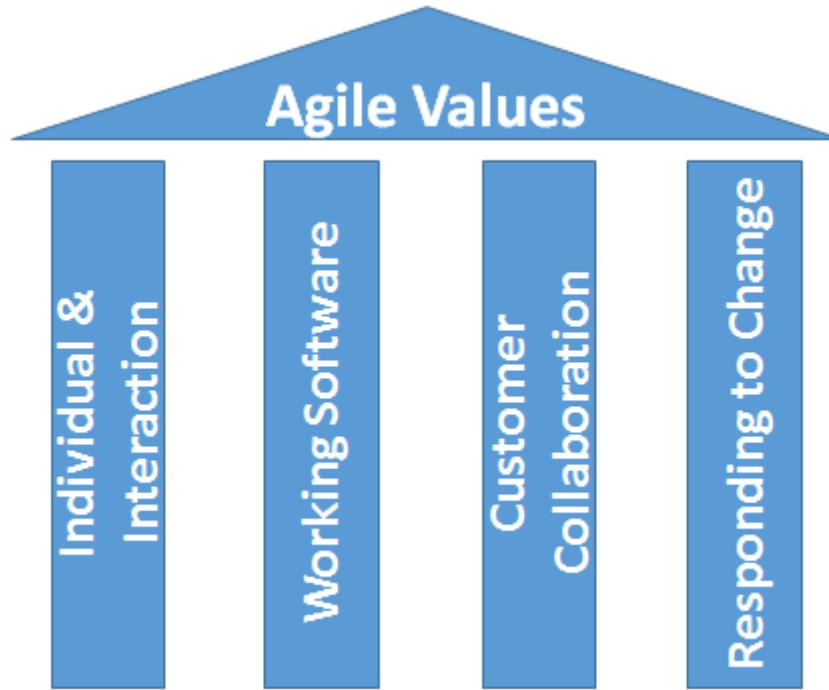
Topics

Principles of Agile

- Agile Values
- Agile Manifesto



Agile Core Values...



Agile Principles...

Principles behind the Agile Manifesto

We follow these principles:

- ✓ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- ✓ Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- ✓ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- ✓ Business people and developers must work together daily throughout the project.
- ✓ Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- ✓ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- ✓ Working software is the primary measure of progress.
- ✓ Agile processes promote sustainable development.
- ✓ The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- ✓ Continuous attention to technical excellence and good design enhances agility.
- ✓ Simplicity--the art of maximizing the amount of work not done--is essential.
- ✓ The best architectures, requirements, and designs emerge from self-organizing teams.
- ✓ At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Manifesto (2001)



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions** over processes and tools
- Usable**
- Working software** over comprehensive documentation
- Customer collaboration** over contract negotiation
- Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

© 2001, the above authors
this declaration may be freely copied in any form, but only in its entirety through this notice.

Source courtesy: agilemanifesto.org

Agile Manifesto (adapted to times)

Early and continuous delivery of software value

*Working software
Business impact is measure of progress*

Welcome changing emerging requirements

Self-organising teams

Deliver frequently continually

Technical excellence and good design

The Manifesto

Business and developers and everyone else working together

Simplicity

Build projects products around motivated individuals

Sustainable pace for sponsors, users, team all stakeholders

Value face-to-face communication

Regular Continual reflection and tuning

9 Principles Agile Project Manager

1. Deliver something useful to the client; check what they value
2. Cultivate committed stakeholders
3. Employ a leadership-collaboration style
4. Build competent, collaborative teams
5. Enable team decision making
6. Use short time-boxed iterations to quickly deliver features
7. Encourage adaptability
8. Champion technical excellence
9. Focus on delivery activities, not process-compliance activities

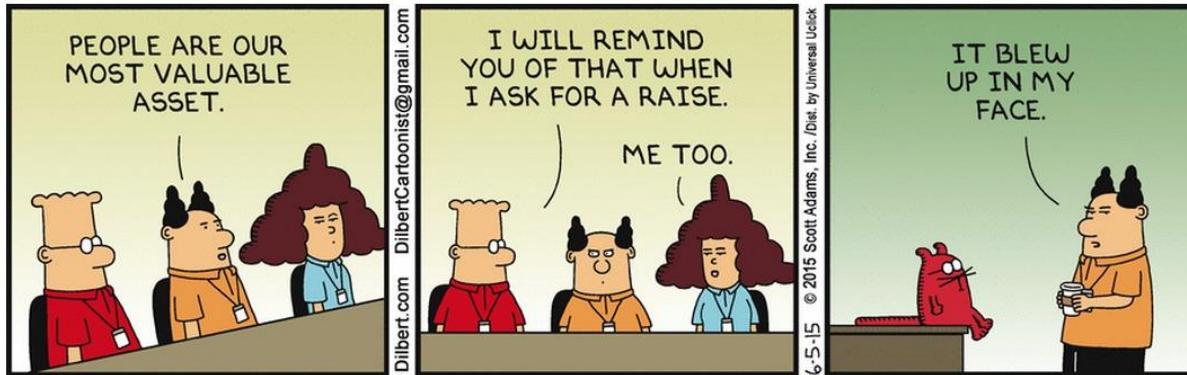
The Agile Project Manager

- Devolve of both control and planning to the entire team, not the manager
- The manager does not create WBS, schedule, estimates or tell people what to do
- The manager does not define and assign detailed team roles and responsibilities

The *Human Touch* in Agile

“People are more important than any process. Good people with a good process will outperform good people with no process every time” – Grady Booch

- Programming is an intense Human Activity - People matter much more than Machines
- Individuals and Interactions over Processes and Tools
- Sustainable Pace – Programmers to maintain a healthy social and family life
- Respect Diversity of Individual Contributions – Skill Transfer through *Pair Programming*
- Preference for Direct Face-to-Face Communication over Virtual Meetings or Remote Teams
- ...



Agile Principles & Manifesto - Summary

- Programming as if People Matter Most
- Customer is at the Centre – The Key Stakeholder
- Adapt to the Reality: Requirements keep Changing till the End of the Project
- Continuous and Incremental Delivery with a series of Working Product Releases
- Involve All Stakeholders Early-on
- Focus on Delivery over Process-compliance
- Close-knit Communication and Collaboration among Teams
- Agile Project Manager is the Leader and Motivator – Not the Boss of Manager of People
- Collective and Collaborative Decision-making
- ...

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

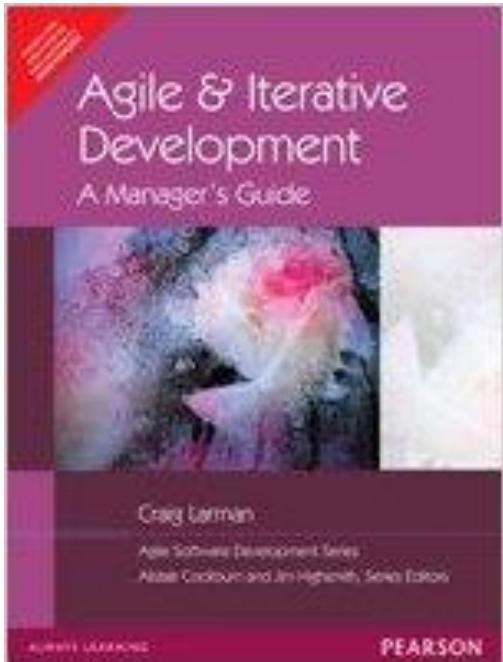


Agile Methodologies

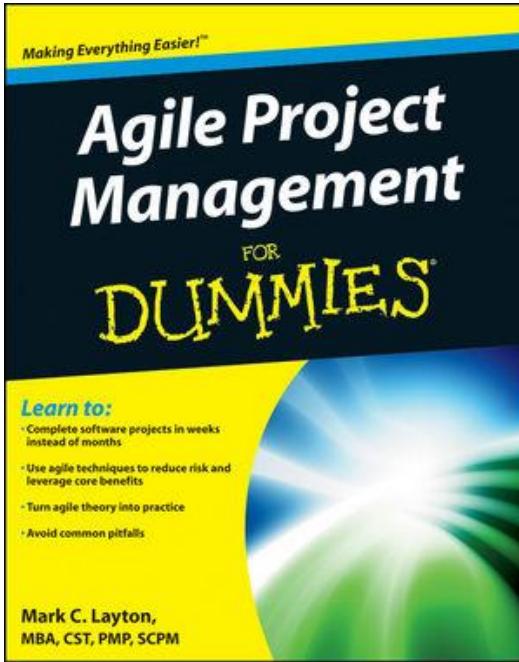
- Prof K G Krishna

Text/Reference Books

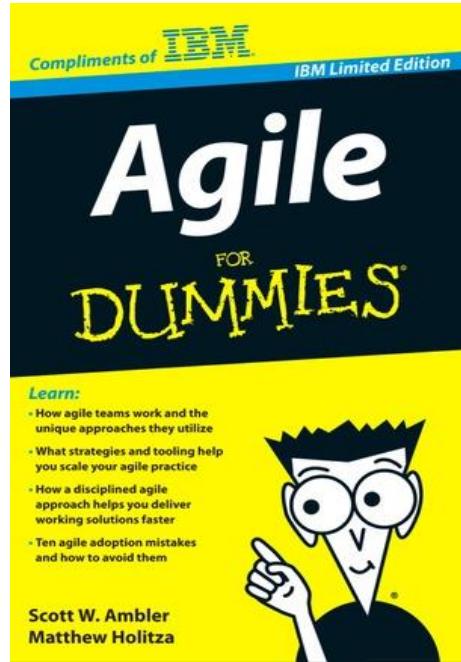
T1



T2



Compliments
of IBM



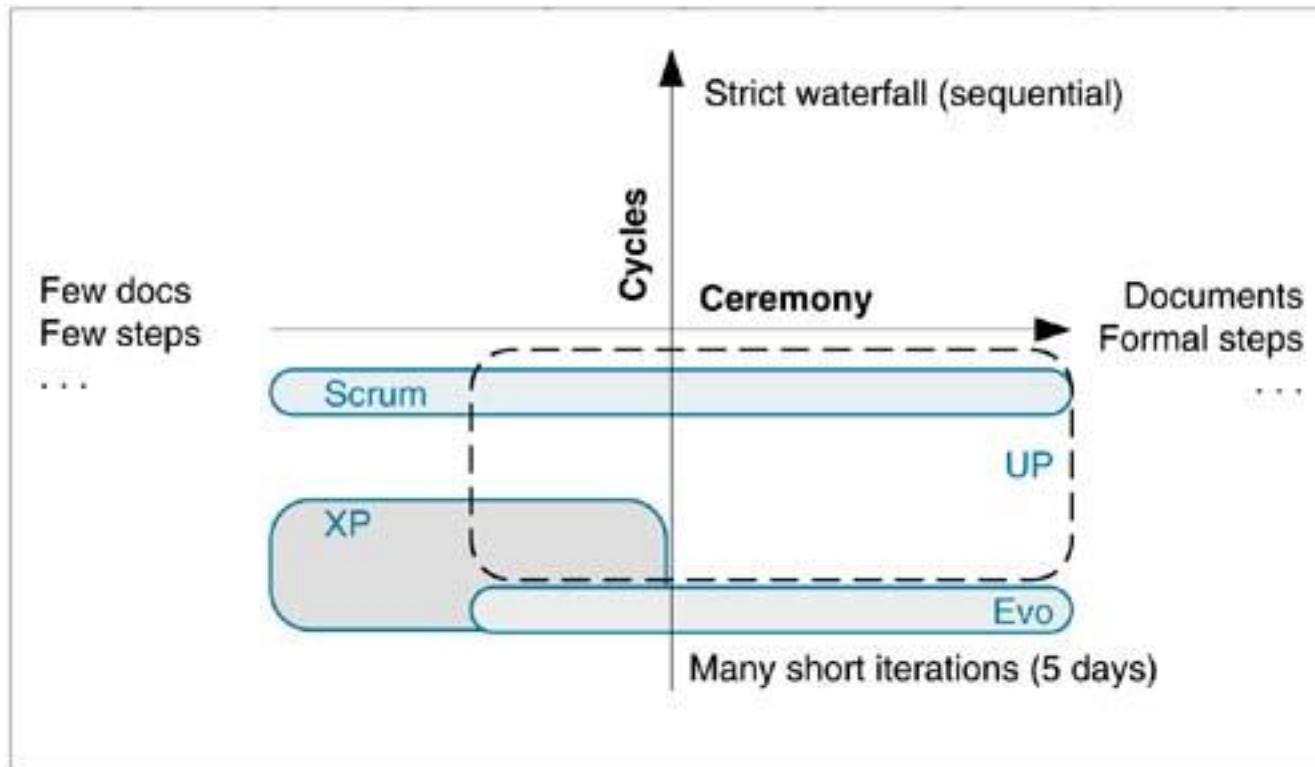
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Overview of Agile Methodologies

- SCRUM
- Extreme Programming (XP)
- Test-Driven Development (TDD)

SCRUM on 'Ceremony – Cycles' Scale



Source: T1-Chap7

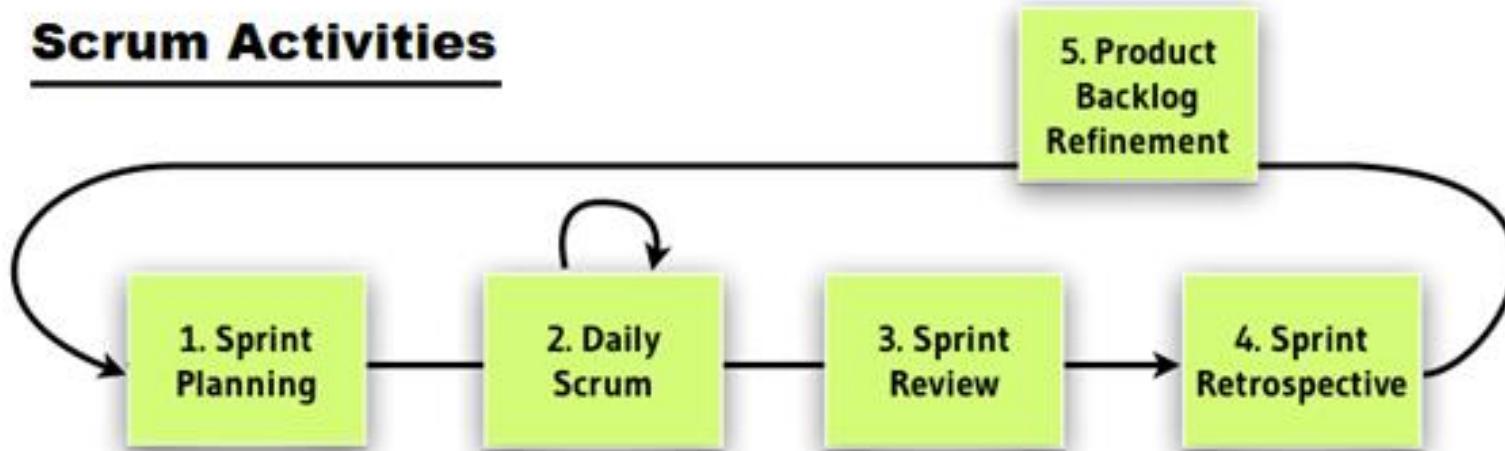
SCRUM Lifecycle

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

Source: T1-Chap7

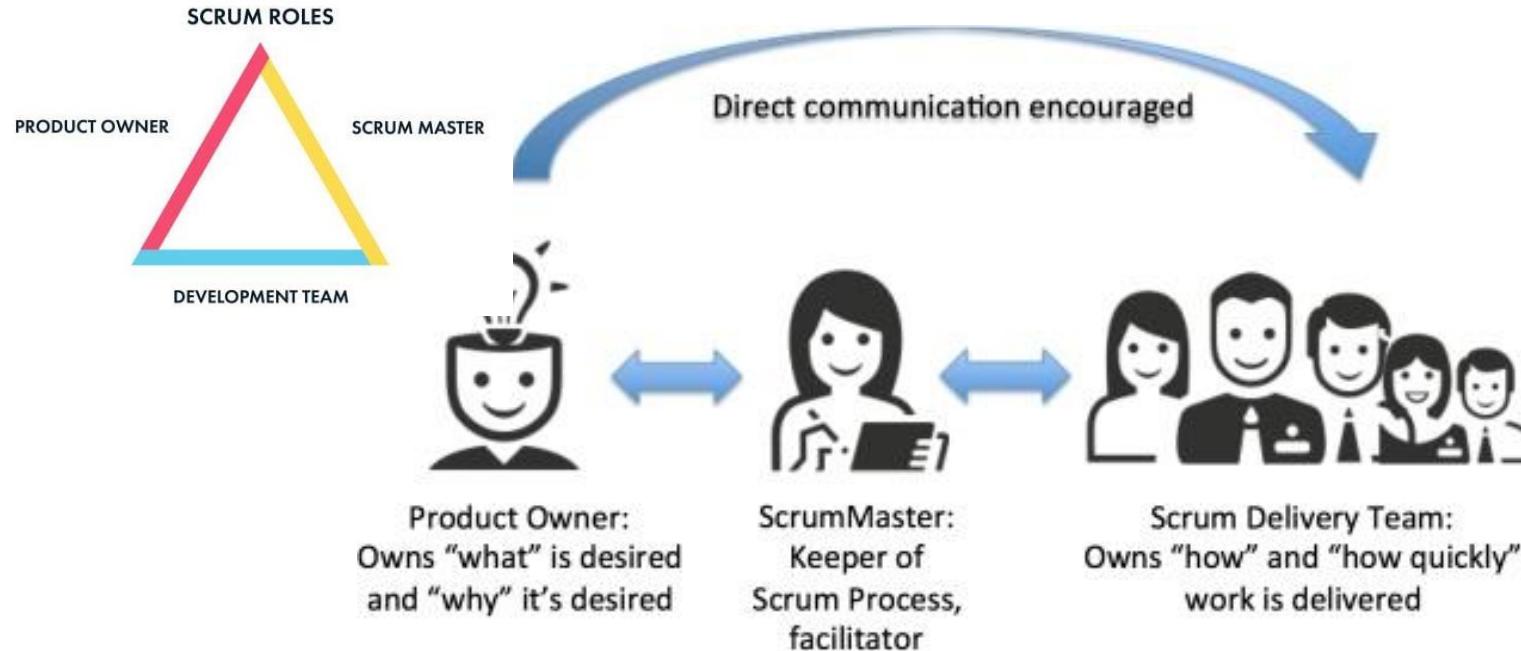
SCRUM Activities

Scrum Activities

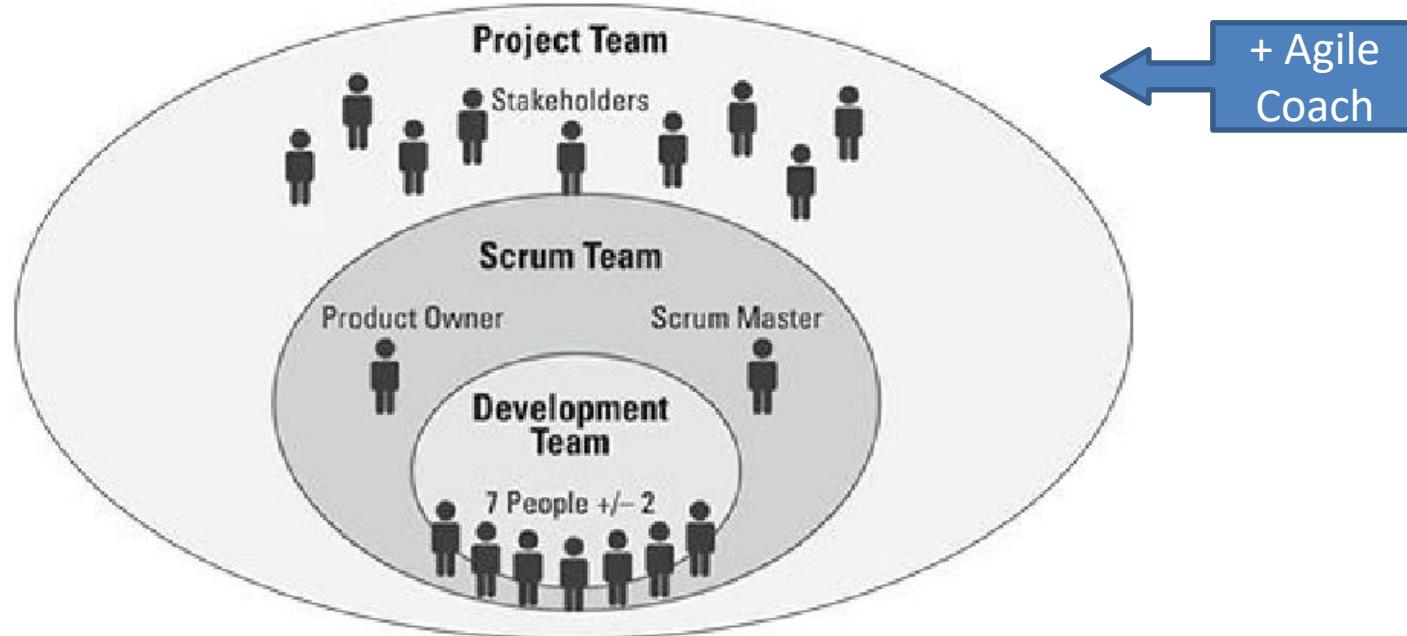


Source courtesy: www.snyxius.com/how-to-run-scrum-planning-meeting-like-pro

SCRUM – The Key Players

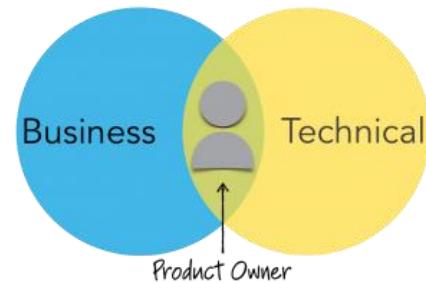
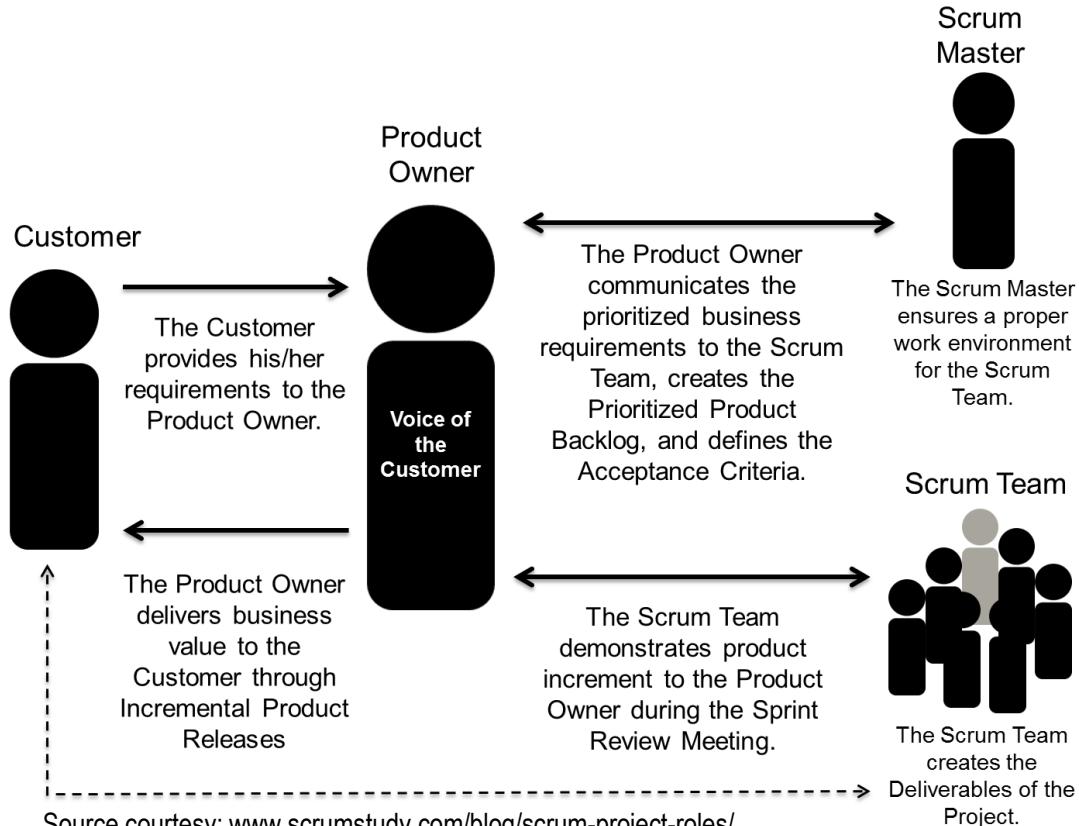


The 'Extended' SCRUM Team

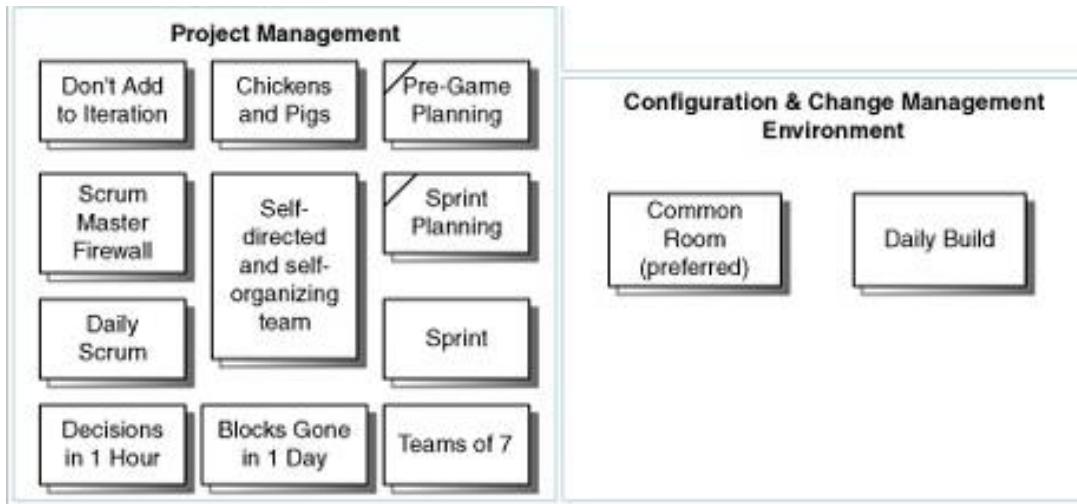


Source: (T2)

SCRUM Project Roles



SCRUM Practices



Source: T1-Chap7

The Daily SCRUM Meeting – The Heartbeat of the Project

- Daily Standup Meetings (1 – 7 Members, 15 – 20 minutes)
- Typical Questions Answered:
 - What have you done since the last Scrum?
 - What will you do between now and the next Scrum?
 - What is getting in the way (blocks) of meeting the iteration goals?
 - Any tasks to add to the Sprint Backlog? (missed tasks, not new requirements)
 - Have you learned or decided anything new, of relevance to some of the team members? (technical, requirements, ...)
 - ...
- Non-Team Members ('chickens') Can't Ask Questions – they just listen
- White-board Meeting / Tele-Conf Call
- Shared Responsibility and Team Cohesiveness is maintained

SCRUM Artifacts

	A	B	C	D	E	F			
1	<h2>Product Backlog</h2>								
2									
3	Requirement	Num	Category	Status	Pri	Estimate			
4	log credit payments to AR	17	feature	underway	5	2			
5	process sale-simple cash scenario	232	use case	underway	5	60			
6	slow credit payment approval	12	issue	not started	4	10			
7	sales commission calculation	43	defect		A				
8	lay-away plan payments	321	enhance		B				
9	PDA sale capture	53	technology		C				
10	process sale-credit pmt scenario	235	use case		D				
					E	F	G	H	I
1	<h2>Sprint Backlog</h2>								
2	Task Description	Originator	Responsible	Status	Hours of work remaining				
3					6	7	8	9	10
4					362	322	317	317	306
5	Meet to discuss the goals and	JM	JM/SR	Completed	20	10	0	0	0
6	Move Calculations out of	TL	AW	Not Started	8	8	8	8	8
7	Get GEK Data	TN		Completed	12	0	0	0	0
8	Analyse GEK Data - Title	GP		In Progress	24	20	30	25	20
9	Analyse GEK Data - Parcel	TK		Completed	12	12	12	12	12
10	Define & build Database	BR/DS		In Progress	80	80	75	60	52

Source: T1-Chap7

SCRUM Values

(“At the end, It’s the People that Matters the Most”)

- **Commitment:** The Team given Authority and Autonomy to decide; The Product Owners commits to Product Backlog; The Scrum Master commits not to introduce new work items till Iteration is complete
- **Focus:** The Scrum Master ensures the Team is not distracted; commits Resources and removes Roadblocks if any
- **Openness:** Product Backlogs and Daily Progress of Work Items are Visible to the entire Team
- **Respect:** Diversity of Individual Strengths and Weaknesses; facilitate Self-directed Teams; Teams empowered to seek/hire resources
- **Courage:** The Team has the courage to take Decisions adaptively; Management is supportive by empowering Teams

SCRUM Drawbacks (discountable however!)

