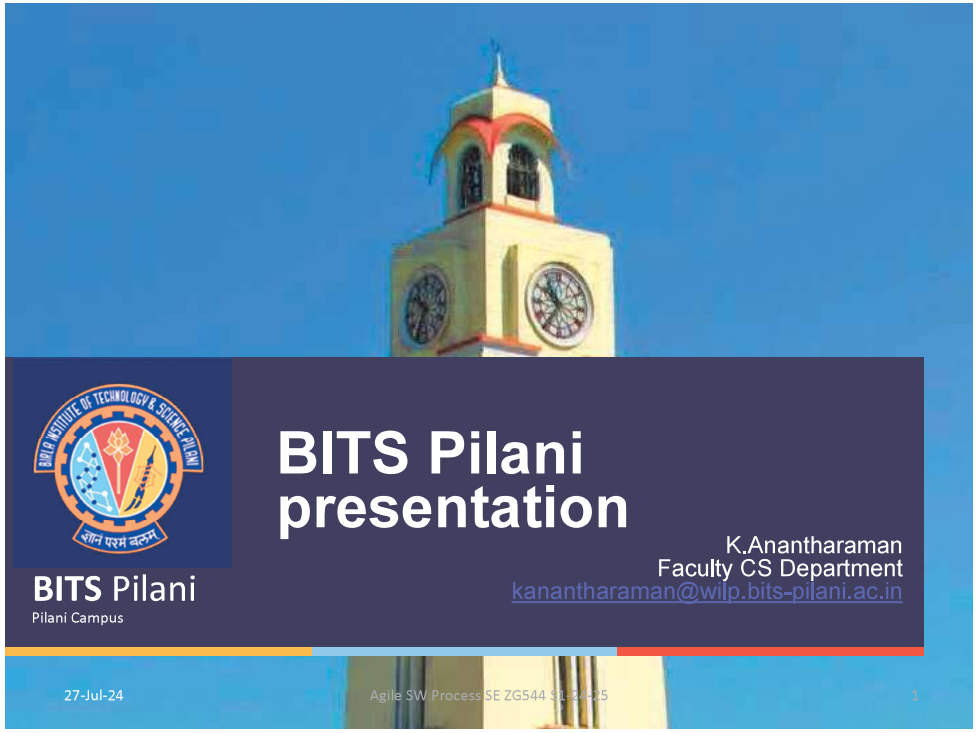
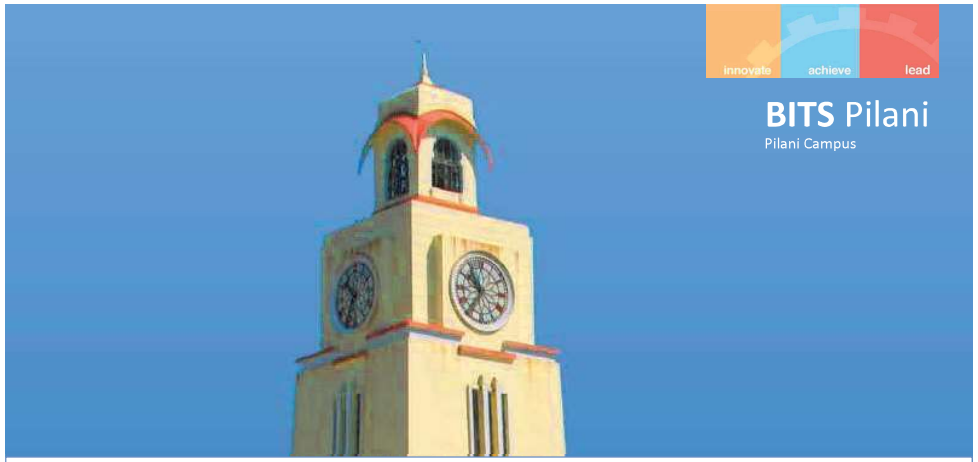