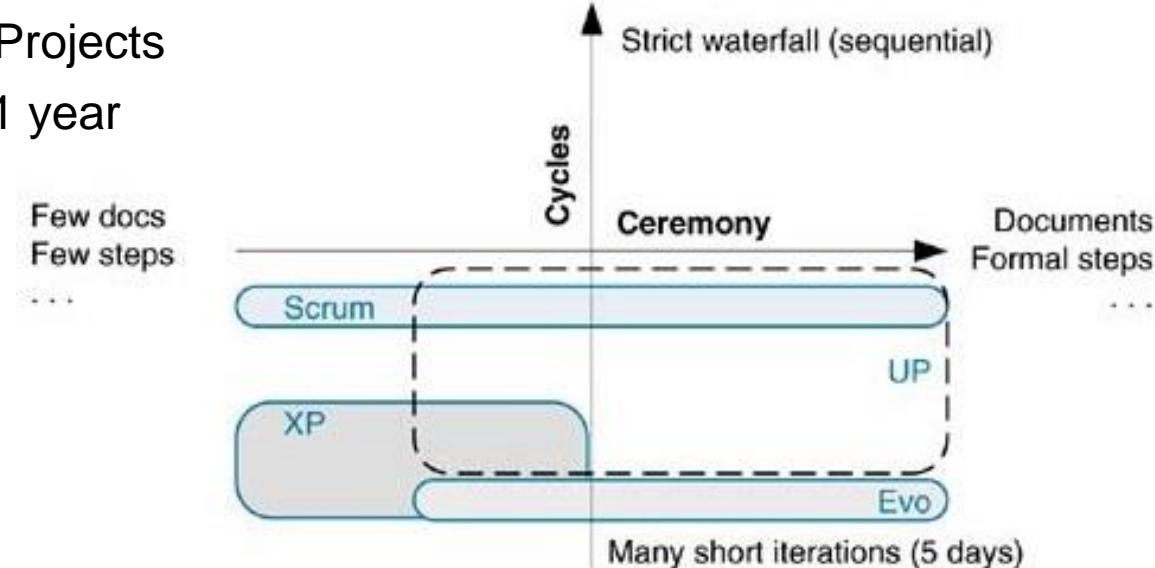
- **Active Engagement of the Customer:** the customer has to be continually involved in the project; however, while this can be seen as an advantage, it requires a lot of time and effort to manage the process
- **High Visibility may lead to Scope Creep:** as the Scrum methodology makes problems more visible, any problems arising at the end of each phase can also lead to "scope creep".
- **Small Teams losing Focus on the Big Picture:** while Scrum encourages small teams, it implies that larger teams may have to be broken down into smaller sizes which could result in the smaller teams loosing sight of the overall project focus.
- **Increase of Project Cost:** Scrum requires a certain level of training for all users which could increase the overall cost of the project.
- ...

Extreme Programming (XP) – A Set of Skillful Practices:



Extreme Programming (XP) – Core Values

- Communication, Simplicity, Feedback, Courage
- “Informal” (low on ‘ceremony’, usage of *Story Cards*)
- Small Teams (<10)
- Non-mission-critical Projects
- Delivery Schedule <1 year



Source: T1-Chap8

XP Lifecycle

EXPLORATION	PLANNING	ITERATIONS TO FIRST RELEASE	PRODUCTIONIZING	MAINTENANCE
Purpose: <ul style="list-style-type: none">- Enough well-estimated story cards for first release.- Feasibility ensured.	Purpose: <ul style="list-style-type: none">- Agree on date and stories for first release.	Purpose: <ul style="list-style-type: none">- Implement a tested system ready for release.	Purpose: <ul style="list-style-type: none">- Operational deployment	Purpose: <ul style="list-style-type: none">- Enhance, fix.- Build major releases
Activities: <ul style="list-style-type: none">- prototypes- exploratory proof of technology programming- story card writing and estimating	Activities: <ul style="list-style-type: none">- Release Planning Game- story card writing and estimating	Activities: <ul style="list-style-type: none">- testing and programming- Iteration Planning Game- task writing and estimating	Activities: <ul style="list-style-type: none">- documentation- training- marketing	Activities: <ul style="list-style-type: none">- May include these phases again, for incremental releases.

Source: T1-Chap8

XP Core Practices (12)

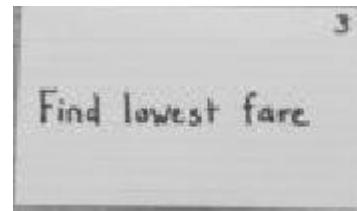
1. Planning Game
2. Small, Frequent Releases
3. System Metaphors
4. Simple Design
5. Testing
6. Frequent Refactoring
7. Pair Programming
8. Team Code Ownership
9. Continuous Integration
10. Sustainable Pace
11. Whole Team Work together
12. Coding Standards

Pitfalls of XP

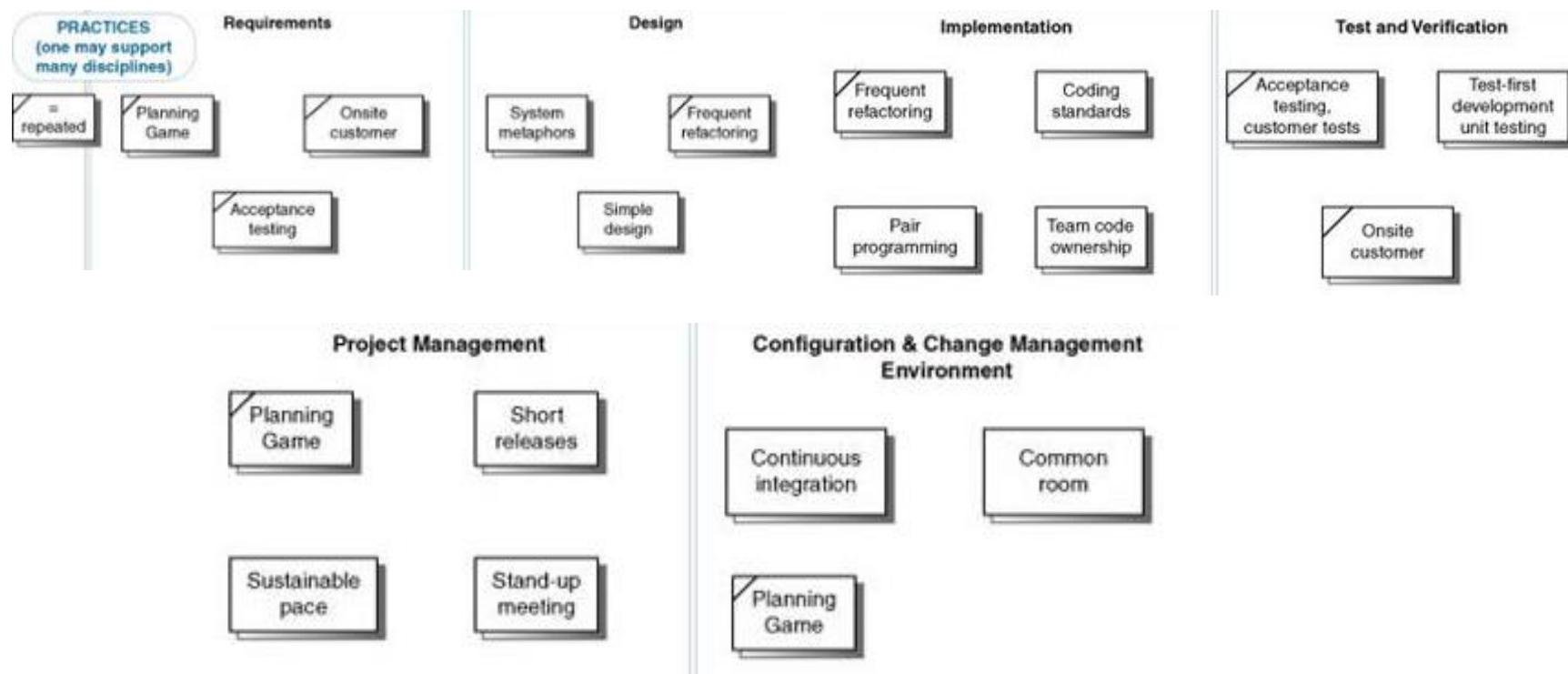
- Requires presence of Onsite Customer or his/her Proxy
- Relies on Informal oral understanding – difficult to ramp-up to speed new members or in large projects
- XP practices are highly interdependent and tightly coupled—we can't be selective in any
- No 'Standard' means of documenting design
- Pair programming may not be favoured by all – *programmers are loners!*
- Lack of Focus on Architecture (relies on simple, if not fragile design and refactoring)

Extreme Programming (XP) Work Products

- Minimalist approach towards Requirements Gathering: Story Cards (sketches, visuals, post-it notes,...) representing *Features* (*not Use-cases/Scenarios*) – just cue-cards for discussion
- Task-list: containing Stories gathered for the Iteration (Task-card); Task-effort ~ 1-2 days
- Visible Wall Graphs for all to communicate with each other— progress of Stories, Test-cases, etc. using Simple Metrics



XP Practices



Source: T1-Chap8

XP – Other Common Practices/Values

- Embrace Change with Onsite Customer
- Volunteering rather than by Assignment
- Light Modeling (Just enough to get stared)
- Minimal Documentation
- Daily Metrics (few vitals) for Progress Tracking
- Incremental Infrastructure (no upfront investment)
- Daily Standup Meetings
- Simplicity – “Do the simplest thing that could possibly work.”
- Communication promoted through Pair-programming

“The practices are what you do. The values are how you decide if you are doing it right.”

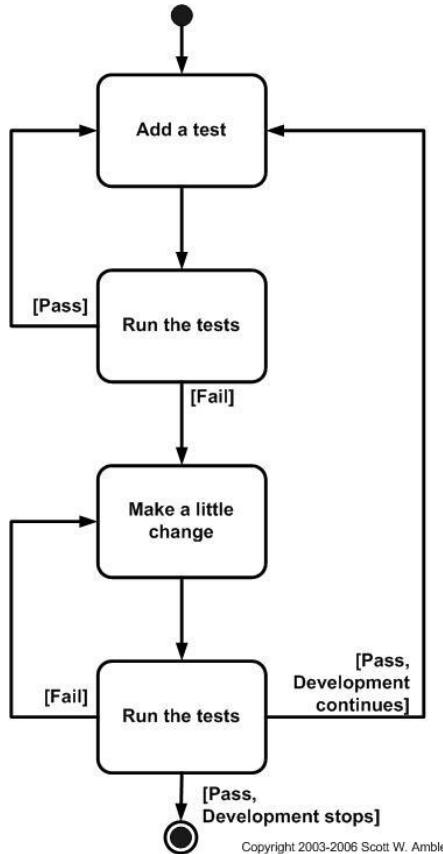
Test Driven Development (TDD): A Test-before-Code XP Practice



Test-Driven Development

- Writing Test-cases before Coding (Unit Tests are written before writing the Code to be tested)
 - Adoption of Short Iterative Development Cycle & Automated-Test Suites
 - Eliminates “coder bias”
-
- “test with a purpose” - know why you are testing something and to what level it needs to be tested
 - “achieve 100% coverage test” – every single line of code is tested
 - “well-written unit-tests provide working specification of functional code” (code ~ documentation, tests ~ specifications)
 - “proxies”: (unit-tests ~ design specifications, acceptance-tests ~ requirements)

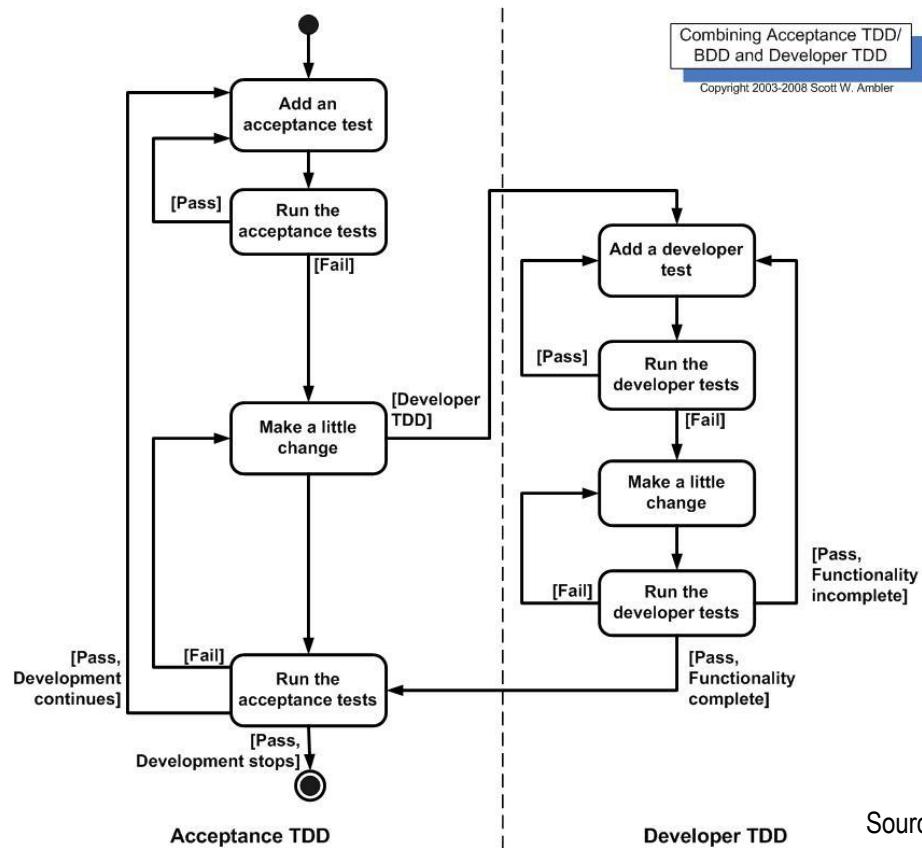
Test-Driven Development (TDD) = Test-First Development (TFD) + Refactoring



- Evolutionary Approach to Development
- Write a Test-case first that fails before you write new functional Code; Coding evolves to fulfil those Test-cases, and then Refactor the Code
- Is TDD an Agile Requirements Capture technique or Programming Technique (than a Validation technique)?
- A Way to arrive at Clean Specifications!
- Does NOT replace traditional Testing, ensures effective Unit Testing

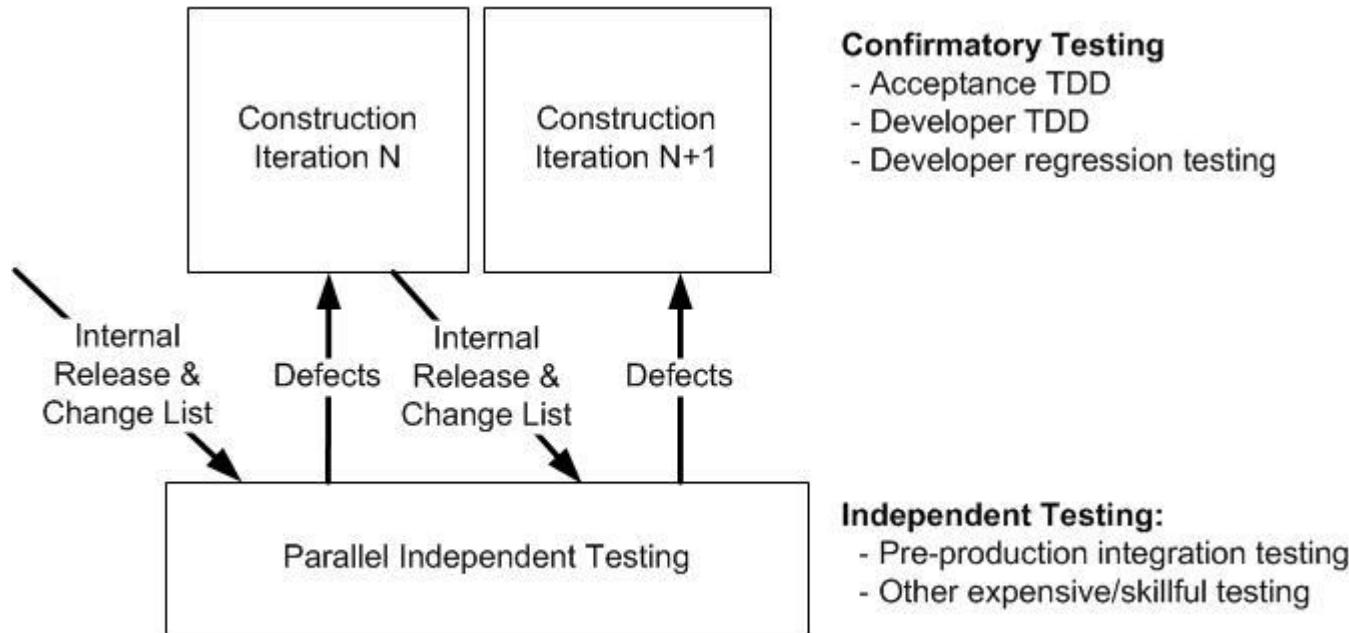
[CUnit](#)
[DUnit \(Delphi\)](#)
[DBFit](#)
[DBUnit](#)
[DocTest \(Python\)](#)
[GoogleTest](#)
[HTMLUnit](#)
[HTTPUnit](#)
[JMock](#)
[JUnit](#)
[Moq](#)
[NDbUnit](#)
[NUnit](#)
[JUnit](#)
[PHPUnit](#)
[PyUnit \(Python\)](#)
[SimpleTest](#)
[TestNG](#)
[TestOoB \(Python\)](#)
[Test::Unit \(Ruby\)](#)
[VBUnit](#)
[XTUnit](#)
[xUnit.net](#)

TDD: Acceptance TDD, Developer TDD



Source: <http://agiledata.org/essays/tdd.html>

TDD is NOT an End to Itself...



Agile Methodologies - Summary

- SCRUM is a Process in Agile Methodology which is a creative combination of an Iterative and Incremental Methods; it is prescriptive in nature and has well-defined Roles
- XP – set of Practices that take Programming to the Extreme, i.e. just enough, just-in-time with minimalist approach; relies heavily on people
- TDD – an XP technique turns traditional Code Development upside-down, i.e. write test first and then write (just enough) code to fulfil that test and move-on to write the next test, then code, etc.; relies on availability of automated test tools

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

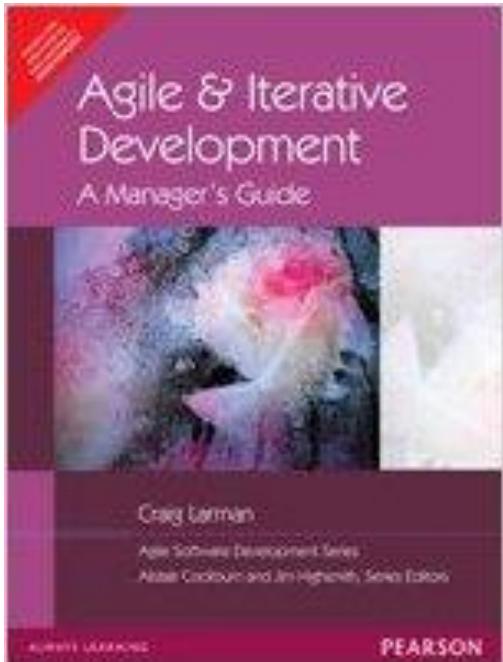


Requirements Management in Agile

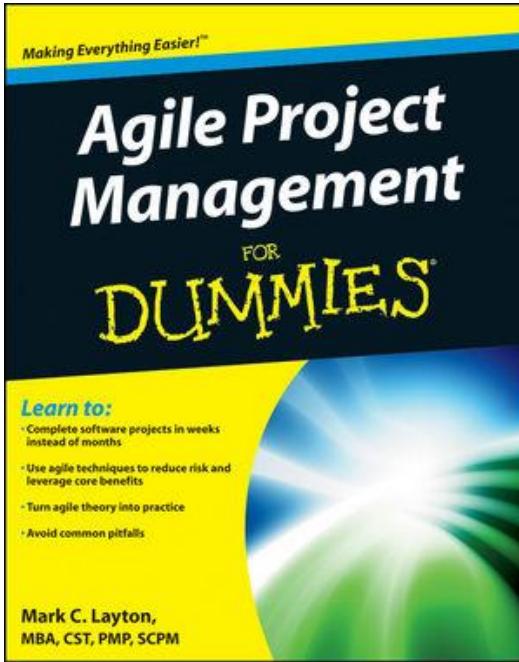
- Prof K G Krishna

Text/Reference Books

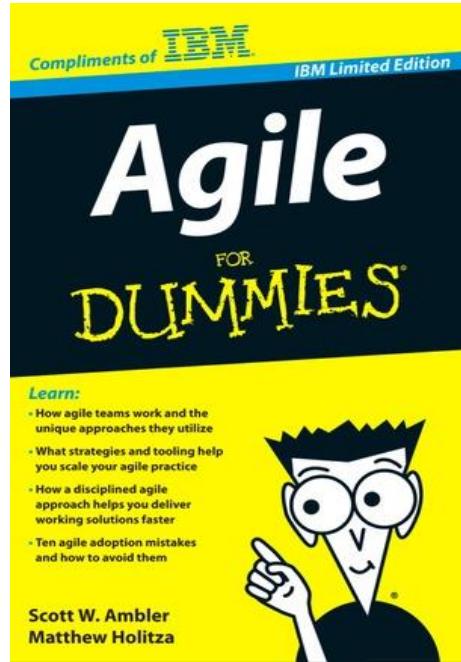
T1



T2



Compliments
of IBM



➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Agile Requirements Management

- Managing Requirements Iteratively
- User Stories as Requirements
- Size/Effort Estimation
- Prioritization Techniques
- Preparing the Product Roadmap

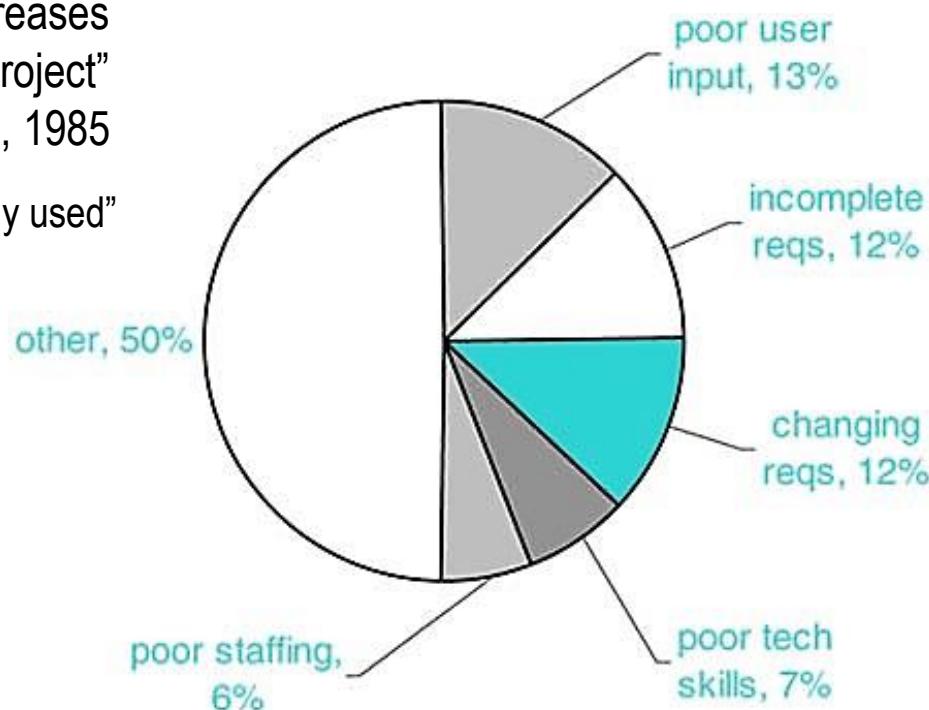
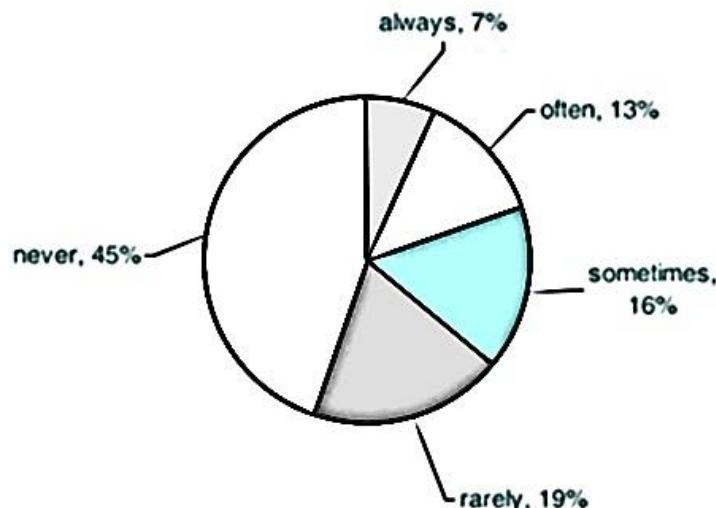


37% of Project Challenges are in Requirements Management

“Cost of fixing a Requirement defect increases non-linearly from early to late in the Project”

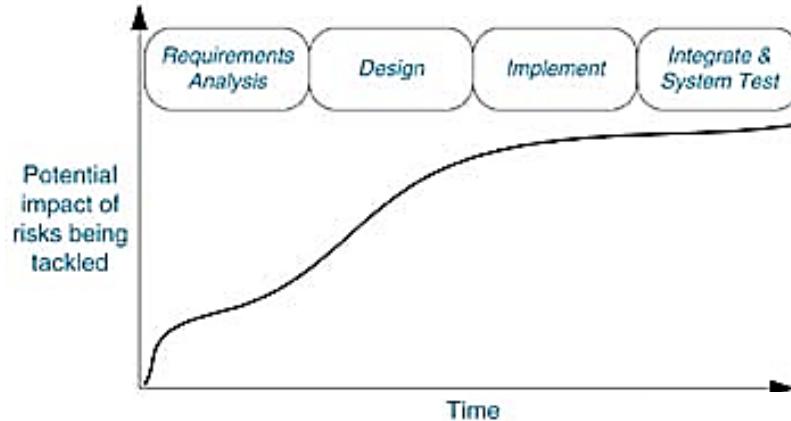
- B. Boehm, 1985

“45% of Features never used and ~20% rarely used”



Source: (T1) / Standish Group Report, 2004

Incremental & Iterative Development (IID) is The Way To Go



In a waterfall lifecycle, high-risk issues such as integration and load test are tackled late.

Risk Profile in Waterfall Model



Risk Profile in IID

In an iterative lifecycle, high-risk issues are tackled early, to drive down the riskiest project elements.

Source: (T1-Chap 5)

Identifying Product Requirements in Agile

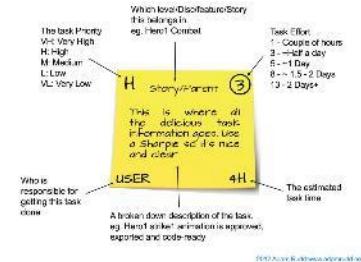
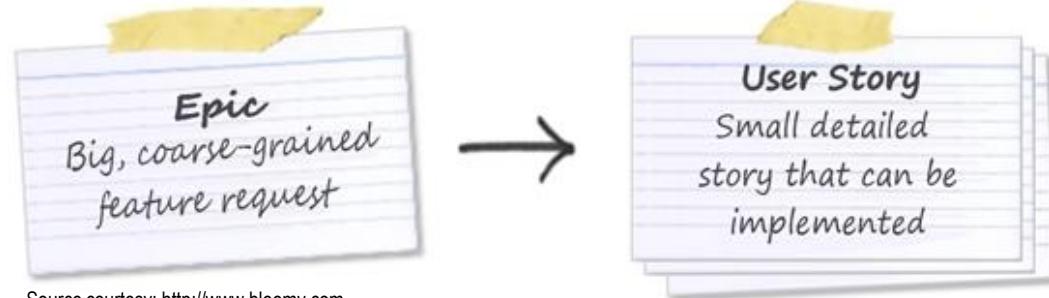
- From the Product Roadmap, arrive at Initial Set of High-priority Requirements
 - Product Roadmap → Requirements → Releases → Sprints
 - Requirements → Logical Groups
 - Decompose Requirements into: (level of detailing depends on early product definition)
 - **Themes**: logical grouping of features into Themes (Requirements at highest level, e.g., Account Info, Transactions, Support functions,...)
 - **Features**: describe capability of the Product (part of the Product, e.g., view balance, pay bills, reset password, transfer money,...)
 - **Epic User Stories**: large set of Requirements that support a Feature containing multiple actions
 - **User Stories**: containing single action enough to start implementation (use-cases, scenarios)
 - **Tasks**: execution steps required to develop a story; breakdown User Story into Tasks during Sprint planning



Source: <http://eleventwenty.com>

Building Product-backlog

- Product-backlog comes to life soon after identification of first Requirement (Theme/Feature/Story)
- User Story - an Expression of Requirement of Business Value
- Group Features (into Themes) by Technical similarity, Usage flow, Business need, etc.
- Use Index cards / Sticky Post-it notes for easy shuffling between Themes, Sprints and Product back-logs
- A Meeting of Stakeholders (Customer) for Identifying and Grouping of Requirements



Size/Effort Estimation in Agile → →

Relative Effort Estimation

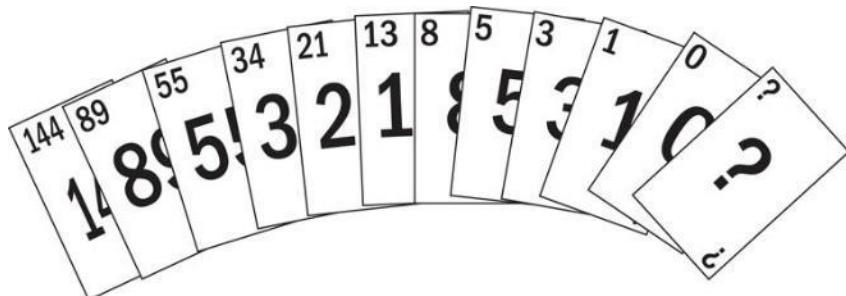
- Estimation & Ordering of Requirements commence soon after they are identified and arranged into logical Groups
- *Effort* in Agile is not exact quantitative estimate, but an assessment of the *ease or difficulty* of implementing the Requirement
- Ordering and Prioritizing is about determining its *value* in relation to other Requirements
- *Value* implies how beneficial (customer value proposition) the Requirement is to the Product (when released to users)
- Ordering of Requirements considers logical dependencies

Relative Scoring of Requirements

- Relative Scoring of Requirements using “Fibonacci Sizing Sequence”
1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
 - The effort-scores of Features (at high-level) during Product Roadmap creation will be high in the range, say 55 to 144
 - When the above are broken down into epic user stories, their scores will be in the range of 13 to 34
 - Upon further break down into low-level User Stories (ready for implementation), their scores should have effort scores between 1 and 8
- Scoring is Relative (Value & Effort)
 - Chose a Requirement that a Project Team can agree has a small value and effort and score it, and use that Requirement as a Benchmark for furthering scoring of other Requirements
 - Use two separate Benchmarks for Value and Effort to calculate Relative Priority

Effort Estimation by “Poker Estimation Game”

- Estimation Poker (aka Planning Poker) is a fun game to determine Story size and build consensus
- Scrum Master acts as a Facilitator and Product Owner provides reads the Story / provide details about the Feature to be estimated
- The Card deck contains cards with numbers of the Fibonacci sequence



Why Fibonacci?

“Fibonacci series represents a set of numbers that we can intuitively distinguish between them as different magnitudes...”??

Value Description

1	equals "very little effort"
2	equals "little effort"
3	equals "very neutral effort"
5	equals "higher effort"
8	equals "very high effort"
13	equals "extremely high effort"

Poker Estimation (by Consensus) contd.,

- Agree on Point-scale of about Six Numbers (representing Story Points)
- Product Owner explains the User Story and provides relevant Information
- Team (Players) briefly discusses the Story and guesses an estimate (Story Points)
- Everyone silently selects one card (they felt represent the 'effort') and lays the card face-down
- Once each Player selects a card, all players turn-over their cards simultaneously
- If the Players have different Story points, it's time for discussion; if the Players do not agree on any one estimate, it's time for Scrum Master to mediate and decide or determine that the User Story needs more detailing
- The above steps are repeated for each User Story to arrive at the collectively agreed Story Point estimates for all
- When number of Stories are large, use **Affinity Estimating** – group Stories of similar affinity (effort value) and apply Poker Estimation to these categories (e.g. *Extra-small, Small, Medium, Large, Extra-large, Epic user story* that is too large to come into the Sprint)

SIZE	POINTS
XtraSmall (XS)	1 pt
Small (S)	2 pts
Medium (M)	3 pts
Large (L)	5 pts
XtraLarge (XL)	8 pts

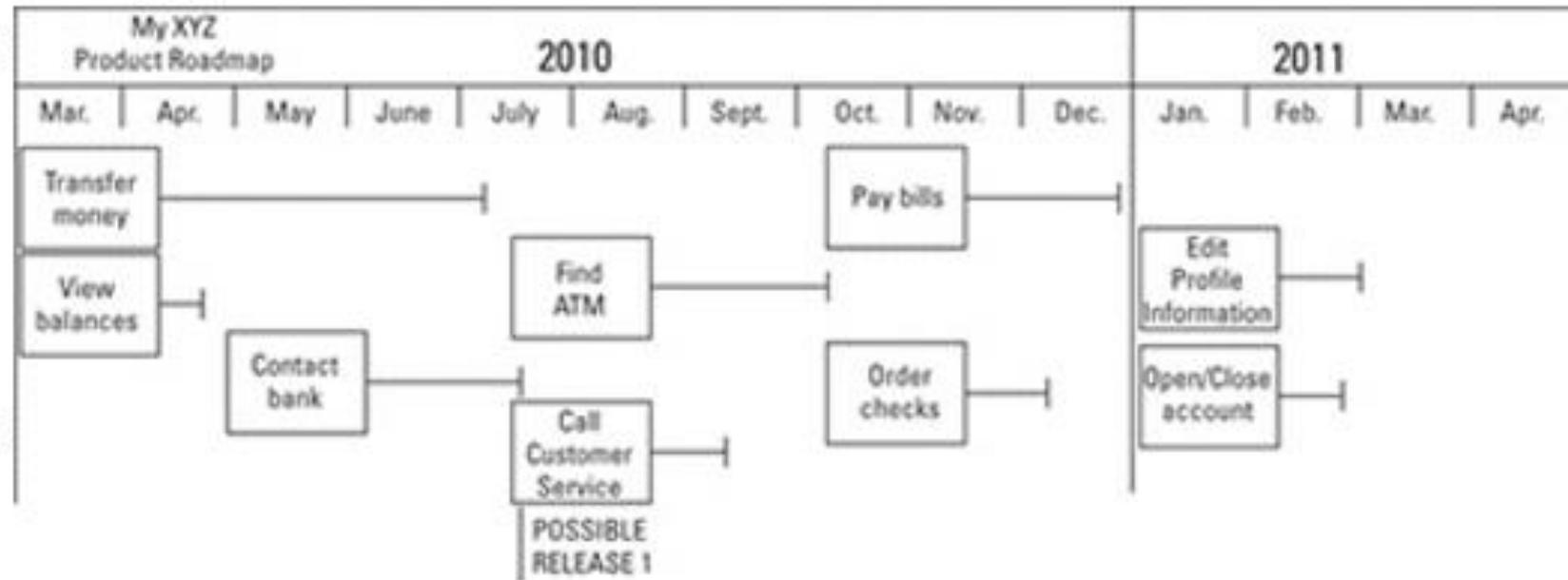
Estimates should be for the total **Done** – Developed, Integrated, Tested and Documented

Relative Prioritizing of Requirements

- After having Effort and Value scores for each Requirement, calculate *Relative Priority* as $Value/Effort$ (and round the value to integer)
- A Requirement with High Value and Low Effort will have High Relative Priority compared to the one with Low Value and High Effort
- Relative Priority is just a mathematical idea to base decisions – however, any other equivalent technique can be used as well
- To determine the Overall Priority answer the questions:
 - What is the Relative Priority (refer the above calculation)
 - What are the Prerequisites for any Requirement
 - What set of Requirements constitute a set for Release (of relative high value)

Build Product Roadmap with Prioritized Requirements

- Build Product-backlog with Feature set, and start arranging as per the Relative Priority computed



Summary: Agile Requirements

- Continuous **Requirements Churn** is the Key Motivation for going Agile
- Requirements are **Evolutionary** in Agile Projects – they continue to Change till the Last Release/Sprint
- Requirements are organized into *Themes* → *Features* → *Epic User Stories* → *User Stories* → *Tasks*
- All Size/Effort Estimations in Agile are **Relative** with baseline Estimation of a small User Story (unit of Requirements Capture/Estimation)
- Estimations are made by **Consensus** (using *Poker Estimation*, *Affinity Estimation*)
- **Relative Prioritization** of Estimates (by Value) helps in building Product Roadmap (→ Product-backlog)
- Requirements are **Managed at every Planning Stage** in Agile – from Product Roadmap (Product-backlog) to Release Planning (Release-backlog) to Sprint Planning (Sprint-backlog)

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

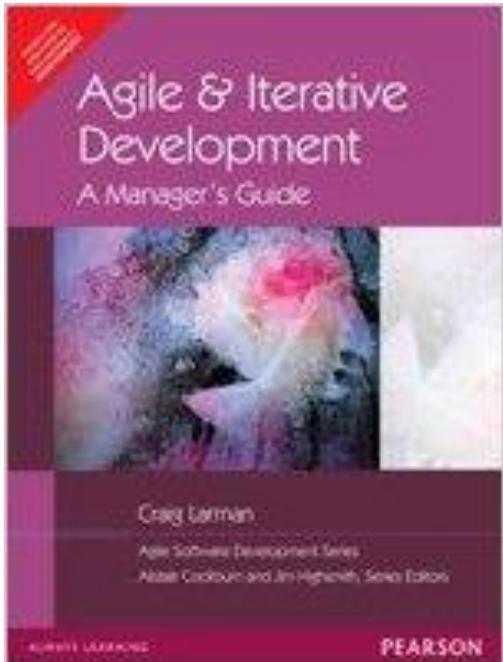


Release Planning in Agile

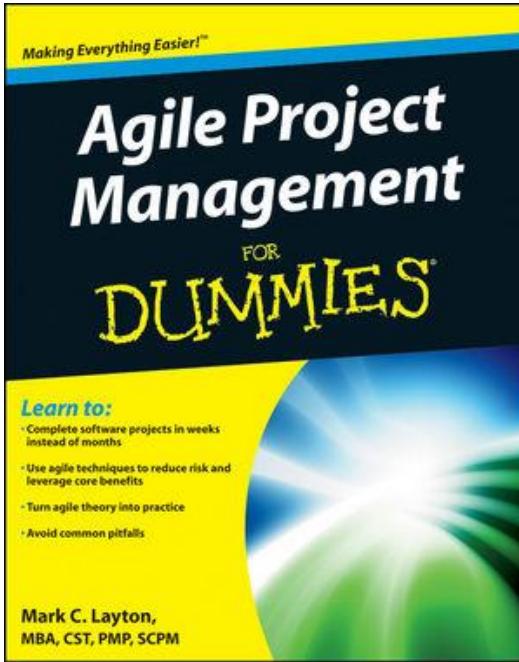
- Prof K G Krishna

Text/Reference Books

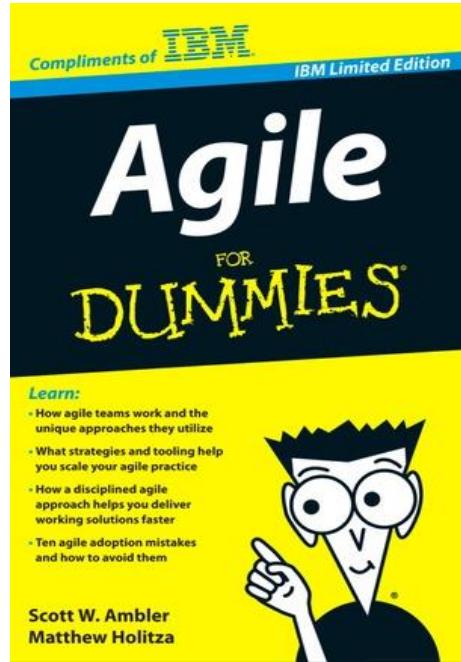
T1



T2



Compliments
of IBM



➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

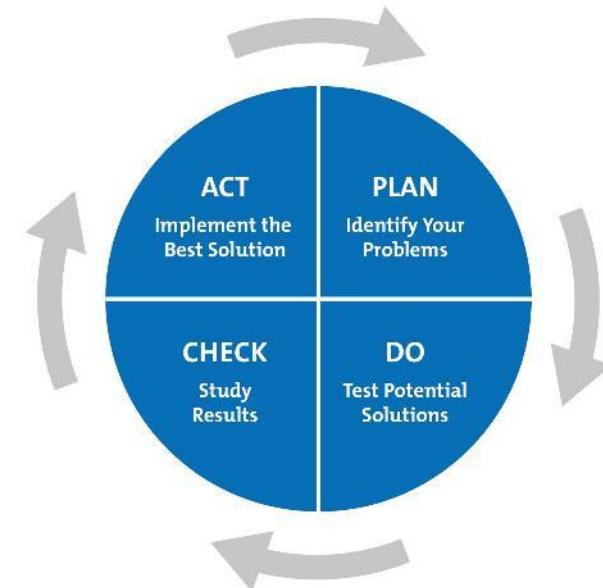
Release Planning in Agile Methods

- Characteristics of Agile Planning
- Stages of Agile Planning
- Release Planning

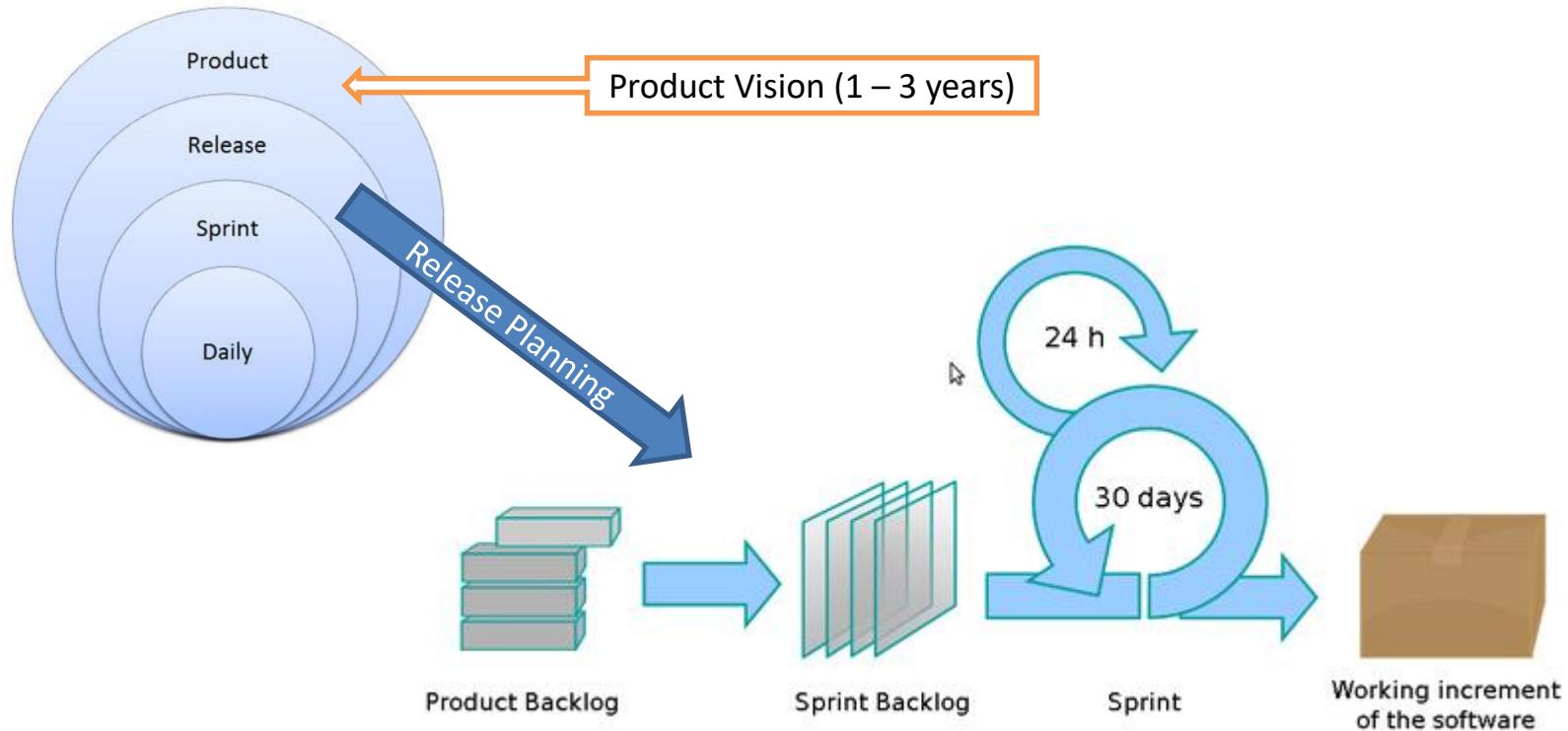
Planning is *Continuous*!

(Deming's Continuous Improvement Cycle)

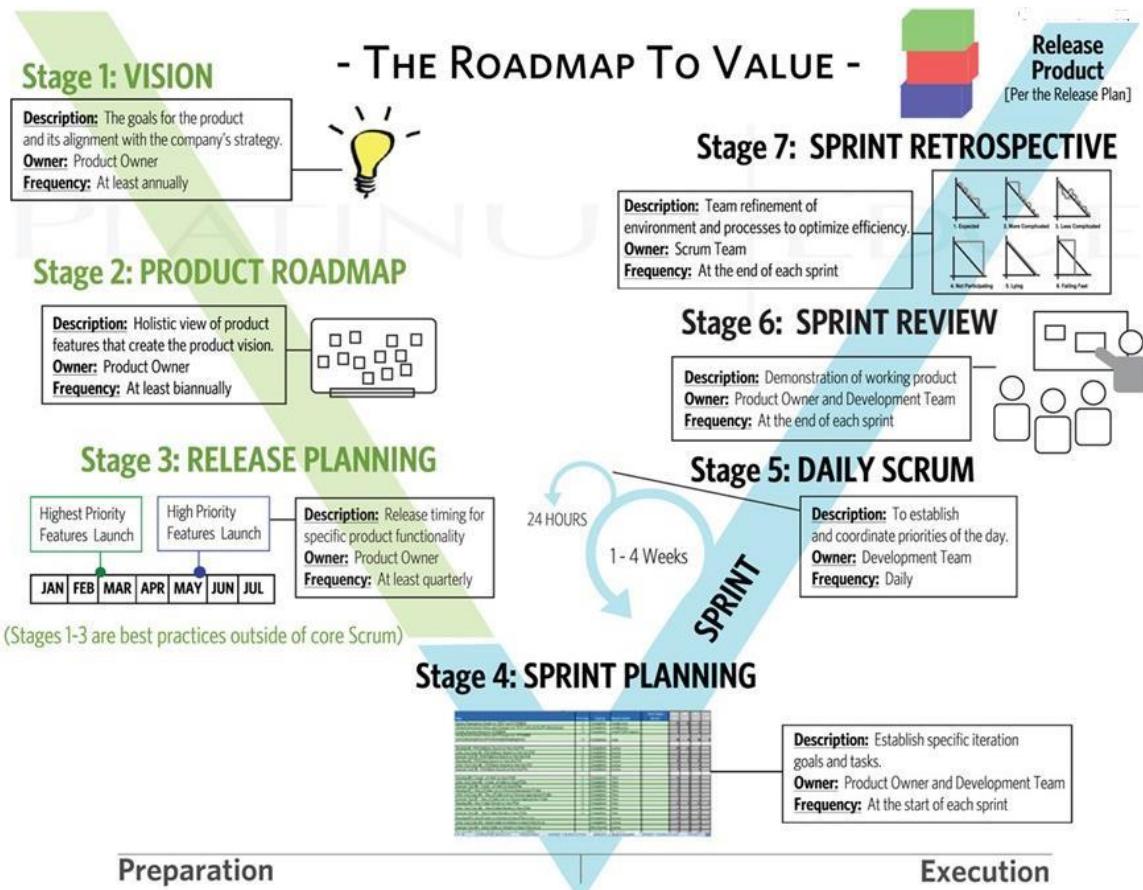
- Planning is NOT a One-time Activity at the Beginning of a Project
- Part of PDCA (Deming's Cycle of Continuous Improvement)
- PDCA vs. “Plan-the-Work, Work-the-Plan” (Waterfall Model)
- Agile Planning
 - Just-In-Time Planning / “Situational Planning”
 - Agile Planning is Continuous throughout the Project



Levels of Planning in Agile



Planning at Every Stage in Agile (SCRUM)



Source: (T2)

Key Characteristics of Agile Planning

- Planning occurs at every Stage
- Planning, like Development is Iterative
- Just-enough and Just-in-Time Planning at every Stage
- Progressive Detailing - start with a broad plan and narrow it progressively
- Prioritizing Value at every Stage: Add High-value Requirements first
- Adapt the Plan after Feedback at every Stage

Plan-Do-Inspect-Adapt

Agile Release Planning Stages in Detail



Stage-1: Defining Product Vision

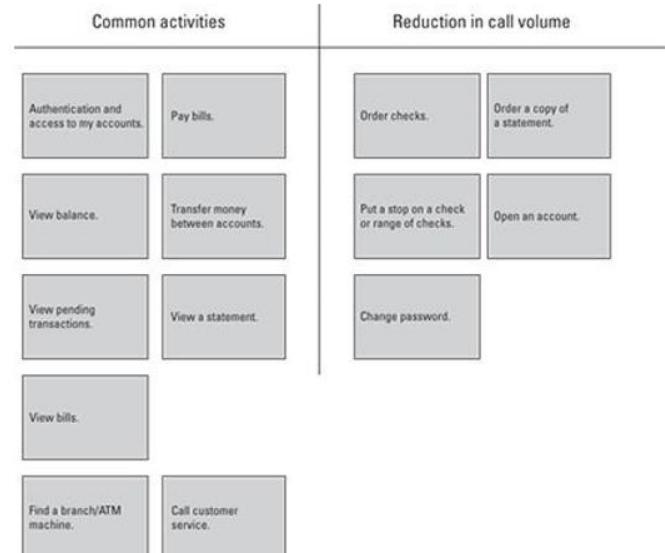
- The Vision Statement (Summary) communicating Product Strategy
 - Product Goals aligned with Strategy
 - Owned by the Product Owner
 - Frequency: annually (minimum)
- Developing the Vision Statement
 - Develop the Product Objective
 - Draft the Vision Statement
 - Validate the Statement with Stakeholders and Revise after Feedback
 - Finalize the Vision Statement
- The Vision Statement must be:
 - Clear with Simple language
 - Non-technical (everyone can understand)
 - Brief (in one or two lines)
 - Internally focused (at development, not a sales pitch)

Vision Statement for Product:	
For:	(Target Customer)
who:	(needs)
the:	(product name)
is a:	(product category)
that:	(product benefit, reason to buy)
Unlike:	(competitors)
our product:	(differentiation/value proposition)

Stage-2: Holistic Product Roadmap

- Derived from Project Vision
- Identify Requirements that Define Product Roadmap
- Arrange Requirements into Logical Groups
- Estimate Effort and Prioritize Requirements
- Set High-level Time-frame for each Group of Requirements
- Beginning of the Creation of Product-Backlog

➔ Update the Product Roadmap throughout the Project
(unlike Project Vision)



Stage-3: Release Planning

- Elaboration of Project Roadmap into Details
- A Release is a *Minimum* set of Marketable Requirements
- Requirements Breakdown Structure (WBS) by granular Decomposition
- “Epic-story ..→ User-stories”
- User-story: Simple Description of Requirements (user-walkthrough or benefit statement)
- Create *Personas* for each Class of Users
- Identify each Story by ID and set a Value (in terms of benefits or priority)
- Estimate Effort for each Story (“story points”)
- Document on Index-cards or Post-it Notes
- Product Owner manages the Stories
- Break Stories further into detailed Features/Tasks for Sprint Planning,

Title	Transfer money between accounts	
As	Carol,	
I want to	review fund levels in my accounts and transfer funds between accounts	
so that	I can complete the transfer and see the new balances in the relevant accounts.	Jennifer
Value	Author	Estimate

Stage-4: Sprint Planning

- A Sprint is a consistent (fixed-length ~ 1-4weeks) Iteration of Time in which Product takes Demonstrable Shape
- Sprint Backlog: List of User-stories (prioritized) with detailed WBS and Time-estimates
- Each Task to be completed in 1-2 days max (no over-committing, reduce Scope if necessary)
- Task Done → Developed, Integrated, Tested and Documented
- Development Team work only on one Requirement (Story/Tasks) at a time
- Only the Development Team can modify the Sprint Backlog
- Each Sprint includes:
 - Sprint Planning (max. 2 hours at the beginning of every week)
 - Daily Scrum Meeting (standup meeting for 15-20mins)
 - Development (bulk of the effort in Sprint)
 - Sprint Review
 - Sprint Retrospective

Stage-5: Daily Scrum

- Each Day can be a Planning/Replanning Day (Daily Scrum meeting)
- Daily Scrum Meeting to be Brief (~15-20mins Standup)
- Scrum Master to facilitate the meeting (review progress, roadblocks,...)
- Participants: Product Owner, Development Team and Scrum Master
- Focus of Meeting: Coordinate/Prioritize (Not to solve Problems)
- Update Sprint Backlog Daily (at the end of the meeting) and make it visible to everyone in the team

To Do	In Progress	Verify	Done
Code the... 9	Test the... 8	Code the... DC 4	Test the... 6
Code the... 2	Code the... 8	Test the... SC 8	Code the... Test the... Test the... Test the... Test the... SC 6
Test the... 8	Test the... 4		

Stage-6: Sprint Review

- Sprint Review Meeting at the end of each Sprint to review and demonstrate User-stories that were completed during the Sprint
- Entire Team (Product Owner, Development Team, Stakeholders and Scrum Master) participates
- Product Owner confirms Status of Completion of Sprint (ready for Release of partial-working product)
- Invites Feedback from all Stakeholders
- Scrum Master to update Product-backlog for the next Sprint Planning

Stage-7: Sprint Retrospective

- Post Sprint Meeting (Scrum Master, Development Team and Product Owner) to discuss *the experience* of the Sprint – What went right and what went wrong
- Focus is on Continuous Improvement of the *Process* to improve *Efficiency* and *Velocity* of throughput
- Adapt Scrum Processes to improve morale of Team and their Work-life balance
- Lasting for ~45mins maximum for every week of the Sprint
- Opportunity to *Inspect and Adapt* (Plan-Do-Inspect-Adapt) Scrum Process

Preparing for Release

- End of every Sprint to be *Working and Demonstrable* Product
- A Sprint outcome can be a Release Sprint meant for Customers
- Sprint-backlog Items in a Release Sprint might include:
 - Creating User Documentation for the just finished Release
 - Testing of Key Non-Functional Requirements (Performance, Security, Load balancing,...)
 - Compliance with mandatory Organizational or Regulatory Procedures
 - Integrating with existing Organization's Enterprise Systems
 - Preparing Deployment Package (Installation scripts, etc)
 - Preparing a Release Note
- Note that Development for Regular Sprint is different that of Release Sprint
- Sprint Review meeting for Release to include Customer and Key Stakeholders from Marketing and Operations as well

Summary: Release Planning

- Planning is Continuous in Agile – at every Stage in the Scrum Process (Release Planning, Daily Scrum, Sprint Review)
- Each Release may span one or more Sprints
- Agile Planning is planning for a pre-determined number of Releases to Customers
- Planning for Release involves more Tasks (Sprint-backlog) than for regular Sprint
- Sprint Retrospective Meeting identifies Opportunities for Improvement in the Scrum Process and implements them before the next Sprint cycle

**More Meetings, More Sharing of Information/Feedback,
Near-Real-Time Visibility into the Product, and finally
'Unsurprising' and Acceptable Product Release(s)!**