# BITS Pilani presentation

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

BITS Pilani
Pilani Campus





BITS Pilani
Pilani Campus

## SE ZG544 S1-24-25 , Agile Software Processes 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

**Evaluation Scheme**

| EC # | Name | Type | Weight | Duration | Schedule |
|---|---|---|---|---|---|
| EC-1A | Quiz-1 | Online | 5% | 1 Week | September 1-10, 2024 |
| EC-1B | Quiz-2 | Online | 5% | 1 Week | October 10-20, 2024 |
| EC–1C | Assignment | Take-home | 15% | 2-4 Weeks | November 1-10, 2024 |
| EC-2 | Mid-Sem Exam. | Closed Book | 35% | 2 Hrs. | Saturday, 21/09/2024 (AN) |
| EC-3 | End-Sem Exam. | Open Book | 40% | 2 ½ Hrs. | Saturday, 30/11/2024 (AN) |



## Additional Reference Books

1.Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)

2.Head First Agile by Jennifer Greene; Andrew Stellman Published by O'Reilly Media, Inc., 2017 (Scrum)

3.Introduction to Agile Methods by Sondra Ashmore Ph.D.; Kristin Runyan Published by Addison-Wesley Professional, 2014 (XP)

# Poll

- https://forms.gle/wRadsyQREA3BpkE26

Objective:
- Online poll to understand the expectations of students attending this course.
- Link



# Module-1 – Topics

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



# Basic Project Management concepts

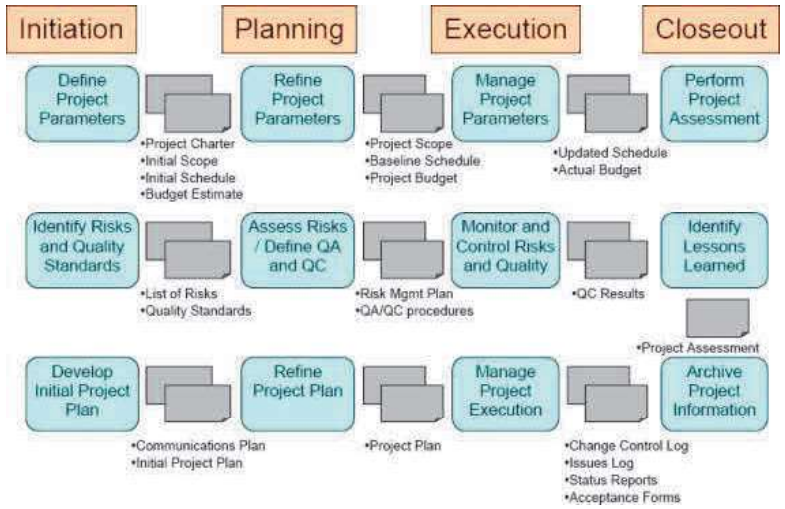
- What is a Project?
  - Definite Start-End date, Temporary, Scope(Produce Specific result) , Budget/Effort – Example: Building a house
- Project Management Life Cycle Phases
  - Initiation, Planning, Execution, Closeout, Monitoring & Control
- System Development Life Cycle/phases (SDLC)
  - Requirements,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