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners



Iteration Planning in Agile

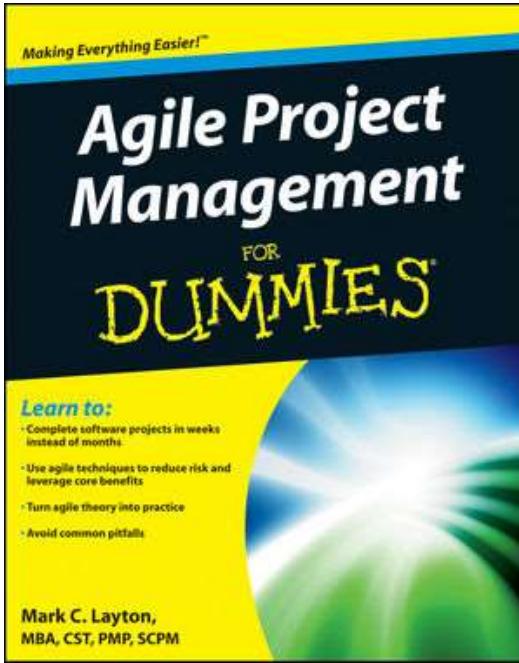
- Prof K G Krishna

Text/Reference Books

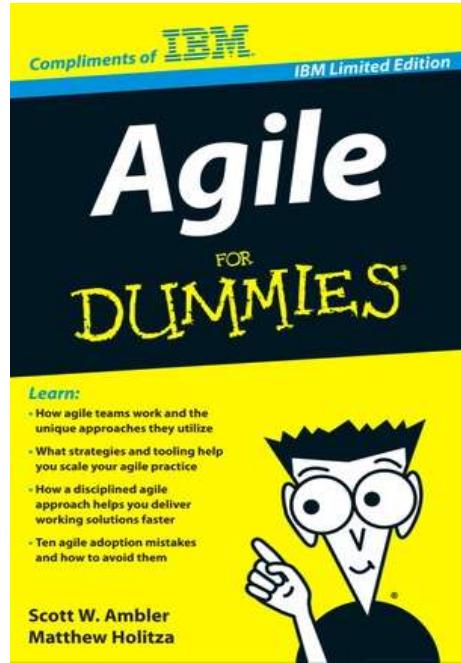
T1



T2



Compliments
of IBM



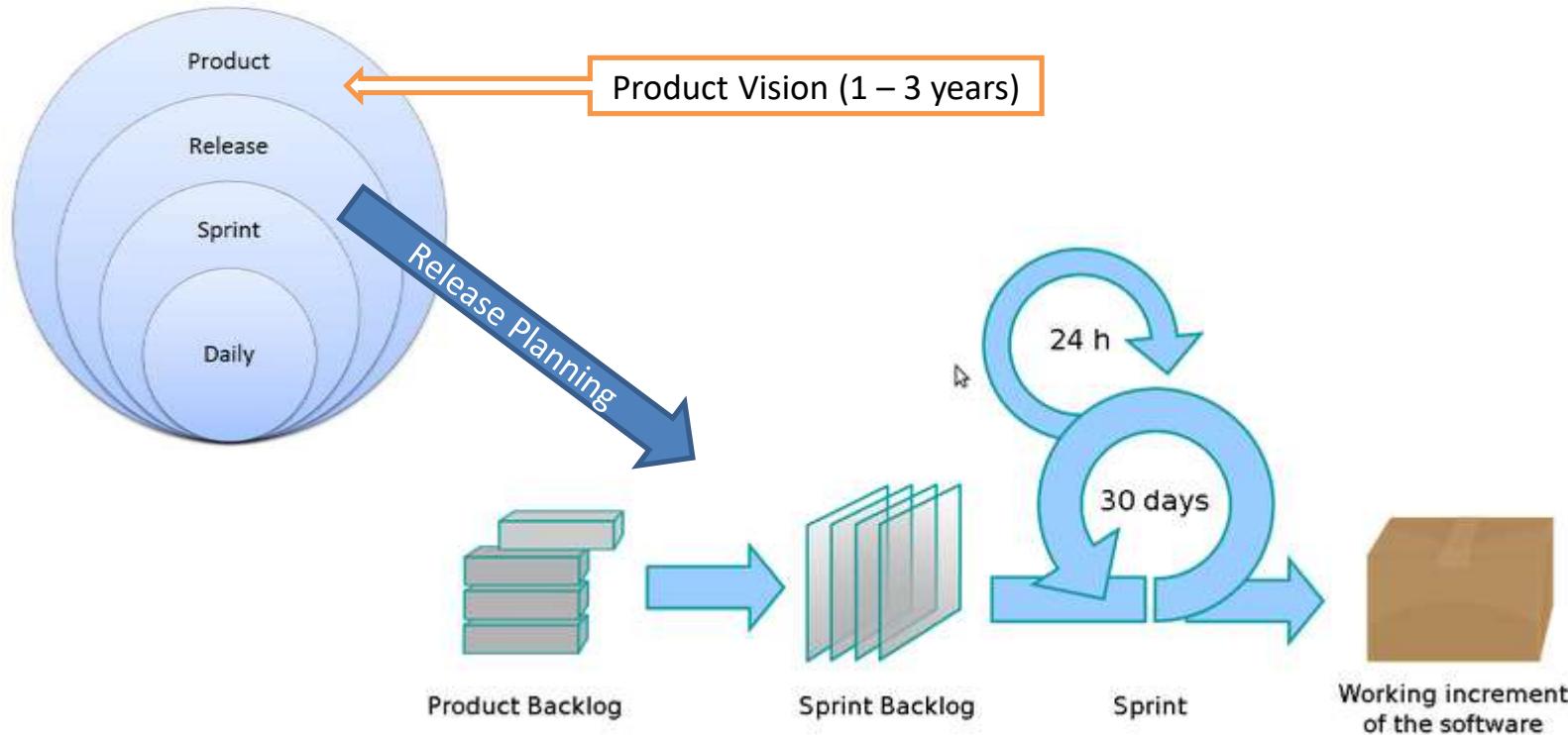
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

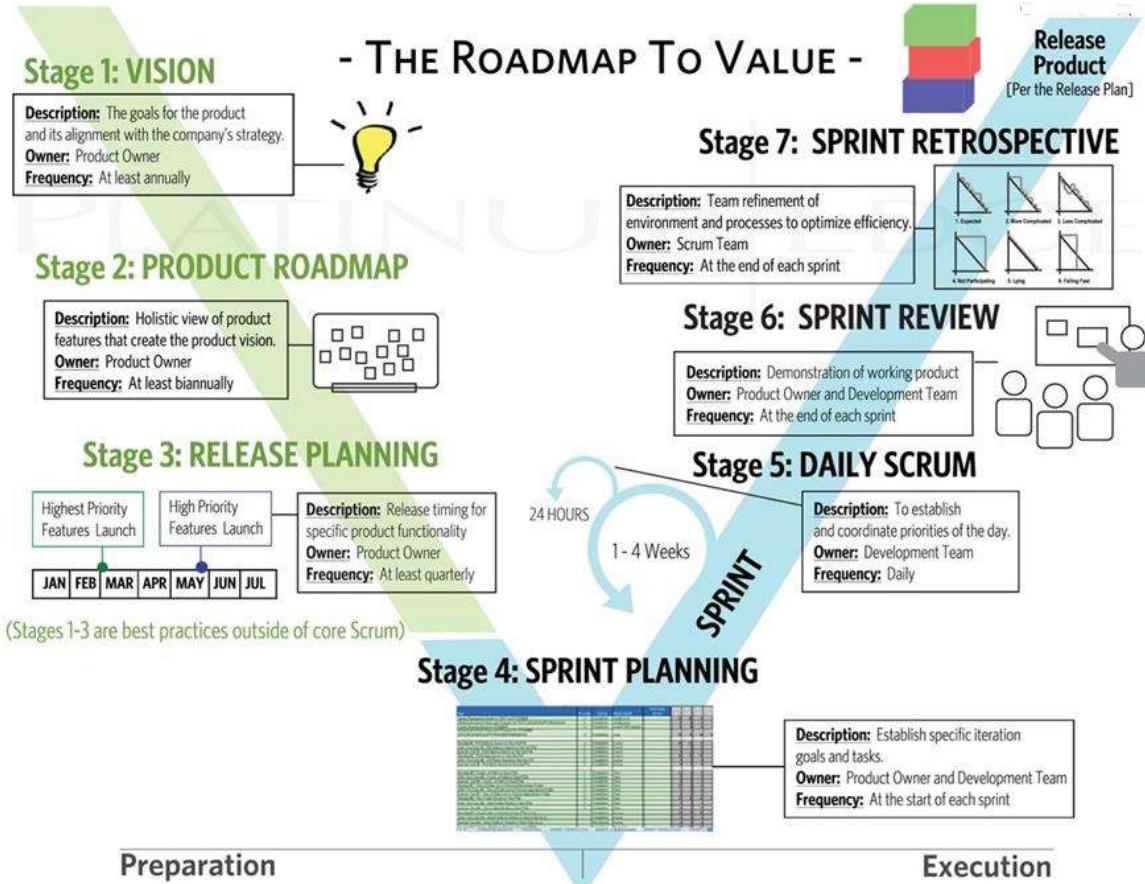
Iteration Planning in Agile

- Sprint as an Iteration
- Velocity based Planning
- Capacity based Planning
- Sprint Planning/Backlog

Levels of Planning in Agile

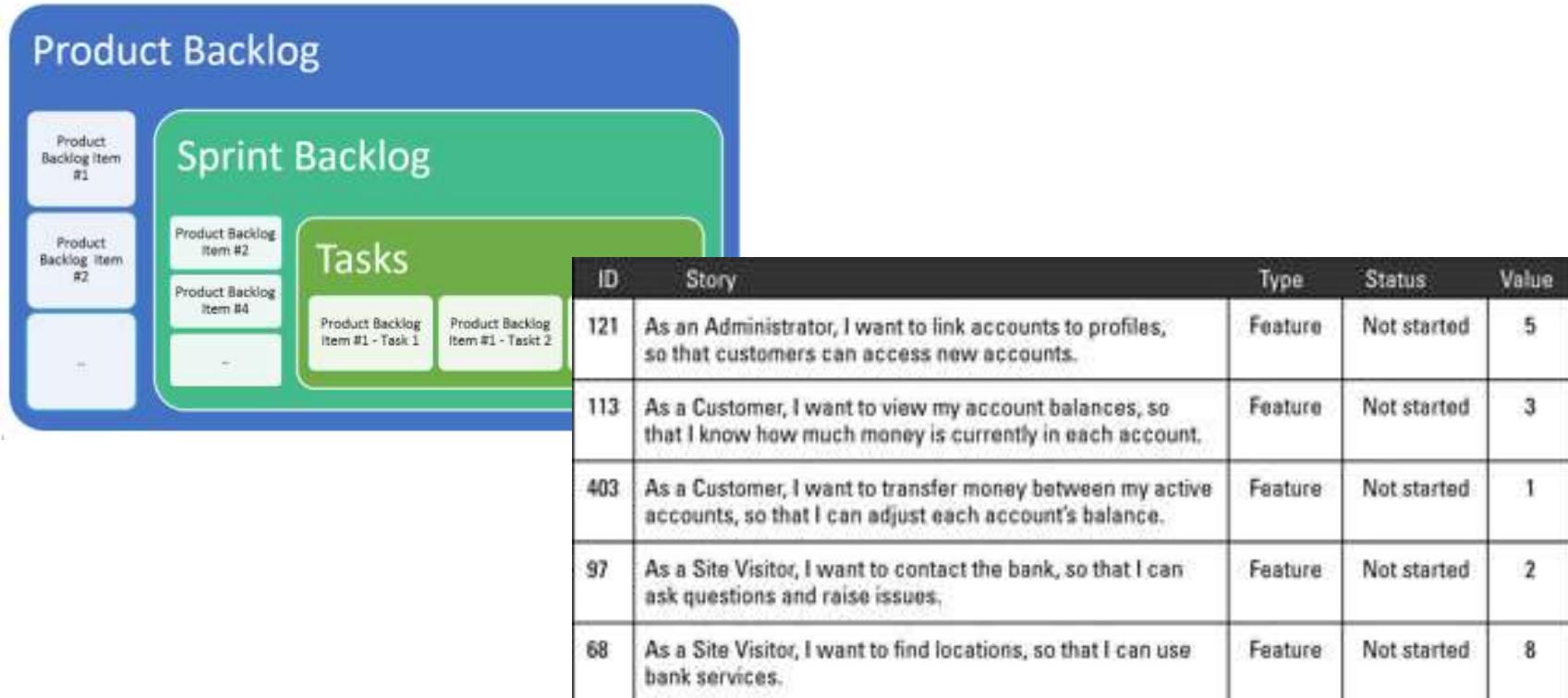


Planning at Every Stage in Agile (SCRUM)



Source: (T2)

Product-backlog vs. Sprint-backlog vs. Tasks/User Stories



User Stories = Requirements for the Developer/Tester

1. Identify the project stakeholders.
2. Identify who will use the product.
3. Working with the stakeholders, write down the requirements that the product will need in a story format (User Story Cards)

Title	Transfer money between accounts
As	Carol.
I want to	review fund levels in my accounts and transfer funds between accounts
so that	I can complete the transfer and see the new balances in the relevant accounts.
Value	Jennifer
Author	
Estimate	

Title	
As	<personal/user>
I want to	<action>
so that	<benefit>
Value	
Author	
Estimate	

As a customer, I want to be able to search for flights between two cities to see which ones have the best price and route.
Estimate: 1.0 points
Priority: 2 - High

→ The best changes often come at the end of the project, when you know the most about your product and its customers

Sprint Planning (work backwards from Goal)

1. Discuss and set a Sprint Goal.
2. Review the User Stories from the Product-backlog that support the Sprint Goal and revisit their relative estimates
3. Determine what the team can commit (Stories) to in the current Sprint.
4. Create Task-list for each Committed Story
5. Always Plan One Sprint at a Time (Just-in-Time Planning)

Example: Sprint Goal

“As a mobile banking customer, I want to log in to my account so that I can view my account balances and pending and prior transactions.”

Sprint Stories (for the above Goal)

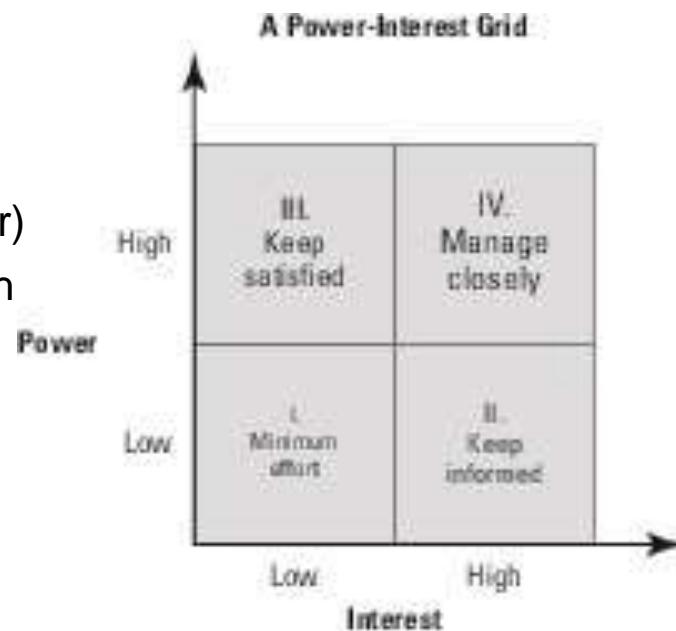
- Log in and access my accounts.
- View account balances.
- View pending transactions.
- View prior transactions

Tasks (for the above Stories)

- ✓ Create an authentication screen for a username and password, with a Submit button.
- ✓ Create an error screen for the user to reenter credentials.
- ✓ Create a logged-in screen (includes list of accounts — to be completed in next user story).
- ✓ Using authentication code from the online banking application, rewrite code for an iPhone/iPad application.
- ✓ Create calls to the database to verify the username and password. Refactor code for mobile devices.

Who are the *Stakeholders* to Contribute User Stories?

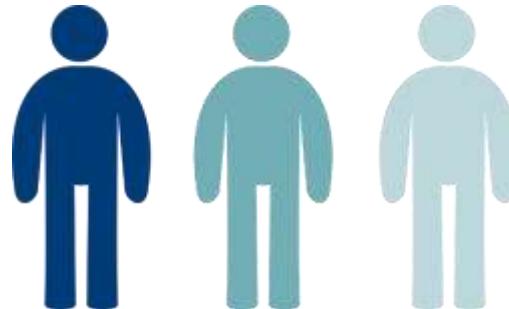
- Who Interact with End-users / Customers on regular basis (Customer Service, Marketing, Sales,...)
- Business Domain Experts
- Actual End-users
- Technical Experts (who developed such Products earlier)
- Technical Experts (when the Product need interface with other systems)
- ...



Source: (T2-Chap8)

Define Personas (Classes of End-users)

- Persona = Representative of End-user Group (with an identified set of similar profile, demographics, behaviours and attitudes)
- Business Domain Experts help identify Personas
- Identify Patterns-of-use through Field Study (Ethnography)
- Personas are key part of Product Vision Statement
- Ensure Manageable Personas (not too many, not too few)
- Not only existing Customers, but also Potential Customers



Source: (T2-Chap8)

Velocity / Capacity Planning →→

Creating Tasks for Sprint-backlog

1. Create the sprint backlog tasks associated with each user story. Break the user stories into discrete individual tasks (can be completed within few hours or less than a day) and allocate a number of hours to each task. Make sure that the tasks encompass each part of the definition of *done*: developed, integrated, tested, and documented.
2. Ensure no over-committing occurs at the beginning of a sprint, especially in the project's first few sprints. The development team should target being able to complete a task in a day or less, for a couple reasons: Short-term goals promote productivity; and a short timeframe brings problems to the forefront quickly. If a team member is working on a task for more than a day or two, that task or that team member may need special attention. If the tasks exceed the hours available, seek the project owner's advice on which user stories to remove from this sprint.
3. Each member chooses a first task to accomplish. Development team members should work on only one task on one user story at a time to enable swarming — the practice of having the whole development team work on one requirement until it's completed. Swarming can be a very efficient way to complete work in a short amount of time.

Velocity Planning (of Sprint)

- Scrum teams use velocity to plan how much work they can take on in a release and sprint. Velocity is the sum of all user story points completed within a sprint. So, if a scrum team completed six user stories during its first sprint with sizes 8, 5, 5, 3, 2, 1, their velocity for the first sprint is 24. The scrum team would plan its second sprint keeping in mind that it completed 24 story points during the first sprint.
- After multiple sprints, scrum teams can use their running average velocity as an input to determine how much work they can take on in a sprint, as well as to extrapolate their release schedule by dividing the total number of story points in the release by their average velocity.

Capacity Planning in Scrum

- In Agile, the Team (not Individual) is the persistent Group (Unit) which has a combination of skills required to do the project
- **Velocity = Amount of work (No. of Story Points) a Team can do in one Sprint**
- **Capacity Planning = Estimating/Calculating the Capacity of Agile Team**
- Units of Measurement: Story Points, Person-Hours (recommended)
- Number of workdays in the period (at five days per week)
- Factors to consider calculating Capacity (as FTE):
 - Number of Team members; Known meetings or activities which involve the whole team, during which no one can contribute hands-on work; Planned time off for each person in the period; Fractional availability of each person for work when not in known meetings
- State Assumptions clearly (Avg complexity of Work, Specialized Task if any)
- Capacity Planning (Team-oriented Estimation) is effective so long as there is no dominance of Specialized Tasks

Capacity Planning (Be *Realistic!*)

Formula = (Total Sprint Hours - Known Spent Hours) * Effective Hours on Task = capacity

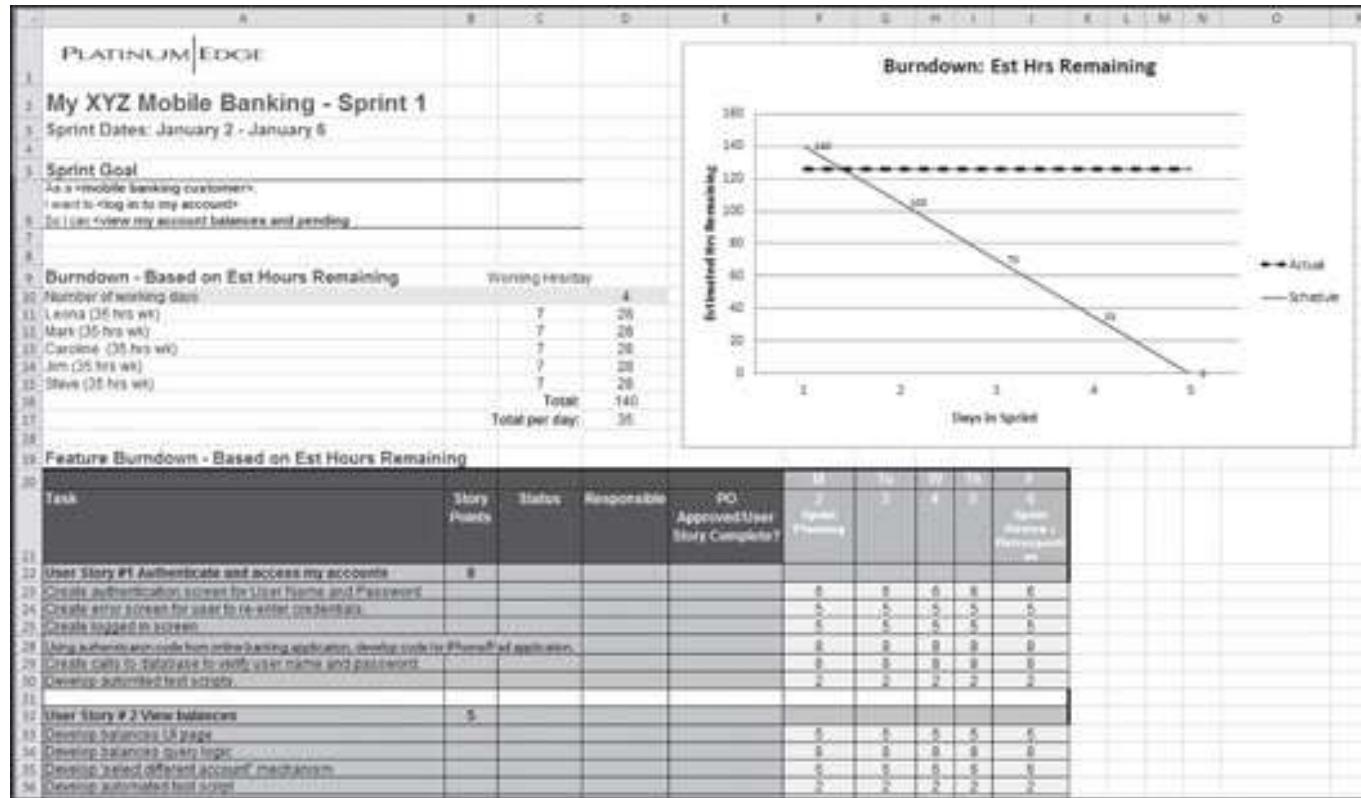
Where:

Total Sprint Hours = Sprint duration (weeks) * Work hours per week

Known Spent Hours = Hours needed for the ceremonies, vacation, statutory holidays, planned training days, recurrent meetings or other things

Effective hours on task = Compensation factor for actual effective hours in a day (factors in e-mail, coffee or bathroom breaks, impromptu hallway banter)

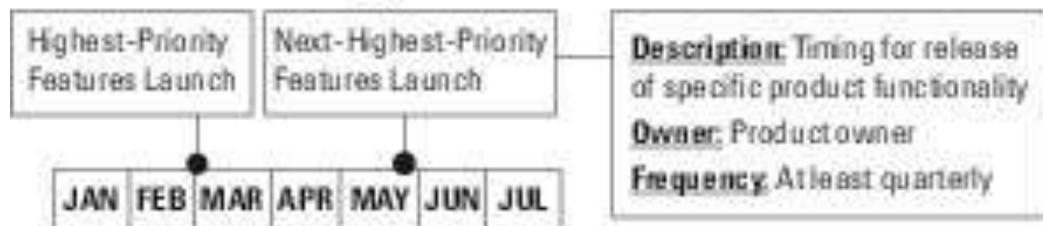
Sprint-backlog can be an Excel Sheet



Release Sprint Planning →→

Release Planning

Stage 3: RELEASE PLANNING



(Stages 1-3 are common practices outside of scrum)

Release Goal: Enable customers to access, view, and transact against their active accounts
Release Date: March 31, 2021

Product Backlog



US = User Story

r = optional release sprint

2 weeks 2 weeks 2 weeks 2 weeks 3 weeks

↑

Source: (T2-Chap8)

No Separate ‘Release-backlog’ Recommended

- Not all agile projects use release planning. Some scrum teams release functionality for customer use with every sprint, or even every day. The development team, product, organization, customers, stakeholders, and the project’s technological complexity can all help determine your approach to product releases.
 - Don’t create a new, separate backlog during release planning. The task is unnecessary and reduces the product owner’s flexibility. Prioritizing the existing product backlog based on the release goal is sufficient and enables the product owner to have the latest information when he or she commits to the scope during sprint planning.
- ➔ The product backlog and release plan are some of the most important communication channels between the product owner and the development team.

Release Sprint? Consider Tasks Too Heavy for one Regular Sprint...

- Some tasks, such as security testing or load testing a software project, can't be completed within a sprint, because the security or load testing environments take time to set up and request. Although release sprints allow scrum teams to plan for these types of activities, doing so is an anti-pattern, or the opposite of being agile. Your goal should be to complete all work required for functionality to be shippable at the end of each sprint.
- Some project teams add a release sprint to some releases to conduct activities that are unrelated to product development but necessary to release the product to customers. If you need a release sprint, be sure to factor that into the date you choose.

Release Plan, Release Schedule

- The Product-backlog and Release Plan are some of the most important communication channels between the product owner and the development team
- The Release Plan contains a Release Schedule for a specific set of Features. The Product Owner creates a Release Plan at the start of each Release. Creating a release plan involves:
 1. Establish the release goal. The release goal is an overall business goal for the product features in your release.
 2. Identify a target release date. Some scrum teams determine release dates based on the completion of functionality; others may have hard dates, such as March 31 or September 1.
 3. Review the product backlog and the product roadmap to determine the highest-priority user stories that support your release goal (the minimum marketable features). These user stories will make up your first release.
 4. Refine the user stories in your release goal. During release planning, dependencies, gaps, or new details are often identified that affect estimates and prioritization. This is the time to make sure the portion of the product-backlog supporting your release is sized appropriately. The development team helps the product owner by updating estimates for any added or revised user stories, and commits to the release goal and scope with the product owner.
 5. Estimate the number of sprints needed, based on the scrum team's velocity.
 6. Identify work necessary to release that can't be completed within a sprint. Plan a release sprint, if necessary, and determine how long it should be.

Just-in-Time Planning of Releases

- Use Just-in-Time Planning, Do not get into microscopic detailing early: commit to the plan for the first release, but anything beyond the first release is tentative (subject to change).
- It's a good idea to achieve releases with about 80 percent of the user stories, using the final 20 percent to add robust features that will meet the release goal (to add to “wow” factor)

Summary: Iteration Planning

- Sprint is a fixed-duration (~30 days) Iteration of Development Activity in the SCRUM Process
- Planning starts with Product Visioning → Product-backlog → Sprint-backlog
- Sprint Planning to involve all Stakeholders
- Identify Personas for Requirements Analysis (creation of Product/Sprint-backlog)
- Story is the Unit of Requirements in Sprint
- Releases (to Customer) can be planned as part of Sprint planning (Development Sprint vs. Release Sprint); no separate Release-backlog may be required
- Estimation/Capacity is Team-centric (not Individual) – Velocity/Capacity Planning

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

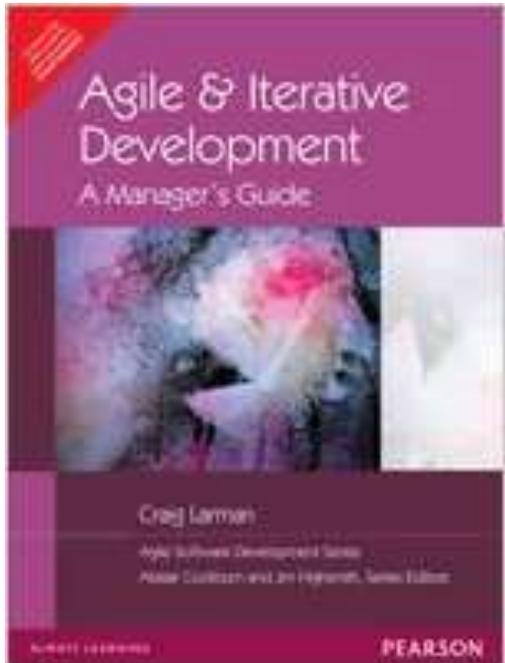


Executing a Sprint

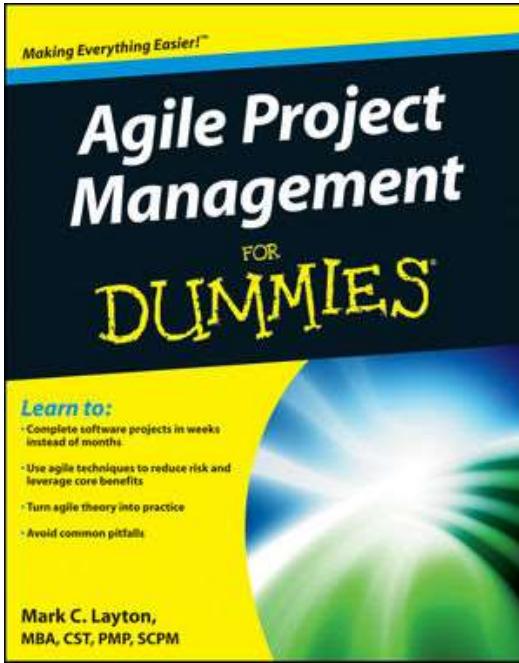
- Prof K G Krishna

Text/Reference Books

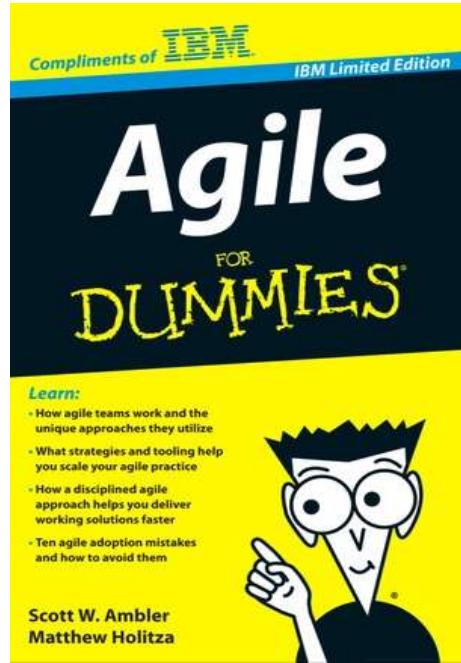
T1



T2



Compliments
of IBM



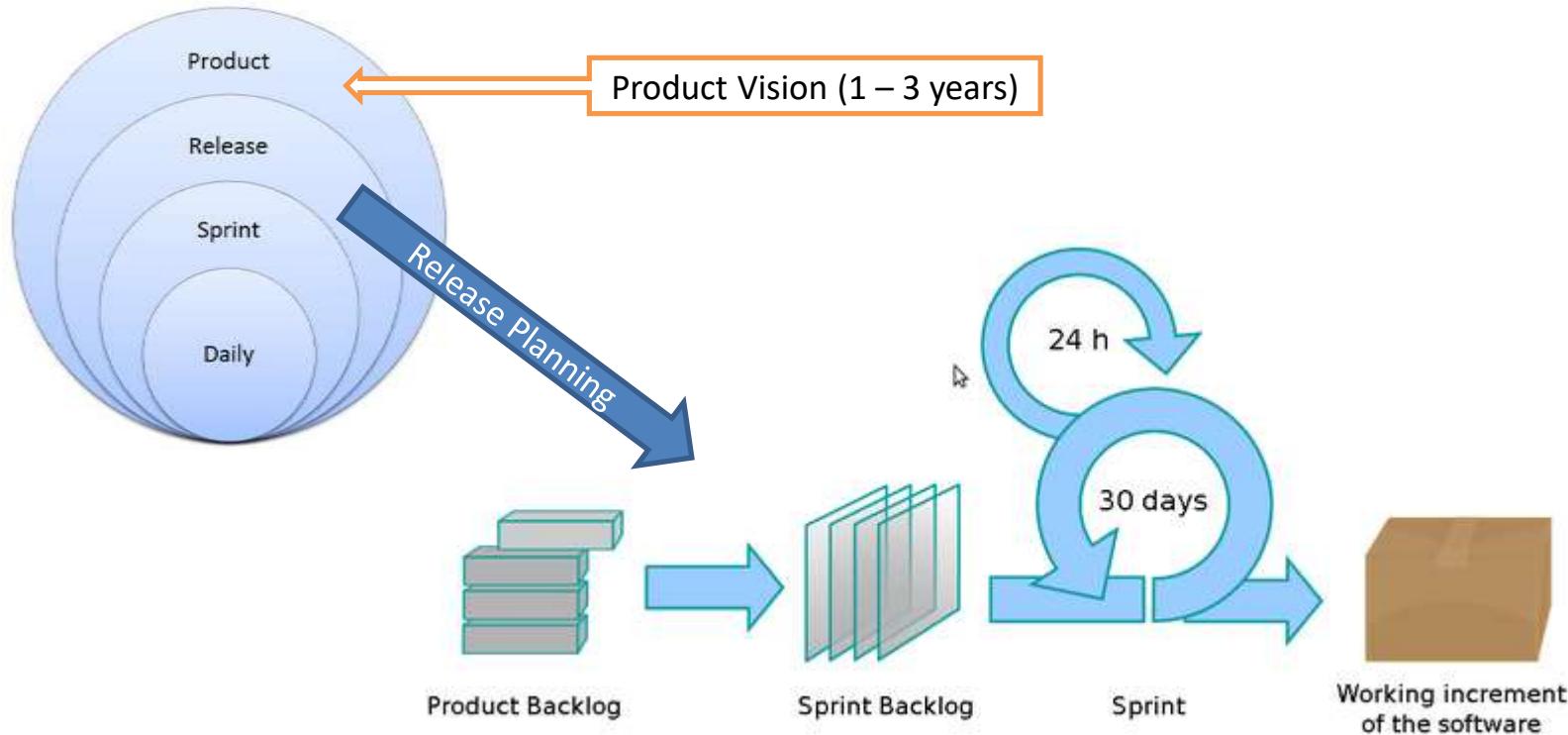
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

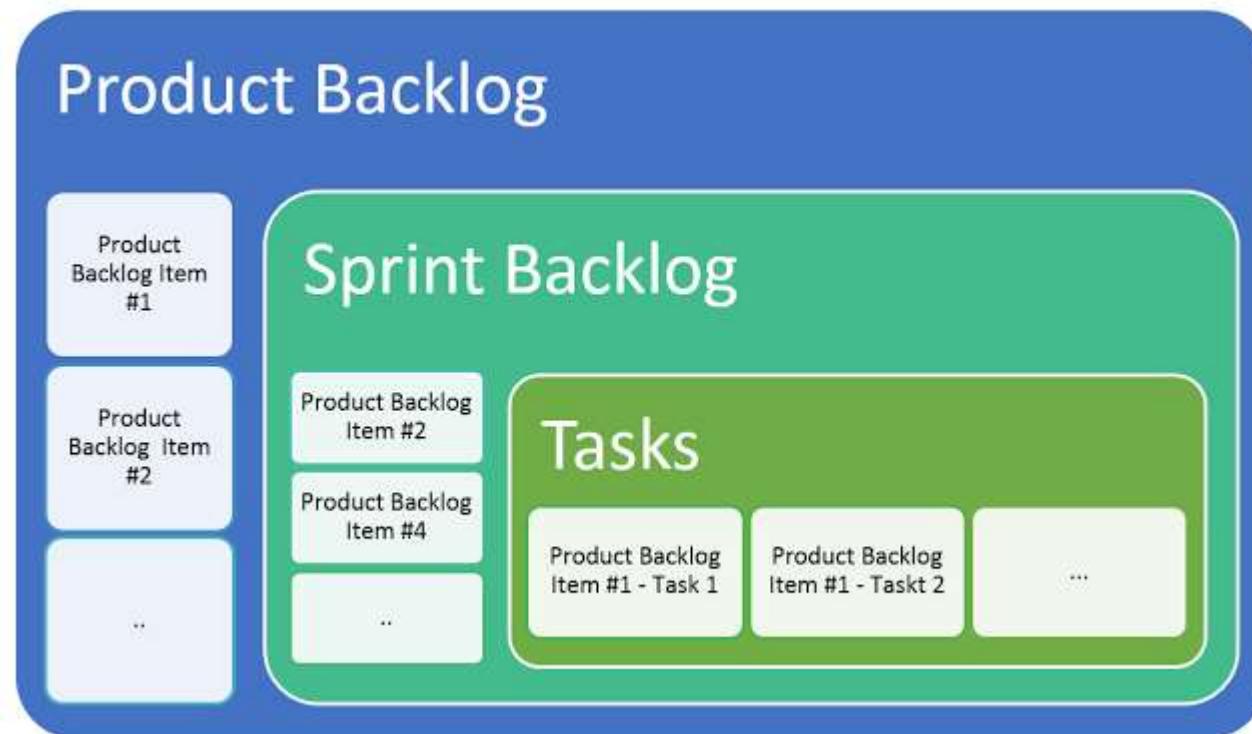
Executing a Sprint

- Sprint Ceremonies.
- Daily Scrums, Scrum of Scrums
- Sprint Review
- Sprint Retrospective

Levels of Planning in Agile



Product-backlog vs. Sprint-backlog vs. Tasks/User Stories



Scrum: Roles – Artifacts - Ceremonies

Roles

Product Owner

- What should we work on and Why? (Vision)

Team Members

- Who will do it and how?

ScrumMaster

- Who will help us do it? (Process Facilitator)

Artifacts

Product Backlog

- What are we doing and in what order?

Sprint Backlog

- What are we doing right now? How will we do it?

Burndown Chart

- How are we doing this sprint? How are we doing this release?

Ceremonies

Sprint Planning

- What are we doing next?

Daily Scrum

- How are we doing today?

Sprint Review

- How did we do?

Sprint Retrospective

- How do we get better?

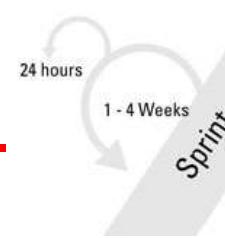
Sprint Ceremonies (Meetings) – A Snapshot

Event	4 Week	3 Week	2 Week	1 Week
Sprint Planning	8 hr	6 hr	4 hr	2 hr
Daily Scrums	15 min daily	15 min daily	15 min daily	15 min daily
Sprint Review	4 hr	3 hr	2 hr	1 hr
Sprint Retrospective	3 hr	2.25 hr	1.5 hr	.75 hr

Sprint-backlog – an Overview

- Within an agile development project, a *sprint backlog* is a list of the tasks and requirements to be completed within the sprint. The sprint backlog includes
- The list of user stories within the sprint in order of priority.
- The relative effort estimate for each user story.
- The tasks necessary to develop each user story.
- The effort, in hours, to complete each task (Each task should take one day or less for the development team to complete)
- A *burndown chart* that shows the status of the work the development team has completed.
- The development team collaborates to create and maintain the sprint backlog, and only the development team can modify the sprint backlog. The sprint backlog should reflect an up-to-the-day snapshot of the sprint's progress

Daily Scrum Meeting



Stage 5: DAILY SCRUM

Description:	To establish and coordinate priorities of the day
Owner:	Development Team
Frequency:	Daily

- Scrum meeting is a Stand-up meeting lasting not more than 15-20 minutes
- Anyone may attend a daily scrum, but only the development team, the scrum master, and the product owner may talk. Stakeholders can discuss questions with the scrum master or product owner afterward, but stakeholders should not approach the development team.
- Focus on immediate priorities. The scrum team should review only completed tasks, tasks to be done, and roadblocks.
- Use the meeting for coordination, not problem-solving. The development team and the scrum master are responsible for removing roadblocks during the day. To keep meetings from drifting into problem-solving sessions, scrum teams can Keep a list on a white board to keep track of issues that need immediate attention, and then address those issues directly after the meeting.
- Hold a meeting, called an after-party, to solve problems when the daily scrum is finished. Some scrum teams schedule time for an after-party every day; others only meet as needed.
- The daily scrum is for peer-to-peer coordination. Save status reports for the sprint backlog.
- The scrum team may request that daily scrum attendees stand up — rather than sit down — during the meeting. Standing up makes people eager to finish the meeting and get on with the day's work

Executing Sprint

- Creating Shippable Functionality
- The objective of the day-to-day work of a sprint is to create shippable functionality for the product in a form that can be delivered to a customer or user.
- To create shippable functionality, the development team and the product owner are involved in following major activities:
 - Elaborating (responding to clarifications, detailing of new features)
 - Developing the Stories/Tasks (the actual work by team)
 - Verifying (automated testing, peer review, and product owner review)
 - Identifying Roadblocks (using organizational cloud, Scrum Master removes roadblocks related to resources, shields team from extra-project activities)
 - Update Sprint Burndown chart and Sprint-backlog at the end of the day

As an existing customer, I want to enter my user information and enter my account securely so I can be confident transacting there.

Acceptance Criteria

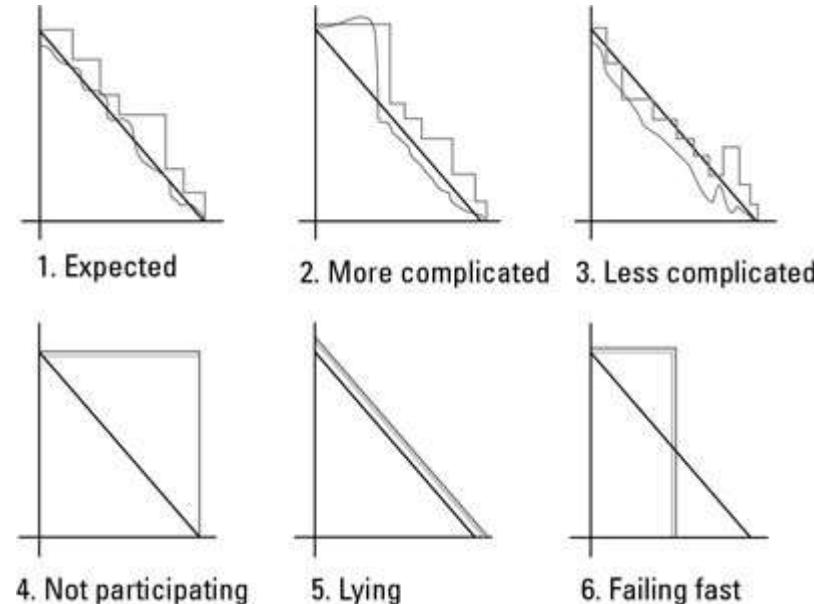
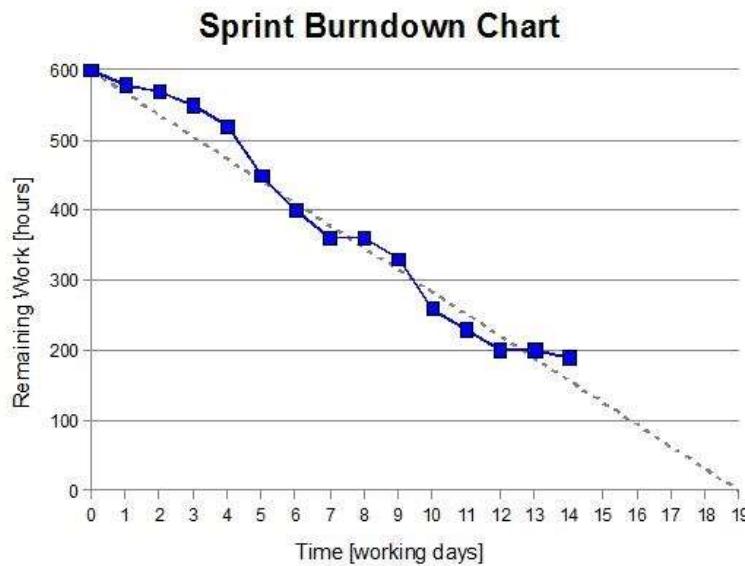
- Accepts all existing user security information as required by IT user management standards
- Allows access to alternate product login
- Authenticates per existing security rules
- Provides username forgiveness
- Provides password context number forgiveness

Design: See wireframes

Development: See workflows and architectural diagrams

Content: All messages must be approved by Compliance and routed through Legal in context with disclaimers

Tracking Progress: Sprint Burndown Chart



Tracking Progress: Sprint Task-board

- Task board — in conjunction with the sprint backlog (which is an electronic version) — gives a quick, easy view of status of the items within the sprint to the development team
- The task board can be made up of sticky notes on a white board.
- Task board to be always visible and accessible to members – they can physically move a user story card through its completion
- The task board encourages thought and action just by existing in the scrum team's work area, where everyone can see the 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... 6 Test the... SC 6	Code the... DC 4 Test the... SC 6	Test the... SC 6 Code the... Test the... SC 6 Test the... SC 6 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 6 Test the... SC 6 Test the... SC 6

Ref: (T2-Chap9)

Progress Tracking through Burndown Charts

- **Expected:** This chart shows a normal sprint pattern. The remaining work hours rise and fall as the development team completes tasks, ferrets out details,
- **More complicated:** In this sprint, the work increased beyond the point in which the development team felt it could accomplish everything. The team identified this issue early, worked with the product owner to remove some user stories, and still achieved the sprint goal. The key to scope changes within a sprint is that they are always initiated by the development team — no one else.
- **Less complicated:** In this sprint, the development team completed some critical user stories faster than anticipated and worked with the product owner to identify additional user stories it could add to the sprint.
- **Not participating:** A straight line in a burndown means that the team didn't update the burndown or made zero progress that day. Either case is a red flag for future problems.
- **Lying (or conforming):** This burndown pattern is common for new agile development teams used to reporting the hours management expects instead of the time the work really takes. A team with this chart likely adjusted its work estimates to the exact number of remaining hours. This pattern often reflects a fear-based environment, where the managers lead by intimidation.
- **Failing fast:** One of the strongest benefits of agile is the immediate proof of progress, or lack thereof. This pattern shows an example of a team that wasn't participating or progressing. Halfway through the sprint, the product owners cut their losses and killed the sprint. Only product owners can end a sprint early.

Sprint-backlog Example

PLATINUM EDGE

My XYZ Mobile Banking - Sprint 1

Sprint Dates: January 3 - January 6

Sprint Goal

As a mobile banking customer,
I want to <log in to my account>
So I can <view my account balances and pending...

Burndown - Based on Est Hours Remaining

Number of working days	Working Hours/day
Leanna (35 hrs/wk)	7
Mark (35 hrs/wk)	7
Caroline (35 hrs/wk)	7
Jim (25 hrs/wk)	7
Steve (25 hrs/wk)	7
Total:	140
Total per day:	35

Burndown: Est Hrs Remaining

Estimated Hrs Remaining

Days in Sprint

Feature Burndown - Based on Est Hours Remaining

Task	Story Points	Status	Responsible	PO	Approved/User Story Complete?	1	2	3	4	5	6
User Story #1 Authenticate and access my accounts	8					0	0	0	0	0	0
Create authentication screen for user Name and Password						0	0	0	0	0	0
Create error screen for user to re-enter credentials						0	0	0	0	0	0
Create logout screen						0	0	0	0	0	0
Using authentication code from online banking application, develop code in iPhone® app application						0	0	0	0	0	0
Create code to validate to verify user name and password						0	0	0	0	0	0
Develop automated test scripts						0	0	0	0	0	0
User Story #2 View balances	5					0	0	0	0	0	0
Develop balances UI page						0	0	0	0	0	0
Develop balances over time						0	0	0	0	0	0
Develop selected account mechanism						0	0	0	0	0	0
Develop automated test scripts						0	0	0	0	0	0

Source: (T2-Chap9)

Sprint-backlog – An Example

ID	Story	Type	Status	Value
121	As an Administrator, I want to link accounts to profiles, so that customers can access new accounts.	Feature	Not started	5
113	As a Customer, I want to view my account balances, so that I know how much money is currently in each account.	Feature	Not started	3
403	As a Customer, I want to transfer money between my active accounts, so that I can adjust each account's balance.	Feature	Not started	1
97	As a Site Visitor, I want to contact the bank, so that I can ask questions and raise issues.	Feature	Not started	2
68	As a Site Visitor, I want to find locations, so that I can use bank services.	Feature	Not started	8

Scaling Agile: Scrum of Scrums

- To Scale Teams beyond 7 members to large Groups
- Divide the Group into Scrum Groups of 5 – 7 each
- A technique to scale Scrum up to large groups (over a dozen people), consisting of dividing the groups into Agile teams of 5-10. Each daily scrum within a sub-team ends by designating one member as "ambassador" to participate in a daily meeting with ambassadors from other teams
- The Scrum of Scrums proceeds otherwise as a normal daily meeting, with ambassadors reporting completions, next steps and impediments on behalf of the teams they represent. Resolution of impediments is expected to focus on the challenges of coordination between the teams; solutions may entail agreeing to interfaces between teams, negotiating responsibility boundaries, etc.
- The Scrum of Scrum will track these items via a backlog of its own, where each item contributes to improving between-team coordination.

Scrum of Scrums (contd.,)

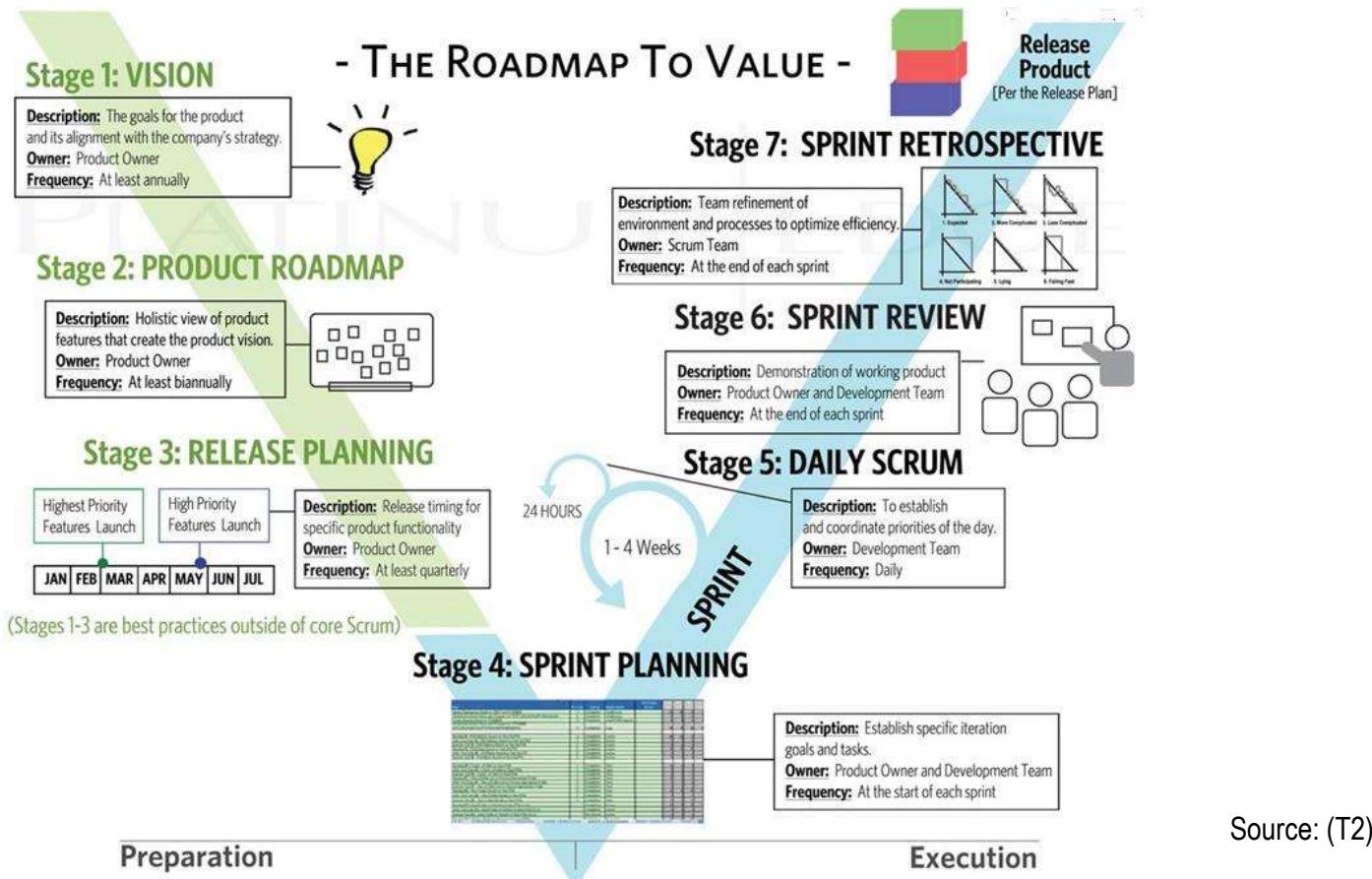
- The scrum of scrums meetings can be scaled up in a recursive manner – more scrum of scrums meetings. Each contains a representative from each of the teams. The work of scrum of scrums meetings can be coordinated through an even higher level meeting (“scrum of scrum of scrums”)
- Scrum of scrums meetings may not be daily
- Agenda - similar to the standard agenda for the daily scrum except the team member represents his own scrum team.

15 minutes	Each participant answers four questions: 1. What has your team done since we last met? 2. What will your team do before we meet again? 3. Is anything slowing your team down on in their way? 4. Are you about to put anything in another team's way? No personal names
As needed	Resolve problems and discuss issues on the team backlog.

Sprint Reviews & Sprint Retrospectives



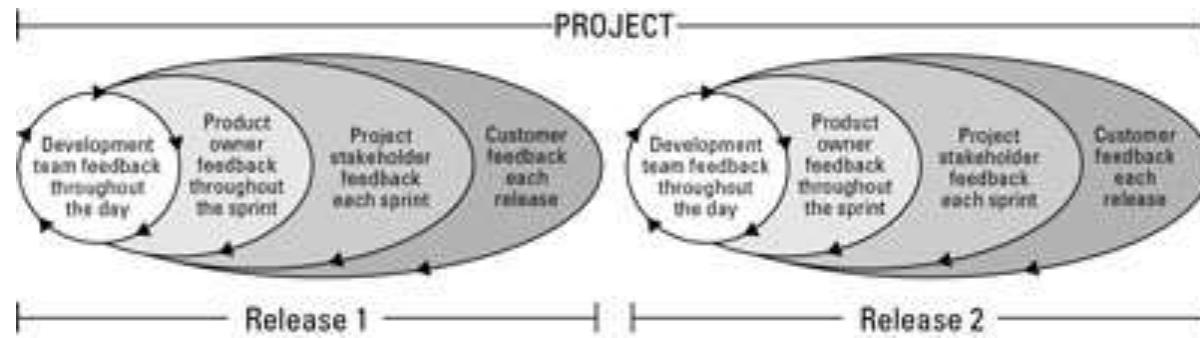
Every Stage in the SCRUM is a Review Stage!



Source: (T2)

Sprint Review

- Apart from demonstration of product, sprint review meeting enables stakeholders to provide continuous feedback
- Each day, team members work together in a collaborative environment that encourages feedback through peer reviews and informal communication.
- In each sprint, as the team completes each requirement, the product owner provides feedback by reviewing the working functionality for acceptance.
- With each release, customers who use the product provide feedback about new working functionality.
- Gathering feedback during the sprint review is an informal process. The product owner or scrum master can take notes on behalf of the development team, as team members are often engaged in presentations/conversations
- New user stories may come out of the sprint review. The new user stories may be new features altogether, or they may be changes to the existing product or code.
- After the review, the product owner has several tasks: Add any new user stories to the product backlog after prioritizing; Add stories that were scheduled for the current sprint but weren't completed back into the product backlog and reorder/reprioritize based on the most recent priorities; Complete updates to the product backlog in time for the next sprint.



Sprint Review Meeting: Guidelines

If my sprint is
this long...

One week

Two weeks

Three weeks

Four weeks

My sprint review
meeting should last
no more than...



45 minutes

1.5 hours

2.25 hours

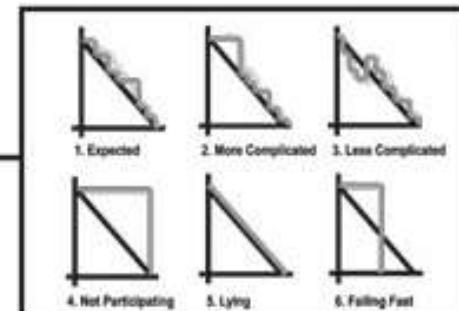
Three hours



Sprint Retrospective (for Process Improvement)

- **SET THE STAGE**
 - Establish the goals (specific areas of improvement) for the retrospective up front.
- **GATHER DATA**
 - Discuss the facts about what went well in the last sprint and what needed improvement. Create an overall picture of the sprint; consider using a white board to write down the input from meeting attendees.
- **GENERATE INSIGHTS**
 - Gather insights/ideas about how to make improvements for the next sprint.
- **DECIDE WHAT TO DO**
 - Determine — as a team — identify ideas and specific actions to make them a reality
- **CLOSE THE RETROSPECTIVE**
 - Reiterate your plan of action for the next sprint.

Description: Team refinement of environment and processes to optimize efficiency.
Owner: Scrum team
Frequency: At the end of each sprint



Summary: Executing a Sprint

- Daily Scrum – where “action” happens (actual development)
- Daily Scrum Meetings (standup meetings ~15 mins) are key to project progress and continuous improvement
- All Team members work on One Story/Task at a time (to ensure *swarming*—collaboration & knowledge-sharing)
- Each Sprint must deliver Working / Demonstrable Product
- Burndown Charts & Task-boards are common tools for Project Status communication
- For larger Agile Teams, Scrum-of-Scrums approach is followed (nesting of Scrum within larger Scrum)
- Sprint Review Meeting is where issues/challenges faced in the last Sprint discussed, working product demonstrated and product-backlog gets updated
- All improvements related to Scrum Process and Team collaboration and communication are addressed in Sprint Retrospective Meeting

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners



Agile Metrics & Tools

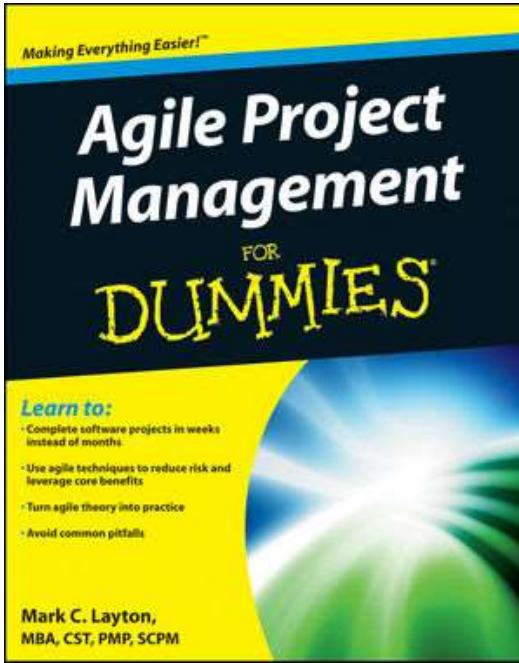
- Prof K G Krishna

Text/Reference Books

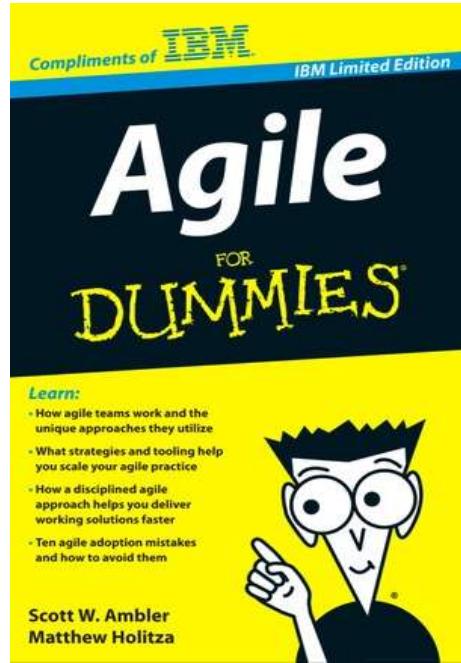
T1



T2



Compliments
of IBM

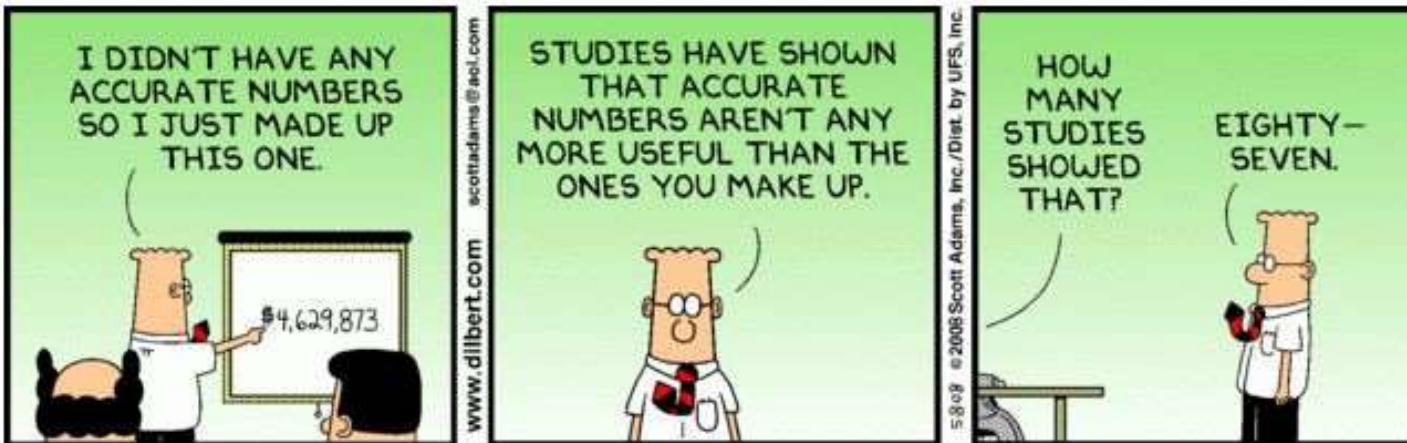


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Agile Metrics & Tools

- Overview of Agile Metrics
- Tools for Metrics



Metrics – Track Performance / Progress

- Metrics is a Measure or combination of Measures for quantitatively assessing, controlling or improving a Process or Product or Team.
- In Agile, We use Metrics for Planning, Inspecting, Adapting and Progress Tracking of Software Development
- Metrics – Quantity /w Unit of Measure; Ratio (e.g. Schedule Overrun, Effort Overrun, Defect Discovery (rate), Project Size (Stories, Components,...), etc.
- Metrics (Term adopted in Software Industry) vs. Measures vs. Indicators (Leading/Lagging) / KPI (strategic, tied to an objective or goal, trend, range,...)
- Metrics are Relative (need Baselining) and Organization/Project/Product-centric
- Typical Metrics in Software Projects:
 - Success metrics
 - Time and Cost metrics
 - Satisfaction metrics



Metrics must be Actionable (→KPIs)

Metrics vs. KPIs – A Comparison	
Metrics provide information that can be digested.	KPIs offer comparative insights that guide future actions.
Metrics are extracted and organized by activity or process.	KPIs are initiated by high-level decision makers.
Metrics can be viewed historically, but do not identify future action.	KPIs incorporate Goals and Objectives.
Metrics are static, and once extracted do not change.	KPIs can be evaluated and reset over time using the SMART methodology.

Source: appdevelopermagazine.com

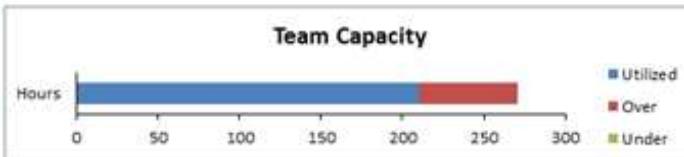
Ten Key Metrics for Agile Project Management

- 1. Sprint goal success rates:** A successful sprint should have a working product feature that fulfills the sprint goals and meets the scrum team's definition of done: developed, tested, integrated, and documented.
- 2. Defects:** Defects are a part of any project, but agile approaches help development teams proactively minimize defects
- 3. Total project duration:** Agile projects get done quicker than traditional projects.
- 4. Time to market:** *Time to market* is the amount of time an agile project takes to provide value,
- 5. Total project cost:** Cost on agile projects is directly related to duration
- 6. Return on investment:** *Return on investment (ROI)* is income generated by the product, less project costs
- 7. New requests within ROI budgets:** Agile projects' ability to quickly generate high ROI provides organizations with a unique way to fund additional product development.
- 8. Capital redeployment:** On an agile project, when the cost of future development is higher than the value of that future development, it's time for the project to end.
- 9. Satisfaction surveys:** A scrum team's highest priority is to satisfy the customer.
- 10. Team member turnover:** Agile projects tend to have higher morale. One way of quantifying morale is by measuring turnover

Sprint Estimation Metrics

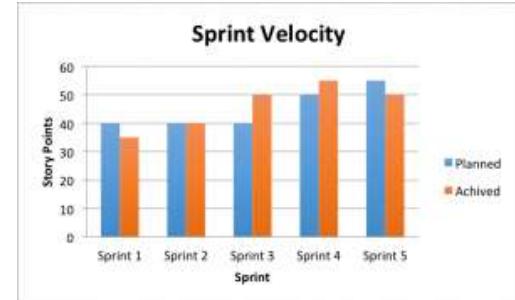
Team Capacity

Totals	Hours
Remaining Work	271
Remaining Capacity	210
Utilized	210
Over	61
Under	0

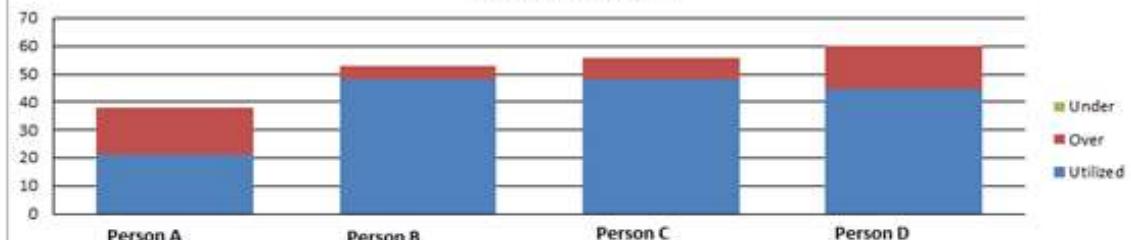


Individual Capacity

Team Member	Hours/Day	Days	Capacity	Assigned	Utilized	Over	Under
Person A	6	8	48	51	48	3	0
Person B	3	7	21	38	21	17	0
Person C	6	8	48	53	48	5	0
Person D	6	8	48	56	48	8	0

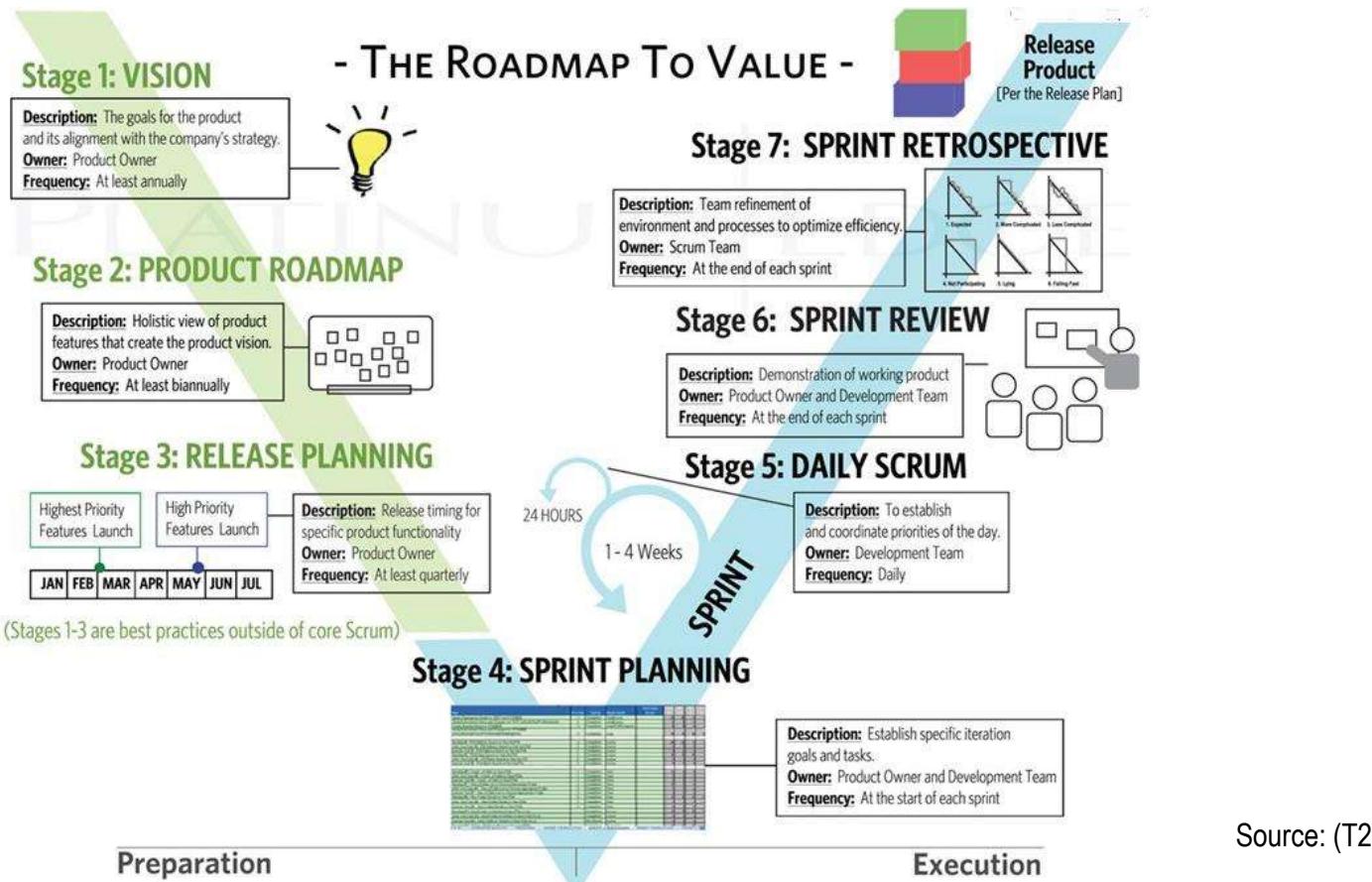


Individual Capacity



Tools for Agile Management →→

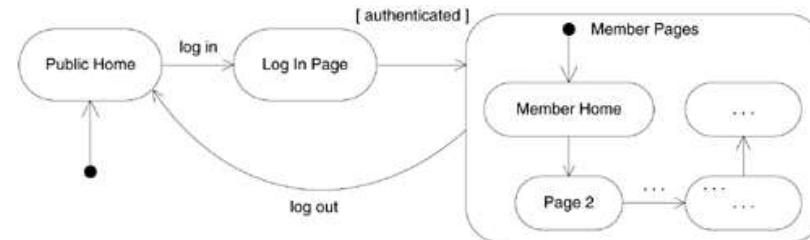
Metrics/Tools at every Stage of Agile Life-cycle



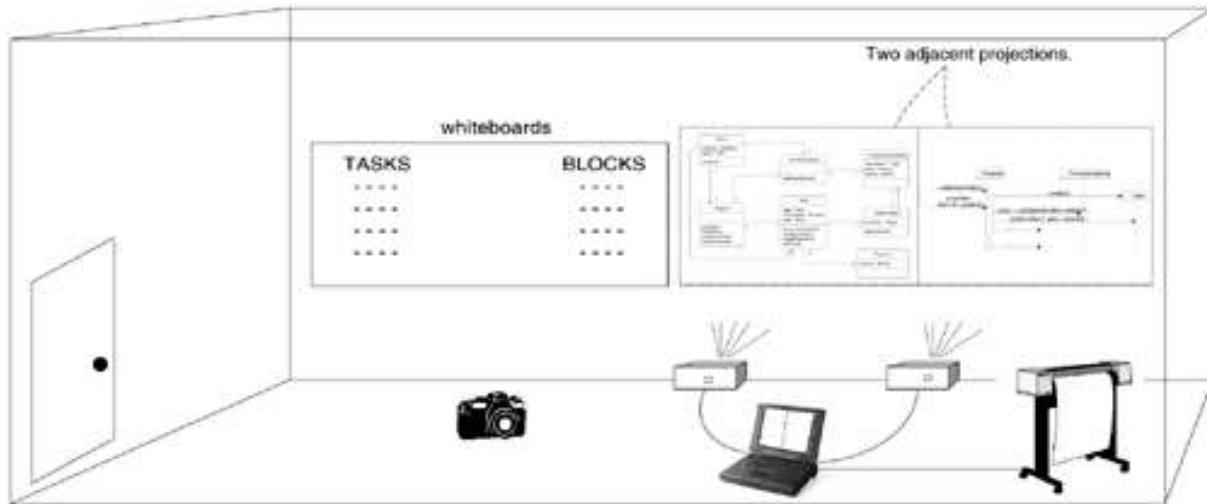
Source: (T2)

Tools for Agile Project/Knowledge Management

- Project Wiki Webs (as “the simplest online Web database that could possibly work” – Ward Cunningham, Founder of XP)
 - Wikis are popular tool on Agile projects to capture project information; when used as Knowledge sharing tool, Wikis allow people to edit Web pages using only their browser, create new pages and hyperlinks between pages
 - CASE Tools for Forward-engineering (generation of code from UML diagrams) and Reverse-engineering (generation of drawings from code)
 - Help in generation of minimal (and automatic) documentation from the code
 - UML diagrams printed on large A0-sheets help enhance communication in common rooms
 - Task-boards: Use of Whiteboards, Cling-sheets, Flip-charts,...for Visual Communication
 - Excel Graphing, Visio Diagramming, Mindmaps, Index/Story Cards,...
 - Fit(fit.c2.com), Fitnesse (fitnesse.org) – an Open-source framework and tool to support acceptance testing developed by Ward Cunningham and Bob Martin.



Sample XP Room for Collaboration, Communication and Brainstorming



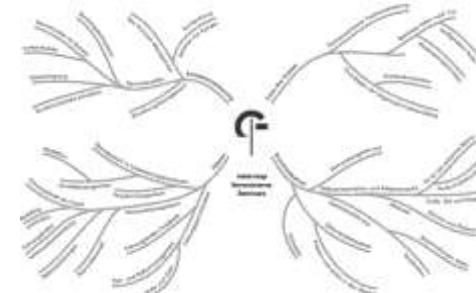
Software:

- Continuous integration tools
- A project Wiki
- Consider a UML CASE tool to reverse engineer diagrams from code.

Hardware:

- Consider two projectors attached to dual video cards.
- For whiteboard drawings, use a digital camera.
- To print noteworthy diagrams for the entire team, try a plotter for large-scale drawings to hang on walls.

Source: (T1)



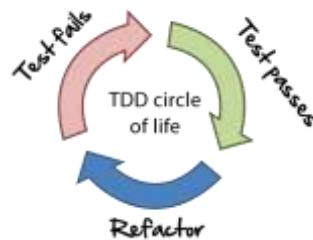
Summary: Agile Metrics & Tools

- Metrics must be **S.M.A.R.T** (Specific/Simple, Measurable, Achievable, Relevant, Time-bound/Transparent)
- Metrics are **Specific/Tailored** to Organizational Processes
- Metrics to be **Baselined** before commencing Tracking for realistic assessment of Progress
- Metrics are categorized under: **Success Metrics**, **Time/Cost Metrics**, and **Satisfaction Metrics**
- Sprint **Velocity** and **Capacity** are the two Metrics for Planning/Estimating in Scrum
- Task Boards and Open Rooms facilitate **real-time** communication of Metrics to all members of Scrum Team
- Project Wikis and CASE tools are some of the Tools for **Knowledge Sharing** in Scrum Team

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

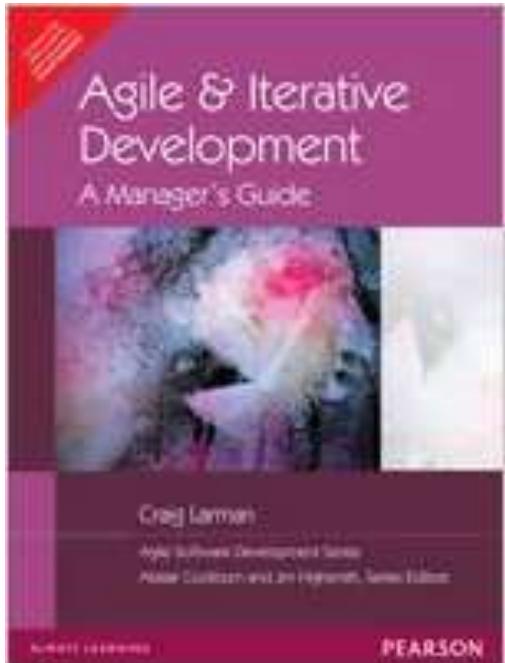


Quality Management in Agile

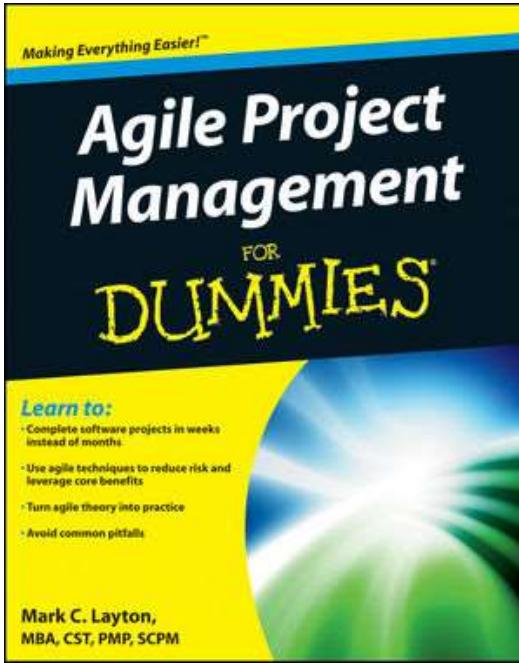
- Prof K G Krishna

Text/Reference Books

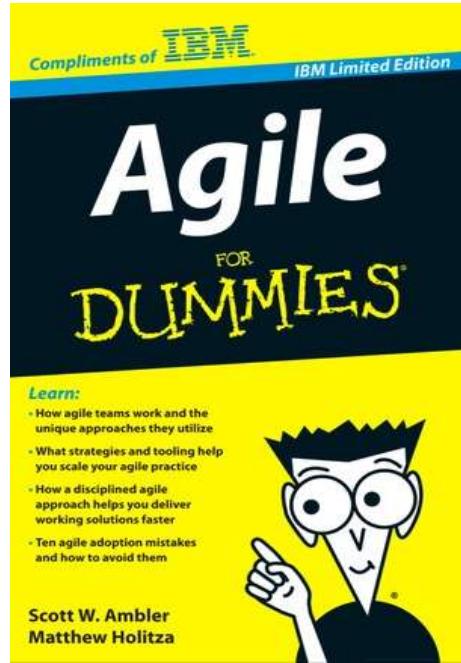
T1



T2



Compliments
of IBM

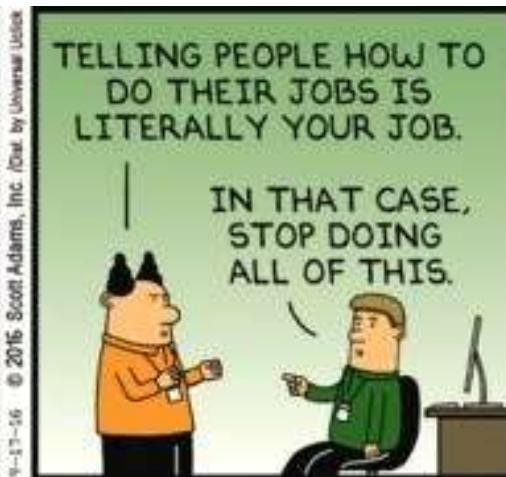


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Quality Management in Agile

- Managing Quality in Agile
- Integrated Testing in Agile
- Managing Risks in Agile

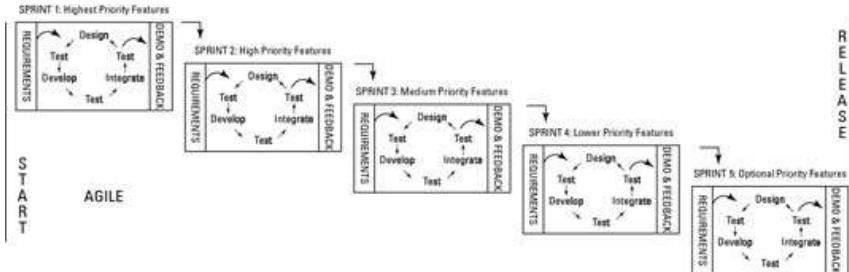


Ensuring Quality: Traditional vs. Agile

- Testing is the Last Phase of the Project vs. Daily Activity as part of Sprint
- Reliance of Manual Testing (unit) vs. Automated Test Tools (necessary)
- Reactive (fixing discovered bugs/issues) vs. Focus on Proactive practices (Pair Programming, Coding Standards, and Face-to-Face Communication)
- Risk is more when Problems surface at the end vs. Risk is avoided by addressing it in early Sprints
- Bugs are hard to discover at the end of Project vs. Finding and Fixing Bugs in small iterations of little code is easy
- Testing gets compromised as Project reaches deadlines vs. Testing is continuous activity and is integrated into daily development (Continuous Testing)

Quality is Integrated into Agile Methods

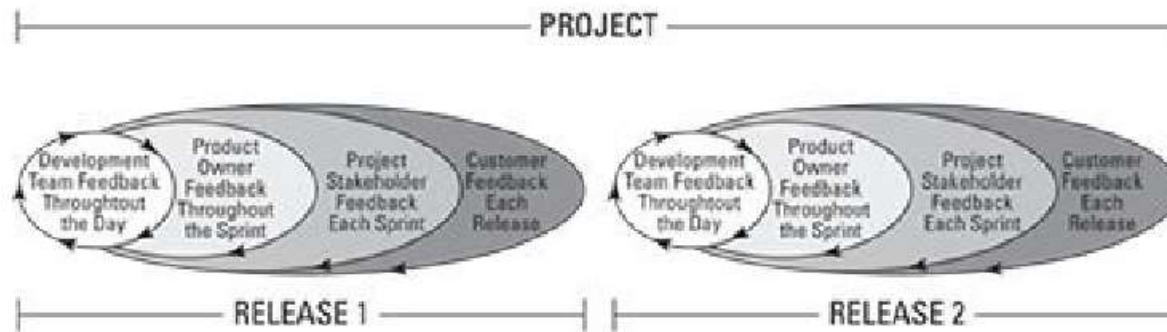
- 1. Test Continuously:** Testing begins in the first sprint, and the scrum team evaluates quality throughout each sprint, finding and fixing any problems immediately.
- 2. Proactive Prevention:** Building Acceptance criteria into User Stories
- 3. Inspecting Regularly and Adapting as Needed:** Product (during Sprint Review) and Process (during Sprint Retrospective) are reviewed and adapted as per the need
- 4. Use of Automated Test-tools** help embed Testing in Daily activities and in every Sprint



To Do	In Progress	Verify	Done
Code the... 9	Test the... 8	Code the... DC 4	Test the... 6
Code the... 2	Code the... 8	Test the... SC 8	Test the... 5
Test the... 8	Test the... 4		Test the... SC 6

Multiple *Feedback Loops* Feeding Quality

- Frequent (~Daily) Feedback from Development Team, Product Owner, Stakeholders and Customer (as part of Daily Scrum, Sprint Review, Release Planning)
- Involvement of Cross-functional Teams, Domain/Technical Experts
- Continuous Inspecting and Adapting ensures Quality
- Easier to Find Bugs in Short Code (of one Iteration) rather than Large Code (end of Project)

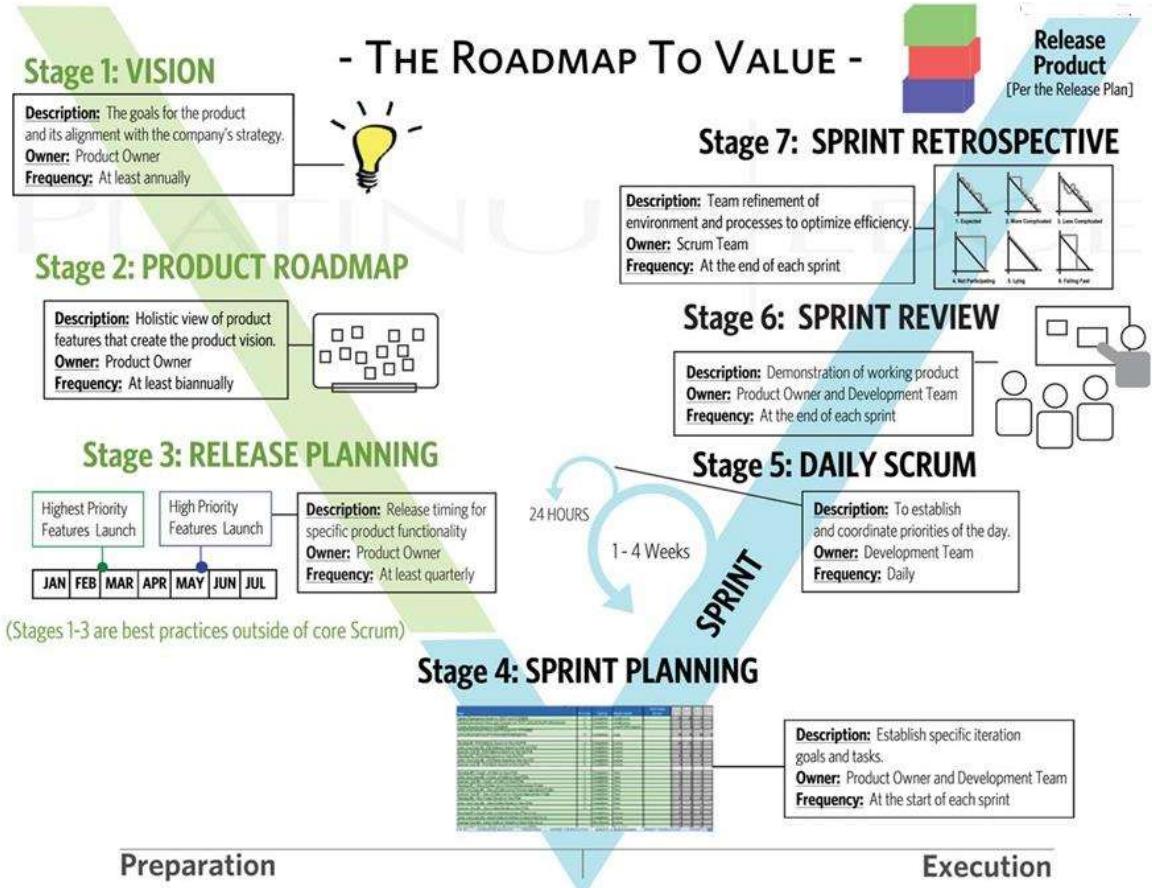


Agile Development Methods emphasize Quality

- **Test-Driven Development (TDD):** Evolved from XP Practices, TDD focuses on creating Tests for the Requirement before coding, then ‘failing’ the Test, then ‘develops’ code to fulfil the test and then refactors the code by taking as much code as possible while the test passes
- **Pair-programming:** XP Practice in which Developers work in pairs—both developers work at the same desk and take turns of development and testing for the same Requirement; ensures quality by constant over-the-shoulder review by the peer by providing instant error checks-and-balances
- **Peer-reviews:** Reviewing each others’ code they are collaborative in nature by allowing peer-level experts in other groups to help ensure objective review
- **Collective Code Ownership:** Key feature of Agile where any Team member can create, fix or change any part of the code on the project; speeds up development, fosters innovation and help find bugs quickly
- **Continuous Integration:** Daily builds helps check how the current story built works with the rest of the product; allows resolve conflicts early on; daily code builds are necessary for running automated test-suites later in the night

Managing Risks in Agile →→

Every Meeting in the SCRUM Execution is a Risk Mitigation Meeting!



Source: (T2)

Managing Risk in Agile Projects

- Risk Management is integral to Agile – it doesn't have to involve formal Risk Documentation and Meetings. Risk management is embedded in these below Agile Principles (part of 12 Agile Principles Manifesto):
 - ❖ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
 - ❖ Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
 - ❖ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
 - ❖ Business people and developers must work together daily throughout the project.
 - ❖ Working software is the primary measure of progress.
- ➔ “Failing-fast” early subsumes Risk in Agile Projects by prioritizing High-value and High-risk Requirements during early Sprints

Managing Risk: Traditional vs. Agile

- Large Projects are challenged with Overruns and Scope-creep and nothing-to-show till the end vs. Catastrophic failures are eliminated with 'something-to-show' at every stage
- Conducting Testing (ST/UAT) at the end means finding serious problems pose grave risk to the project vs. Continuous testing which surfaces problems early and if required project can be abandoned early on without significant upfront investment
- Requirements change in later phases unacceptable vs. Requirements change can be welcome at any stage by dynamically adjusting priorities
- Inaccurate estimates of Cost/time made at the start when we know least information about the Project vs. No upfront estimates – only running estimates/restimates using Scrum Team's actual performance or Velocity
- Multiple Stakeholders having diverse view of the Product may end up arriving at conflicting Requirements vs. Single Product Owner responsible for end-to-end Product development from Visioning to Delivery
- Project Stakeholders may be unresponsive for issue resolution once the Project starts vs. Scrum Master with organizational clout dedicated to the Team to remove roadblocks
- Income generation/revenues from the Project only upon final delivery vs. Income generation may commences from the first release (or after first Sprint); a Self-funded project that generates additional revenue with every release has a good chance of continuing during a crisis

Risk Management Artifacts/Meetings in Agile

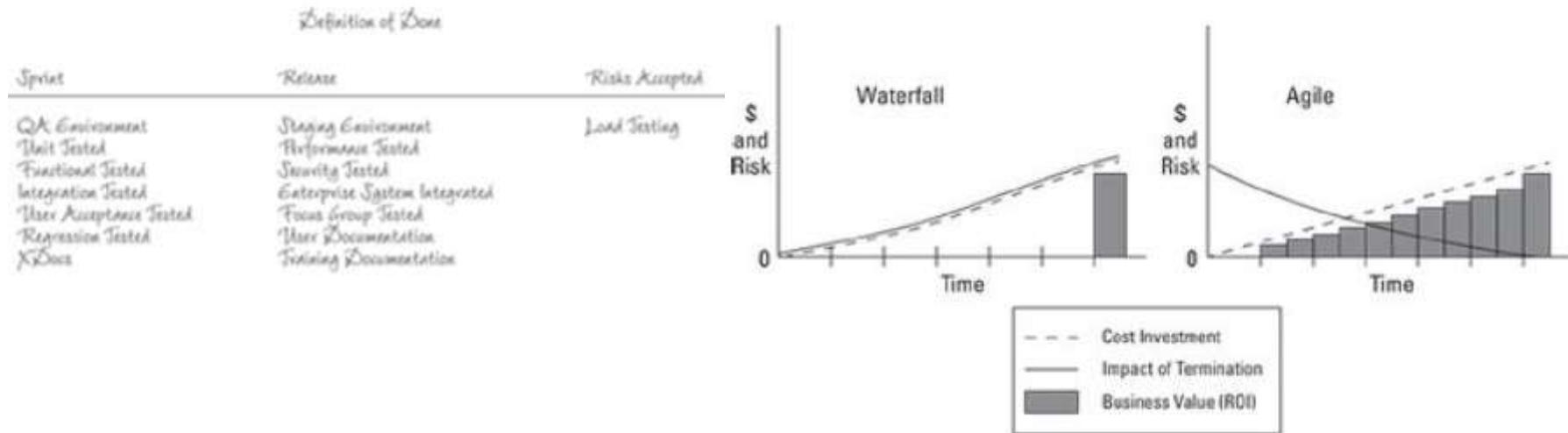
- Product Vision
 - Product Roadmap
 - Product-backlog
 - Release Planning
 - Sprint Planning
 - Sprint-backlog
 - Daily Scrum
 - Task-board
 - Sprint Review
 - Sprint Retrospective



Title	Transfer money between accounts		
As	Carol,		
I want to	review fund levels in my accounts and transfer funds between accounts.		
so that	I can complete the transfer and see the new balances in the relevant accounts.		
Value	Jennifer	Author	Estimate
When I do this:	<p>1. click on transfer funds</p> <p>2. choose from account drop-down</p> <p>3. choose to account</p> <p>4. type in an amount and click transfer</p>		
This happens:	<p>Transfer options appear on screen</p> <p>My available accounts with balance appear</p> <p>My available accounts with balance appear</p> <p>Money is transferred between by from and to accounts</p>		

Risk Mitigation is Inherent in Agile

- Short Development Cycles (Sprints) and Daily Scrum Meetings trap all potential risks early on in the Project and take decisions by adapting Product-backlog and Scrum Process if necessary
- The definition of “Done: Developed, Tested, Integrated and Documented” reduces risk factor by creating a Product that meets this definition in every Sprint



Summary: Quality and Risk Management in Agile

- Quality and Risk Management are embedded in the Agile Process itself
- Testing is integrated into Development ('Done' definition includes all Testing activities for every intermediate working product of Scrum Team)
- Quality and Risks are reviewed and acted upon in every Meeting (Daily Scrum, Sprint Review and Sprint Retrospective)
- Product Owner and Scrum Master are empowered to deal with Risks of Quality or Schedule/Cost overruns as and when they surface
- Ensures Transparency and Open Discussion of issues of Quality/Risk through highly visible Task-boards and Face-to-face Communication
- Pair-programming ensures continuous Peer-review by working in pairs of Developer-Tester/Reviewer with one overlooking the other
- The inherent Structure of Scrum—Product Owner, Scrum Master and Collective Ownership of the Product along with involvement of Customer and Key Stakeholders—safeguards against many of the impediments to Quality or Risks in the Project

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

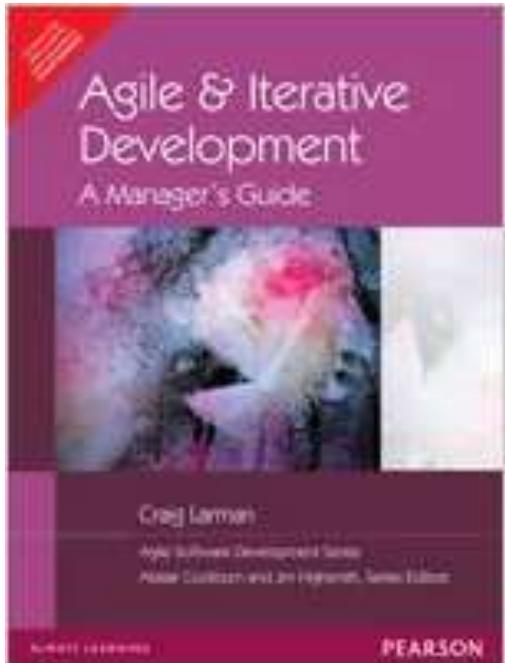


Agile Pitfalls

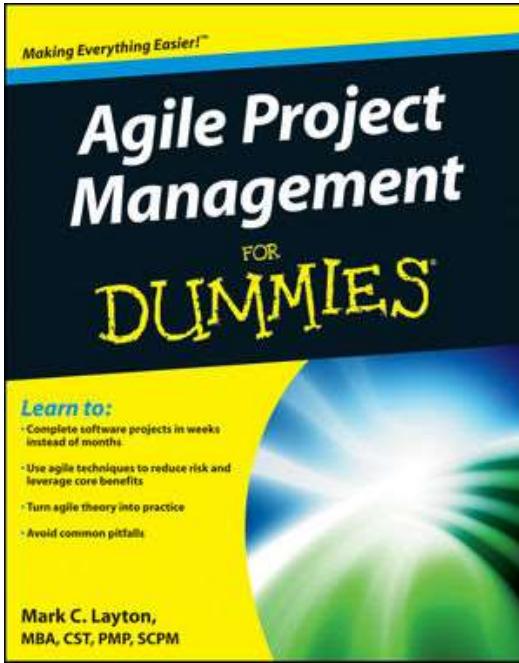
- Prof K G Krishna

Text/Reference Books

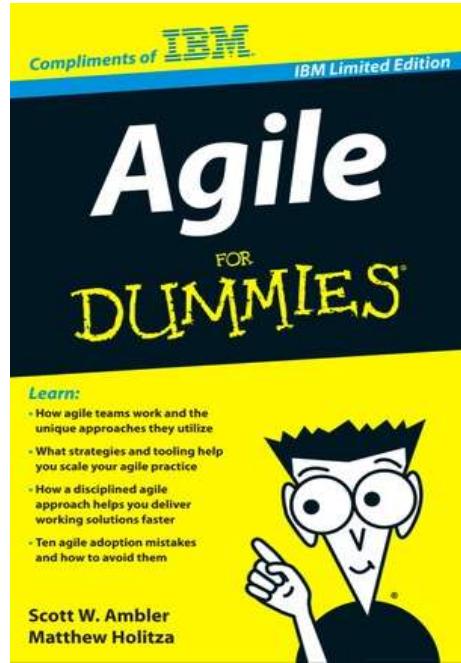
T1



T2



Compliments
of IBM

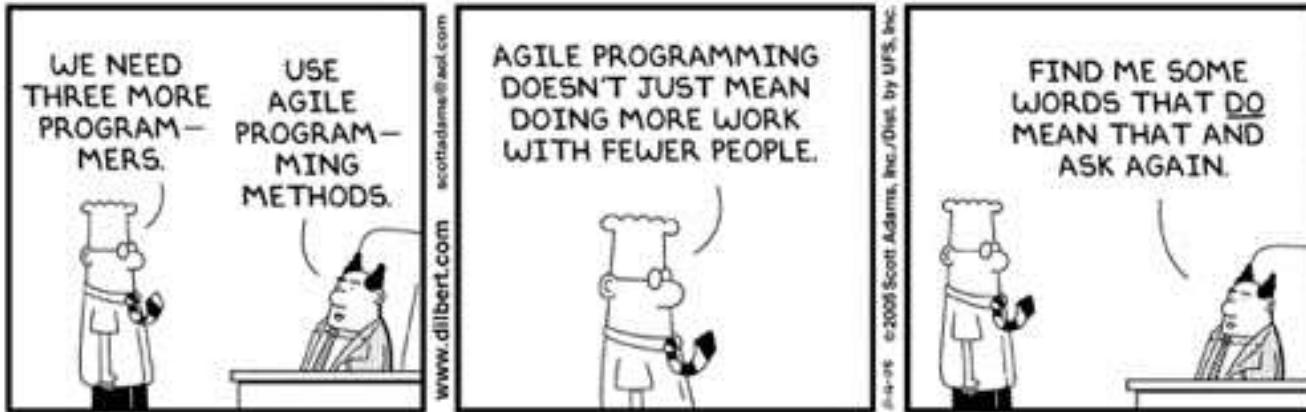


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Agile Myths & Pitfalls

- Common Mistakes/Myths in Agile
- Predictive Planning vs. Adaptive Planning
- CMMI vs. Agile
- Distributed Agile



Common mistakes/misunderstandings

- ✗ Agile Process is Adoption/Adaptation of ‘Waterfall-inspired’ *Predictable* Iterative and Incremental Process

“...We have adopted iterative method for two months and finished use case analysis and plan and schedule of what we'll be doing in each iteration. After review and approval of final requirements set and iteration schedule, we'll start programming...”
- ✗ Scrum Masters/Managers Direct the Team
 - Scrum Master is a only a Facilitator of the Process, removes roadblocks
- ✗ No Daily-update of Sprint-backlog by Members
- ✗ New Work added to Iteration or Individual
 - Stability is required at least once an Iteration starts
- ✗ Owner is Not Involved or take Decisions--too Many ‘cooks’ involved
 - Scrum is Customer-driven and Product Owner represents Customer’s Requirements and hence defines/prioritizes Product-backlog
- ✗ Design/Diagramming is followed, Documentation is bad
 - Scrum is pragmatic in its choice of tools—simple and value-adding (“don’t use sword when nail is enough”)
- ✗ Too many Meetings, too often
 - Short (<15mins) Daily Meeting to sync-up on Issues (not problem-solving which happens in constant communication/collaboration among all team members)
- ✗ Iteration is Not complete and without Integration testing
 - Each Iteration is Fully Done—coded, unit-tested, integration-tested and documented; can be a release to customers (though after couple of iterations)

Common mistakes/misunderstandings (contd.,)

- ✗ Applying a subset of Practices that do not complement/compensate each other
 - e.g., Collective Code Ownership is not feasible with Testing, Continuous Integration and Coding Standards; minimal requirements documentation is not feasible without onsite customer; frequent refactoring doesn't work without testing, etc.
- ✗ Not Writing Unit Tests First
 - Writing the tests first influences how one conceives, clarifies, and simplifies the design; test-first has an interesting psychological quality of satisfaction: I write the test, and then I make it succeed; there is a feeling of accomplishment that sustains the practice
- ✗ No Customer-owned Tests (UAT)
- ✗ Minimal Refactoring
 - The XP avoidance of design thought before programming is meant to be compensated by a relatively large refactoring effort, such as 25% of total effort applied to refactoring
- ✗ Many Fine-grained Tasks (Story cards)
 - Most task cards should be in the one or two-day range of effort. Most in the “few hours” range creates unnecessary information management.
- ✗ Pairing with One Partner Too Long
 - XP pairing changes frequently, often in two days or less. Variation also helps spread the learning.
- ✗ Customer or manager is tracker
 - Programmers will feel awkward reporting slow progress.

Common mistakes/misunderstandings (contd.,)

✗ Not integrating the QA Team

- Many organizations have a separate Quality Assurance team, used to having a completed system “thrown over the wall” to them.

✗ Post development design is wrong

- XP isn't anti-documentation, but prefers programming if that's sufficient to succeed. XP can support the creation of design documentation to reflect the existing code base, once the program is complete

✗ In Pair programming, not willing to learn or explain

- For successful pair programming, an attitude of openness to learning and of explaining yourself is required.

✗ Iterations are NOT Time-boxed

- It is a misunderstanding to let the iteration length expand when it appears the goals can't be met within the original time frame-- rather, the preferred strategy is usually to remove or simplify goals for the iteration and analyze why the estimates were off the mark

✗ Each iteration ends in a production release

- The baselined software produced at the end of an iteration is an internal release rather than shippable code. It represents a subset of the final production release, which may only be ready after a dozen or more short iterations

✗ Predictive Planning

- It is a misunderstanding to create, at the start of the project, a believable plan laying out exactly how many iterations there will be for a long project, their lengths, and what will occur in each. This is contrasted with the agile approach: adaptive planning.

Agile Myths in a Nutshell...

- ❖ Should customize the choice of practices without having first applied them all.
- ❖ “Doing XP” means to avoid the waterfall model and develop iteratively, or just to avoid documentation, or just to write some unit tests.
- ❖ Customers are not involved in the Planning Game, creating acceptance tests, or reviewing the iteration results.
- ❖ Create a plan laying out how many iterations there will be for the project, their lengths, and what will occur in each (*Predictive Planning*)

Agile Pitfalls/Challenges

- **Culture Change:** Top-down, big-bang culture change, adaptation, and institutionalization; traditional management methods clash with Agile concepts and practices
- **Fixed-price Contracting?** Not amenable to traditional fixed-price contracts for large agile projects which are complex
- **Resistance to Drop Established QA Practices:** Not willing to integrate or totally dissolve structures related to QA and Testing functions
- **Inadequate Management/Development Tools**
- **Fuzzy Interface** with Architecture, Quality, Security and Usability functions
- **Lack of Planning:** Non-standardization of Release and Iteration Planning
- **Lack of Trust/Teamwork:** Tendency to Micromanage, Organizational Politics, and Conflicts between Developers and Project Managers
- **Inadequate QA:** Inconsistent use of Agile testing, Usability, Security and other QA Practices; Inadequate testing to meet iteration time-box constraints
- **Lack of Expertise:** Inadequate Skills and Experience (SME and Coaches)
- **Multisite/Multiteams:** Working on complex R&D/Large Projects across Locations

Planning (Waterfall vs. Iterative vs. Agile) →→

Predictive Planning vs. Adaptive Planning (Agile)

- Predictive Planning (flavor of ‘waterfall’)
 - Planning for few Large Milestones
 - Determining in advance Number of Iterations
 - Plan in Detail All The Iterations
- Adaptive Planning (flavor of ‘agile’)
 - Though Milestones can be fixed, but the Path towards Milestones is flexible
 - Milestones can themselves change later in the best interests of Project
 - Plan in Detail for the Just Next Iteration

“Plans change very easily. Worldly affairs do not always go according to a plan and orders have to change rapidly in response to a change in circumstances. If one sticks to the idea that once set, a plan should not be changed, a business cannot exist for long.”

Taiichi Ono
Father of Lean (Toyota Production System)

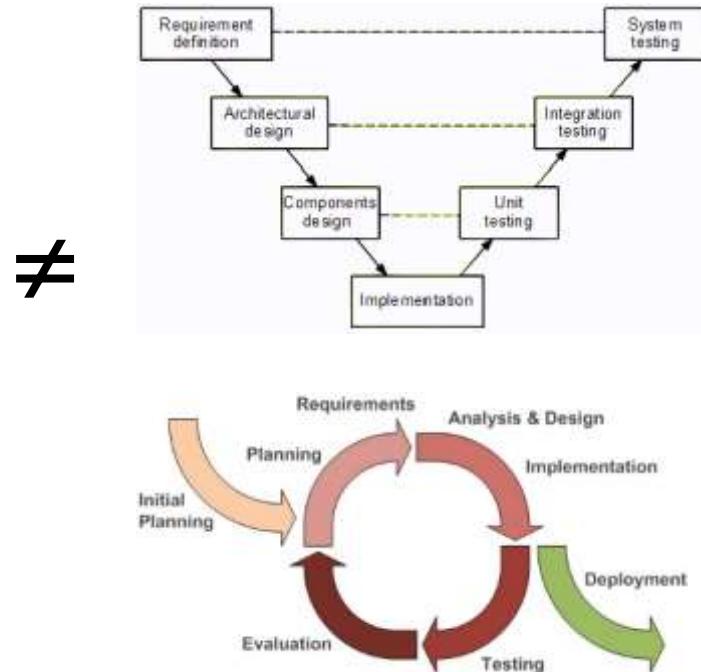
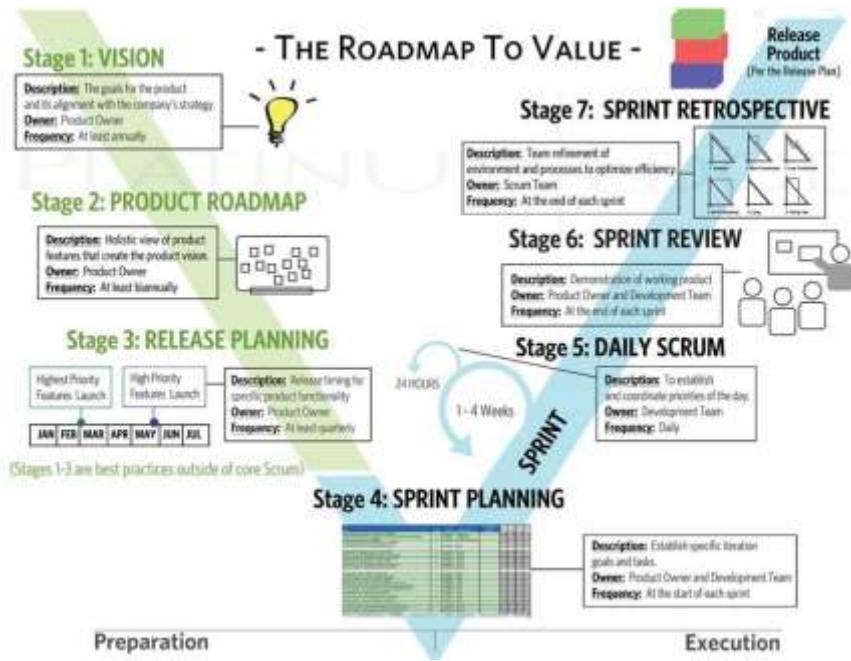
Planning: Waterfall vs. Iterative (Predictive Planning) vs. Agile (Adaptive Planning)

Practice	Waterfall	Iterative (hybrid)	Agile
Effort Estimation	PM provides estimates and get approval from PO for entire project.	Project Manager (PM) provides the estimation for each iteration.	Scrum Master facilitates and Team does the estimation. Story points can be reviewed and refined during sprint planning meeting.
Scheduling	Scheduled by phase wise milestones – Analysis, Design, Development and Testing	Scheduled based on Iteration wise delivery commitments – Iternation#1, #2, #3, #4, etc.,	Scheduled based on velocity and Release backlog. Time boxed in short cycles of duration say 1wk, 2wks, or 3wks – Sprint#1,#2,#3,#4, etc.,
Plan Review	Team need to stick to baseline project plan.	Team need to stick to baseline iteration plan	Team can review during mid sprint planning

The 'Big' Cultural Differences between CMMI and Agile

	CMMI	Agile
Application	Process improvements	Software development
Focus	Existing processes	New processes and products
Main goal	Organizational improvements	Shippable product
Management processes (risk, quality, etc)	Standardized	Not included

Agile is NOT an Iterative Adaptation of Waterfall (V-Process) Model!



Source: (T2)

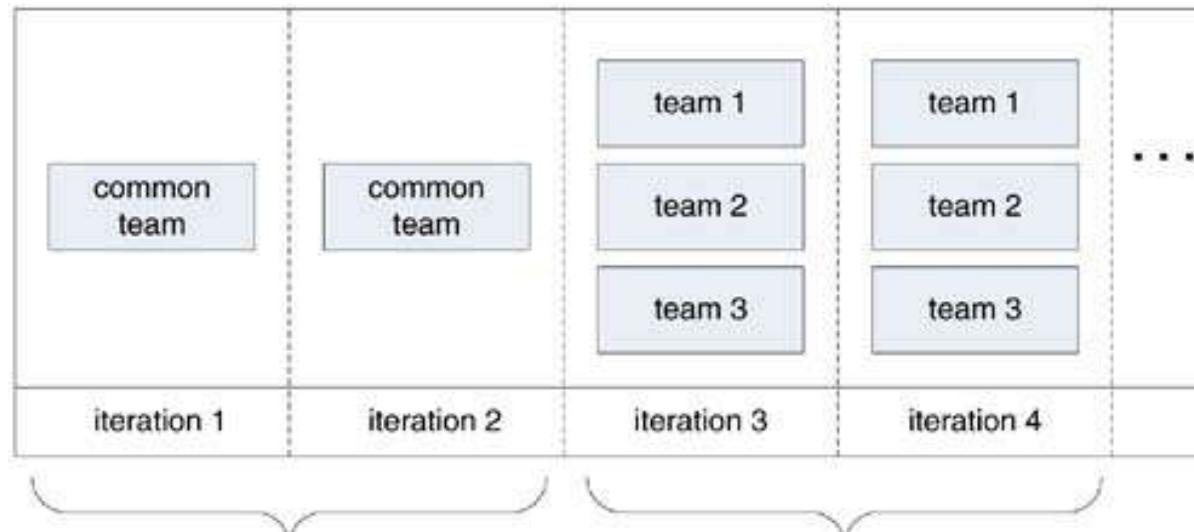
Distributed Agile (Multisite/Multiteam) →→

Multisite / Multiteam Planning

- For projects that will be composed of multiple teams, spread across different locations, consider **doing the early iterations at one location**, in one common project room, with a small group ideally composed of one or two skilled representatives from each of the subteams.
- During these iterations, there is an **emphasis on requirements analysis** and development to discover and build the core architecture of the system—the foundation.
- Major components, and their collaborations and **interfaces are ideally clarified through early programming and testing** rather than just speculative design; the early project benefits from the close communication, collaboration, common vision, and technical strength of the initial group.
- Once the core is built, the representatives return to their respective locations, form larger teams, and the **remaining work is done in parallel with multiple subteams**.
- Each representative has developed a clearer picture of the vision and architecture, and can better convey and maintain that for the remainder of the project. Further, each acts as a liaison to the other teams. Also, after having spent some close time with the other subteam leaders, there is **improved communication between the subteams**.

Multiteam Development

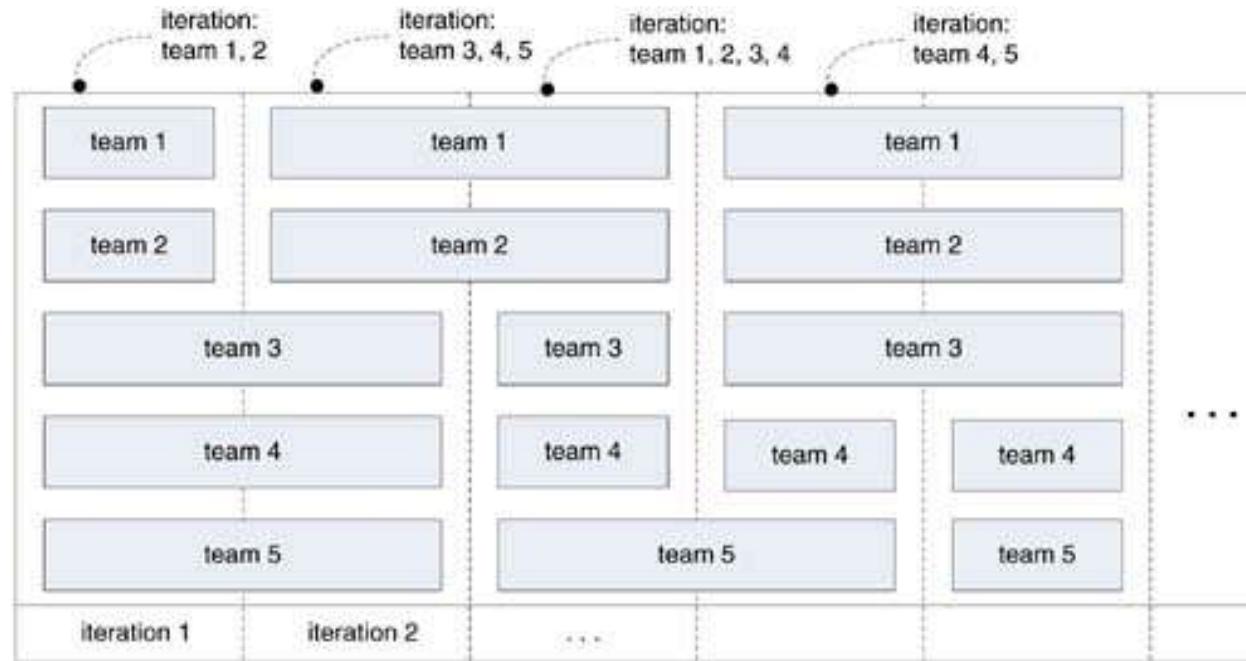
- Have all components, across all subteams or sites, integrated and tested together to conclude the release of a common iteration (to be relaxed in certain projects involving R&D, e.g. development of 5G protocol stack)



small team in common project room, representatives break up, form sub-teams, composed of representatives from the sub-teams and lead the remaining work

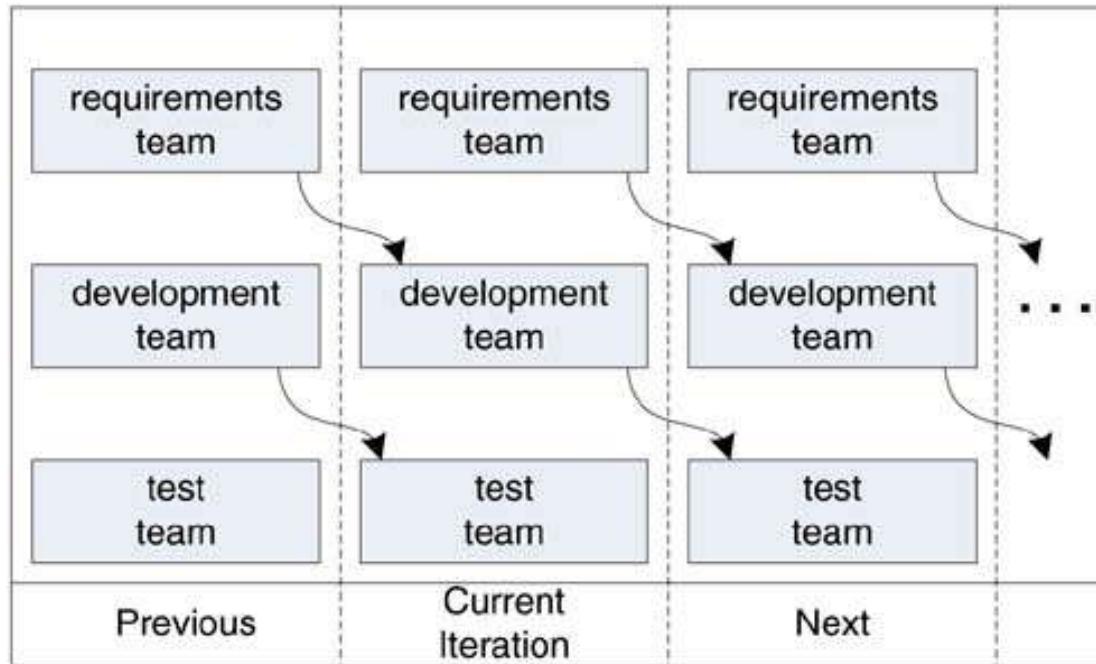
Subteam Iterations (for R&D Projects)

- Ideal for Multisite research-oriented project, the preferred guideline is that all teams work together towards a common goal and integrate on a common iteration end date



Pipelining Iterations (for large projects)

- More appropriate for larger projects, those with offsite requirements donors, or those with long, complex testing that must be performed by a separate expert team (e.g. complex testing which includes labour-intensive manual testing, security testing and memory-leak stress testing, etc. where the system needs to run for many hours or days to discover defects)



Summary: Agile Myths – Debunked / Major Challenges

- Agile is NOT an adoption of Spiral and other IID methods of ‘Waterfall era’
- There IS just enough Planning for meeting major milestones, but the path towards milestones can involve multiple (and unpredictable number of) iterations
- Agile does NOT avoid Design, but evolves Design with each Iteration
- Agile (Scrum, in particular) DOES support minimum (and agile) Process vs. (rigid and monolithic) process in Waterfall model
- Not EVERY Iteration should lead to Production Release
- Handling Fixed-price Contracts is a Challenge in Agile—but NOT impractical
- Culture change is the BIG Transformation effort required before going for Agile
- Multisite/Multiteam Projects too can be handled using Agile using appropriately planned and coordinated Iterations across locations

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners

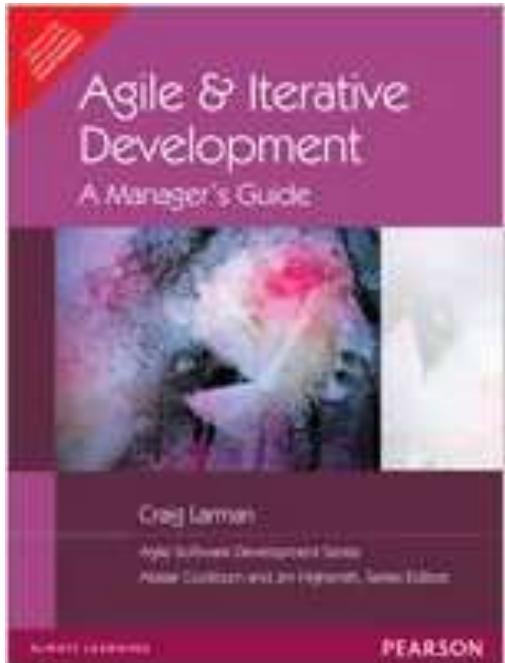


Ensuring Agile Success

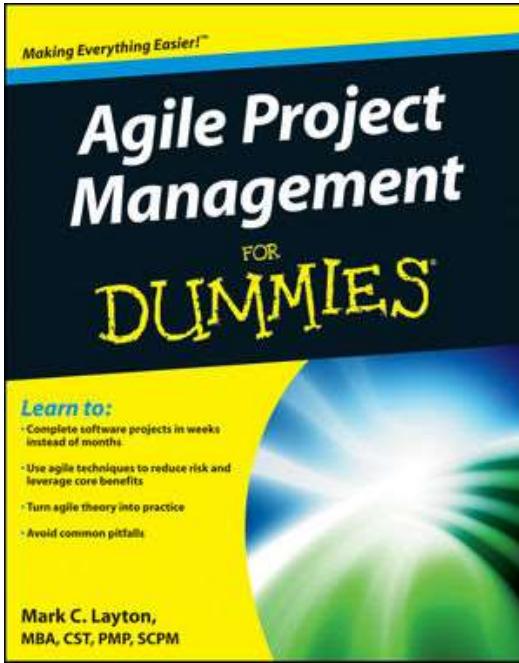
- Prof K G Krishna

Text/Reference Books

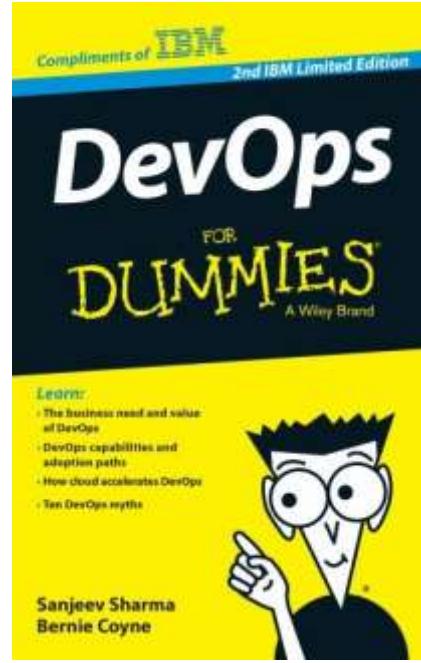
T1



T2



Compliments
of IBM

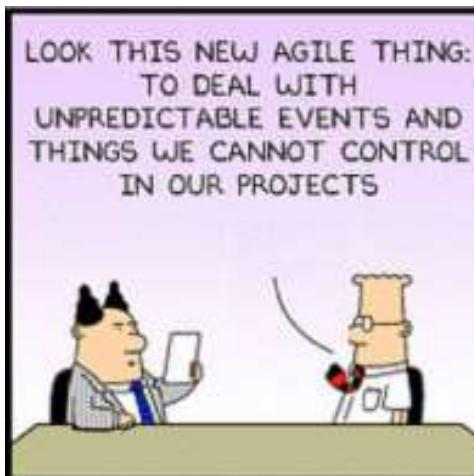


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Ensuring Agile Success

- Managing Change
- Agile Success Factors
- Agile Evolving with Times



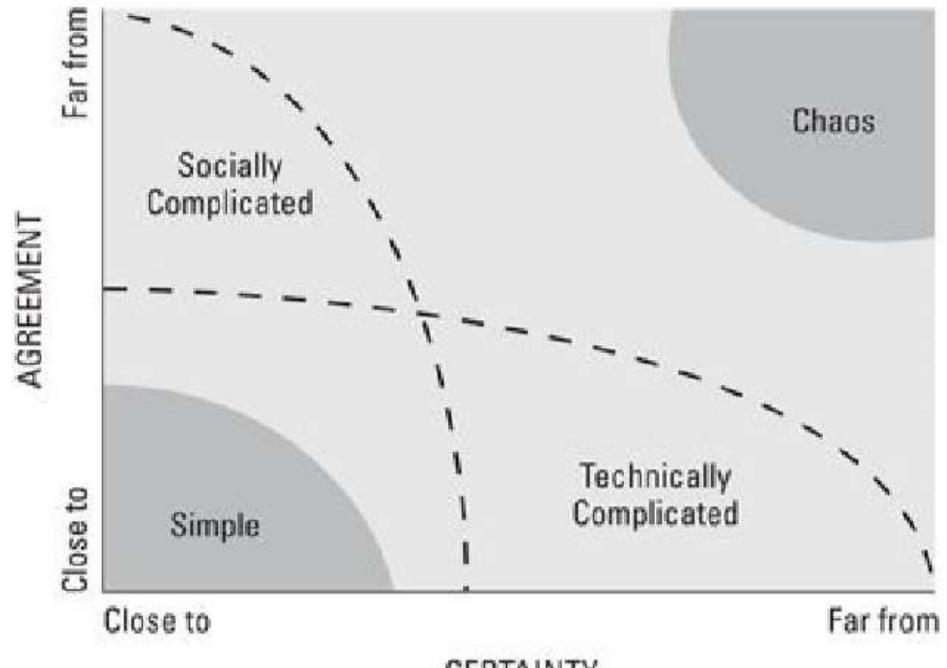
10 Key Factors for Agile Success

- **Dedicated Team Members** (“Task switching wastes 40% of time”)
 - Team members should be dedicated — product owner, development team members, as well as scrum master — to a single project at a time
- **Collocation**
 - The Agile Manifesto lists individuals and interactions as the first value. The way you get this value right is by collocating team members to be able to have clear, effective, and direct communication throughout a project.
- **Automated Testing**
 - Development teams cannot develop at the rate technology and market conditions change if they have to manually test their work every time they integrate new pieces of functionality throughout the sprint.
- **Enforced Definition of Done**
 - Ending sprints with non-shippable functionality is an anti-pattern to becoming more agile. Your definition of done should clarify the following: The environment in which the functionality should be integrated; The types of testing; The types of required documentation
- **Clear Product Vision and Roadmap** (“The best requirements, and designs emerge from self-organizing teams”)
 - Product owner owns the Product Vision and Roadmap; however, ownership is with all the members;
- **Product Owner Empowerment** (The product owner’s role is to optimize the value produced by the development team)
- **Developer Versatility**
- **Scrum Master Clout (empowered Scrum Master to work with leadership of the organization)**
 - Scrum master empowered by leadership to work with members of the scrum team, stakeholders, and other third parties to remove roadblocks
- **Management Support for Learning**
- **Transition Support**
 - Coaching at leadership and team levels increases chances to succeed (training, one-on-one mentoring for specific role-based challenges)

Getting Commitment to Agile Transition

- **Identify an Agile Champion:** a senior-level manager or executive who can help ensure organizational change; the fundamental process changes that accompany agile transitions require support from the people who make and enforce business decisions and a good agile champion is able to rally the organization and its people around process changes.
- **Identify Problems** that can be best be solved using Agile Methods: Illustrate how Agile can provide greater benefits by addressing the existing issues in the current projects:
 - **Profit benefits:** Agile approaches allow project teams to deliver products to market quicker than with traditional approaches. Agile organizations can realize higher return on investment.
 - **Defect reduction:** Quality is a key part of agile approaches. Proactive quality measures, continuous integration and testing, and continuous improvement all contribute to higher-quality products.
 - **Improved morale:** Agile practices such as sustainable development and self-managing development teams can mean happier employees, improved efficiency, and less company turnover.
 - **Happier customers:** Agile projects often have higher customer satisfaction because agile project teams produce working products quickly, can respond to change, and collaborate with customers as partners.

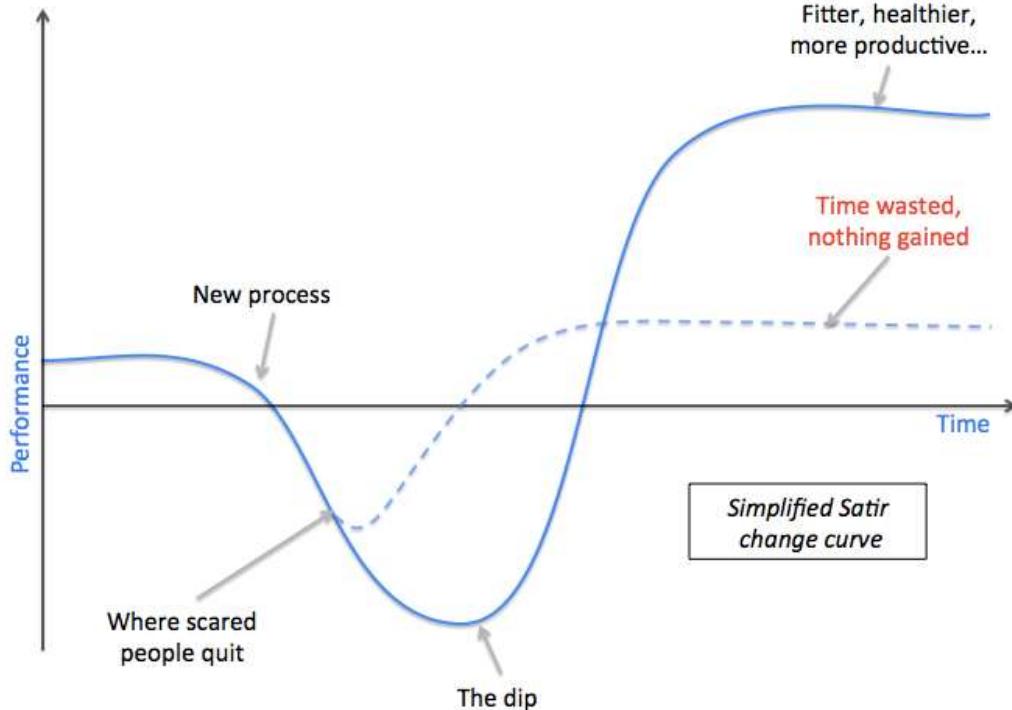
Projects that can benefit from going Agile



Agile's benefits are most evident in these conditions

Source: (T2-Chap17)

“Change is not Smooth” (Satir’s Curve)



Source: (T2-Chap17)

Managing Change (12 Steps)

1. **Conduct Implementation Strategy** by Identifying: Success Factors; Current Process vs. Agile Methodology; Step-by-Step Plan to implement Agile; Benefits and Challenges;
2. **Establish a Transformation Team**: Responsible for Agile transformation at the organization level (a Champion with Management Support); Focus on areas of Change for Agile Transformation
3. **Build Awareness and Excitement**: Educate people across the organization on the benefits of Agile through wide communication via newsletters, town-hall meetings and encouraging pioneers, etc.
4. **Identify a Pilot Project**: Select a small project which is important, visible, clear and containable and measurable
5. **Identify Success Metrics**: Rate of Sprint Goal success, No. of defects, Responding to change, Income generation potential, Customer satisfaction, Stakeholder Happiness,...
6. **Train Sufficiently**: Training by an Agile coach in a face-to-face workshop setting, work-exercises based on real-life challenges
7. **Develop a Product Strategy**: Start with Product Vision and Roadmap exercise
8. **Develop Product Roadmap**, Product Backlog, Sprint Backlog and Estimates
9. **Run the First Sprint**
10. **Gather Feedback, Analyze and Improve**
11. **Maturity by Constant Inspection and Adpatation**
12. **Scale Vertically** by creating Evangelists / Agile Ambassadors

Agile Success Metrics

- ✓ How often did the scrum team meet sprint goals? Did the rate of sprint goal success rise throughout the project?
- ✓ Did the number of defects in each sprint decrease throughout the project? How much time lapsed between finding and fixing defects?
- ✓ How soon was the scrum team able to release a valuable product to the marketplace? How often did the scrum team provide valuable updates?
- ✓ If the product generated income, when did the first dollar come in? What was the overall return on investment?
- ✓ How did the agile project time to market and return on investment compare with that of past projects using the company's old methodologies?
- ✓ Is the customer happy? Are stakeholders happy? Did customer and/or stakeholder satisfaction increase throughout the project?
- ✓ Did scrum team member satisfaction increase throughout the project? What other types of metrics does your organization value?
- ✓ Can your project demonstrate any specific company goals?

Agile is an Attitude, Not a Technique...

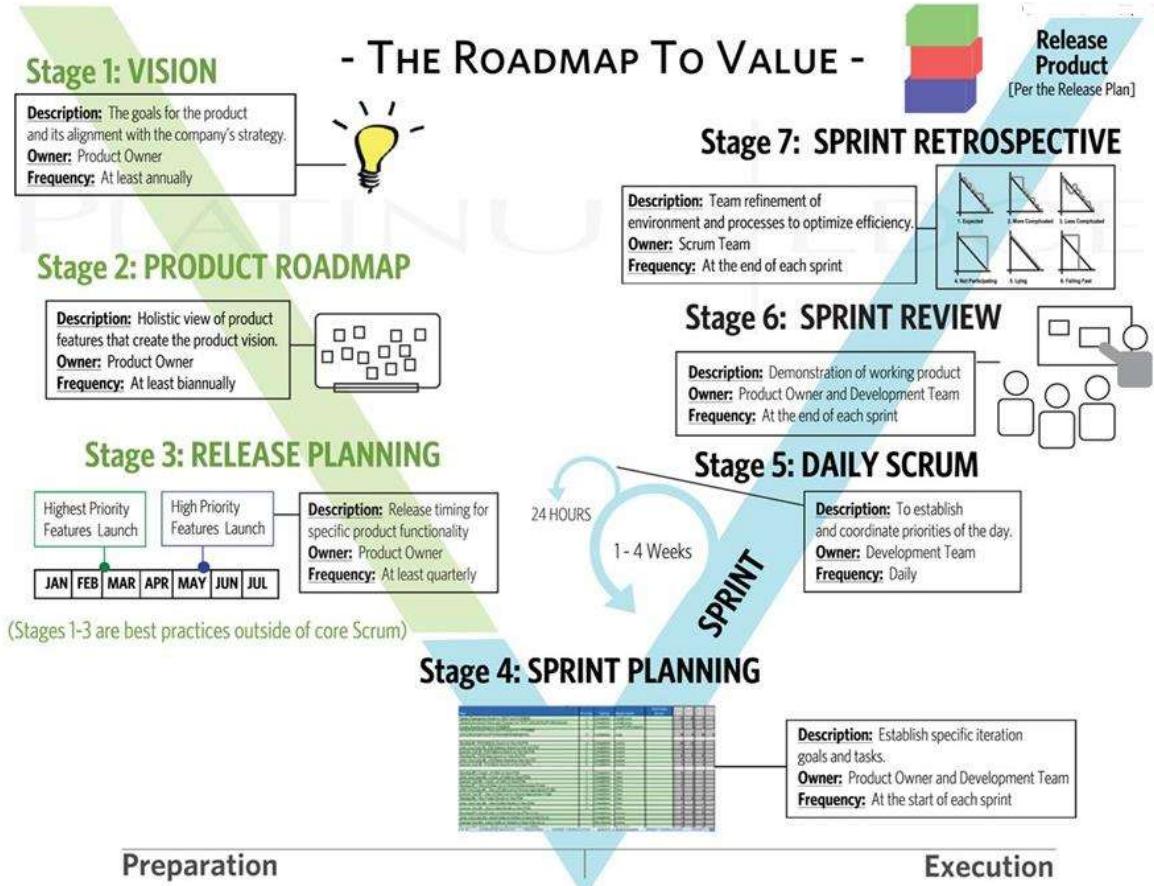
“Agile...is an attitude, not a technique with boundaries. An attitude has no boundaries, so we wouldn’t ask ‘can I use agile here’, but rather ‘how would I act in the agile way here?’ or ‘how agile can we be, here?’”

- *Alistair Cockburn (Agile Guru)*

Evolution of Agile with the Times ➔➔

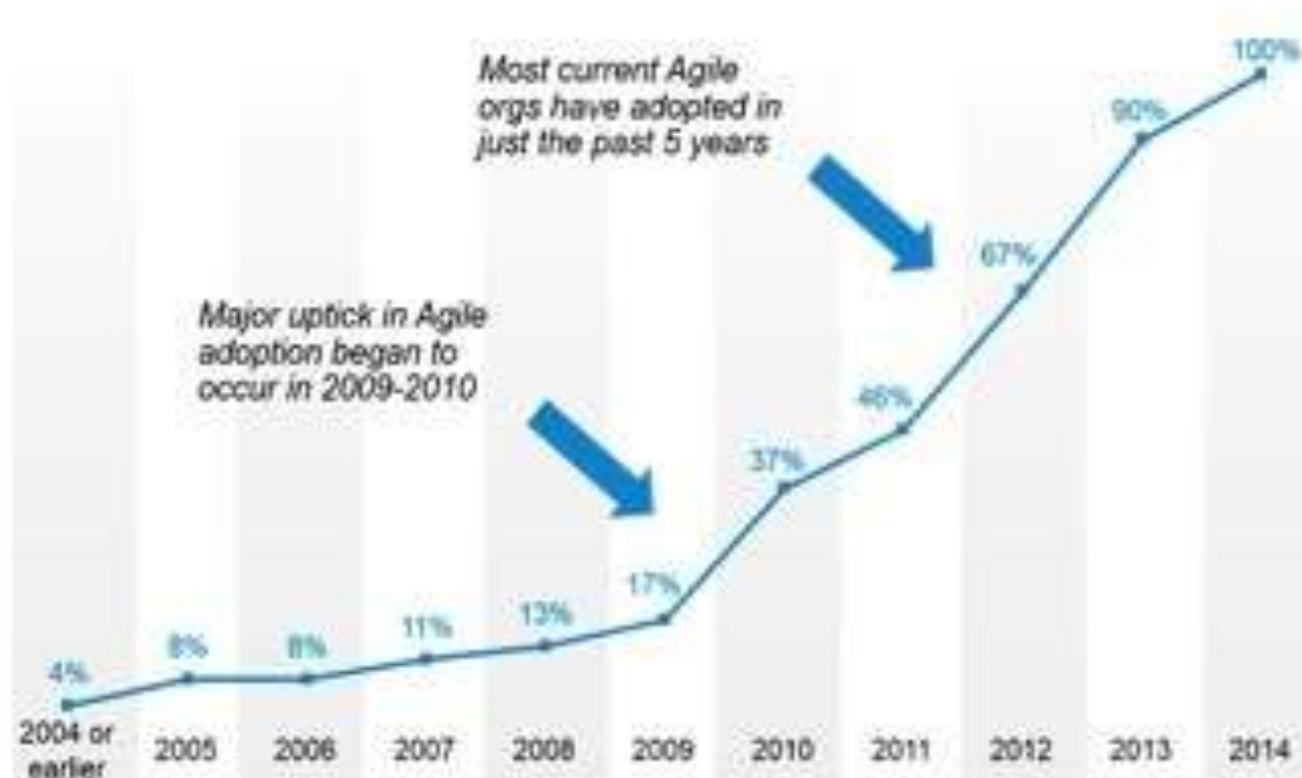


Agile is a *Natural Evolution* towards Survival in this Digital Era, and Scrum is just the Beginning...



Source: (T2)

Agile Adoption in Recent Years...

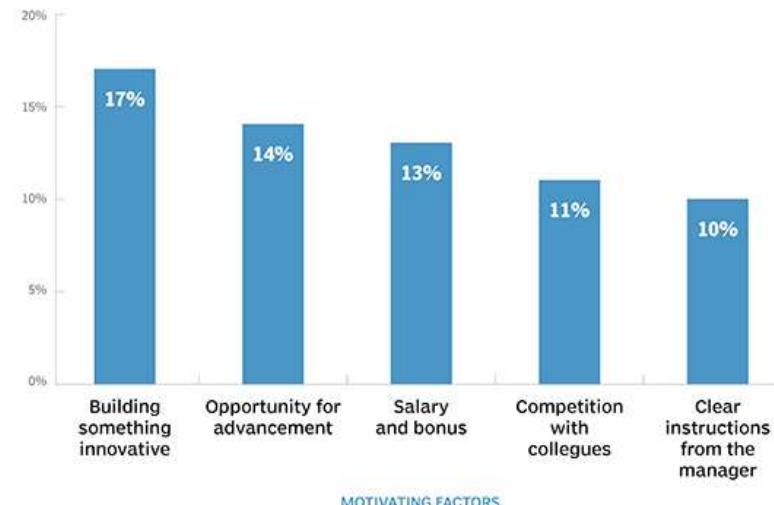


Source: Tech Beacon

Agile-driven Transformations

- Agile + DevOps → AgileDevOps
 - (Development + Operations) /w Daily build (CI) = DevOps
 - Evolving with Cloud, Mobile Infrastructure & SoA (Microservices Architecture)
 - Infrastructure + Programming Frameworks → Containerized Apps (infrastructure independent development/deployment on Cloud)
 - Code heavy Development → Low-code/No-code Platforms/API/Microservices-driven development frameworks
- Agile Software Processes → Agile Methods in any Product Development
- Transformation of Organization Structures: Top-down, Hierarchical and Networked Structures → Self-organizing Empowered Agile Teams
- Agile is the **language of Startups** who dream game-changing products and services in this digital era

What drives high-quality work



Source: 2017 Survey of 5000 Development Professionals by CAST

Let's “Google-walk” the Agile

agile methods

All Images News Videos Maps More Settings Tools

About 7,27,00,000 results (0.47 seconds)

“Agile Development” is an umbrella term for several iterative and incremental software development methodologies. The most popular agile methodologies include Extreme Programming (XP), Scrum, Crystal, Dynamic Systems Development Method (DSDM), Lean Development, and Feature-Driven Development (FDD).

What is Agile? Learn About Agile Software Development - VersionOne
<https://www.versionone.com/agile-101/>

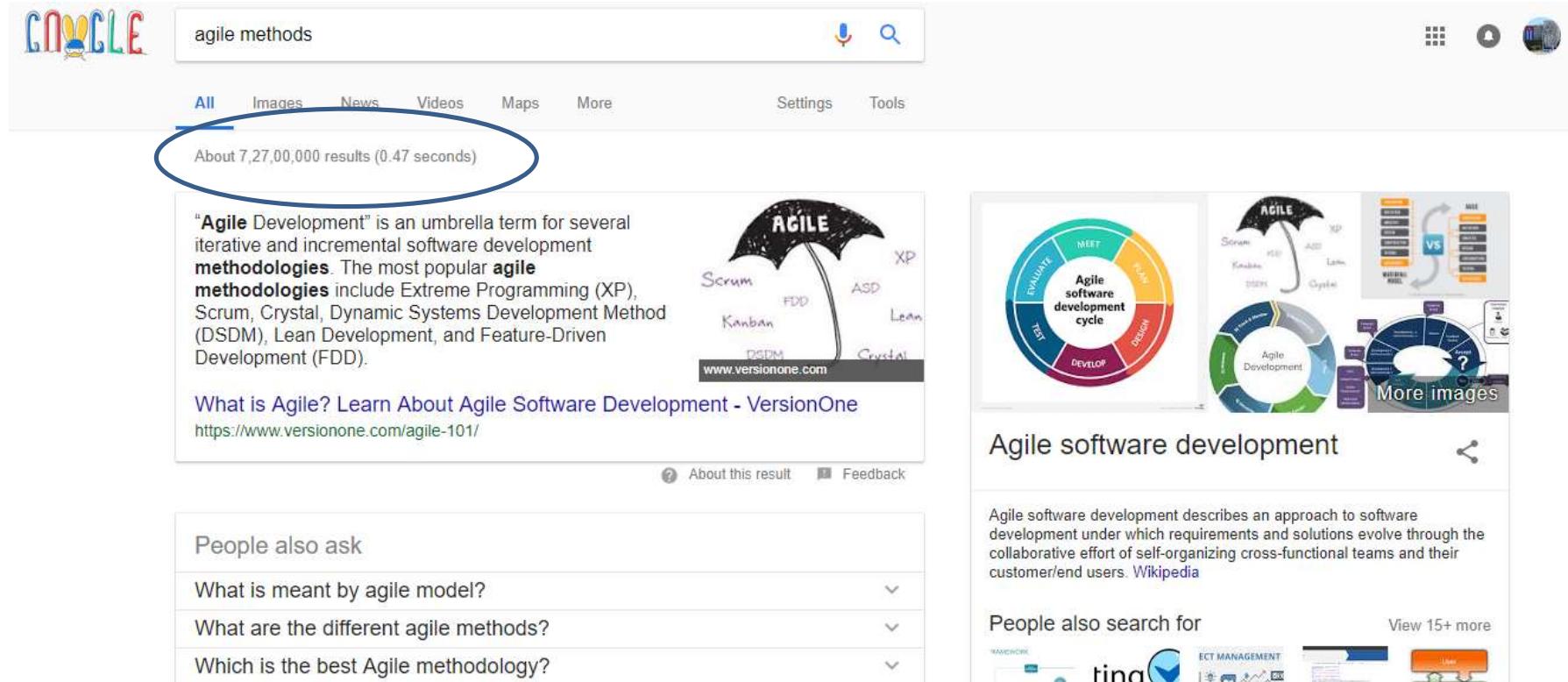
About this result Feedback

People also ask

What is meant by agile model? ▾

What are the different agile methods? ▾

Which is the best Agile methodology? ▾



Summary: Ensuring Agile Success

- Transform Organizational Culture towards Agile Teams (micromanaging → self-managing teams)
- Managing Change with a dedicated Agile Champion (Agile Coach) for driving transformation with communication and training
- Start with identifying a Simple Lighthouse Project as a ripe candidate for going Agile
- In these times of rapid digital transformation, evolve Agile Practices in tune with technology/business trends

Let's look forward to the day when the words 'Agile' and 'Customer Orientation' would disappear or get subsumed the way of 'Quality' and 'Management' in Organizational Vocabulary !!!!

Thank You

© Copyrights of original Authors are duly acknowledged

™ ® All Trademarks, Registered Trademarks referred in this document are the property of their respective owners