# 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



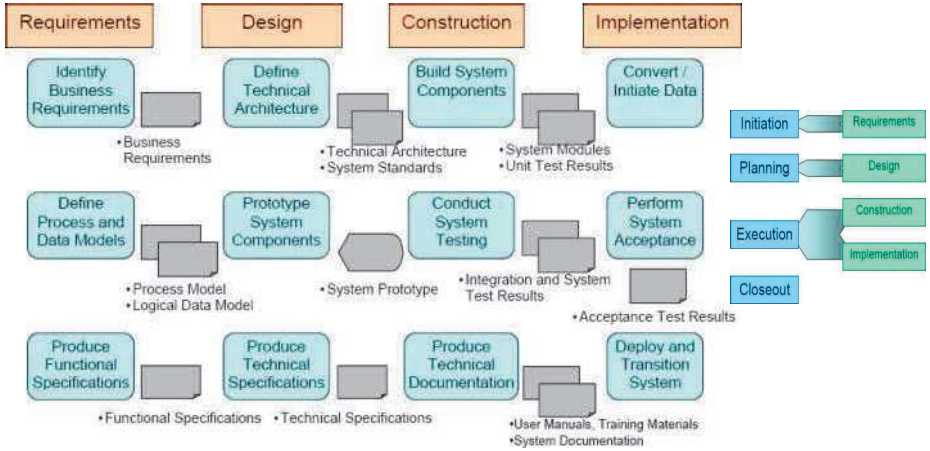

https://www.pmi.org/learning/library/project-managing-sdlc-8232




# Project Management Model Water Fall Model and Agile



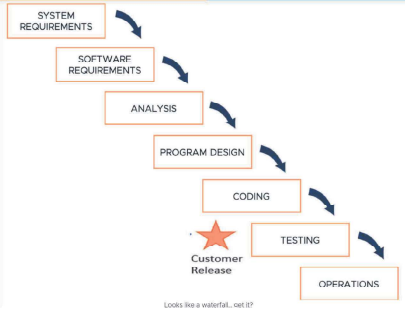
# System Development Phases (Engineering activities)




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



# Waterfall Approach to Software Development




- **Waterfall/Predictive/Traditional** ( Different terminologies that refer to same approach)
- Phases & Phase Gate
- 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
- Customer Release-Value realization

https://www.beyond20.com/blog/when-to-use-agile-and-when-to-use-waterfall-when-managing-projects/

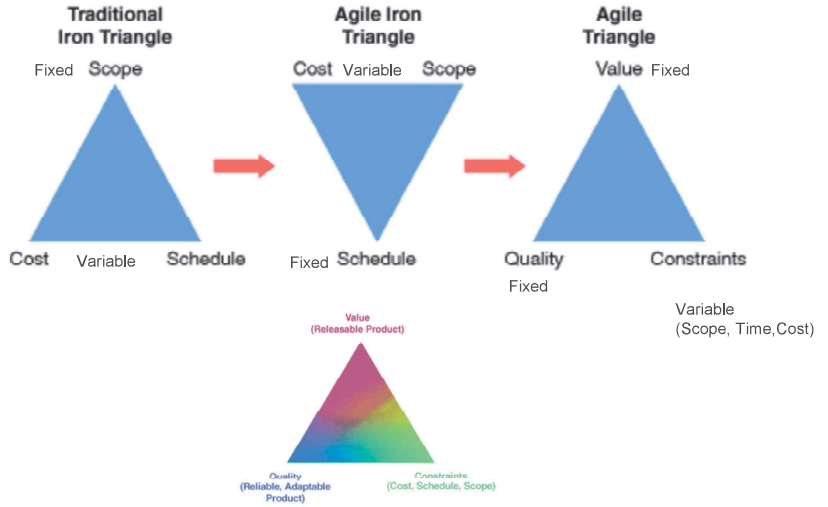
# Agile Approach to Software Development


**Agile/Adaptive/Iterative & Incremental**
( Different terminologies that refer to same approach)
- Sprints & Sprint Review
- Design, Coding and Testing in each iteration in any order
- Origin from lean manufacturing

- **Rolling Wave Planning**
- **Less documentation**
- **Negotiable feature sets**
- Minimum process and tools
- Customer Release-Value realization in each iteration



# Iron Triangle of Project management The Evolution to an Agile Triangle



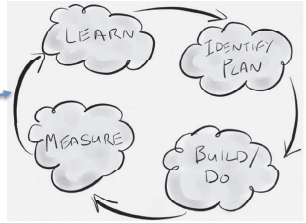

Reference: Agile Project Management: Creating Innovative Products, Second Edition, Jim Highsmith, Published by Addison-Wesley Professional



# 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

PDCA Cycle






# Questions?

Link – 5 Sections

# Advantages and Disadvantages of Waterfall

**Advantages:**

- Sequential, Upfront planning
- Good Documentation
- Scope of work is generally fixed

**Disadvantages:**

- Error propagation
- Missing requirements
- Error correction is costly
- Late customer feedback



# Application of Waterfall Model

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



# Advantages and Disadvantages of Agile Model

**Advantages:**

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

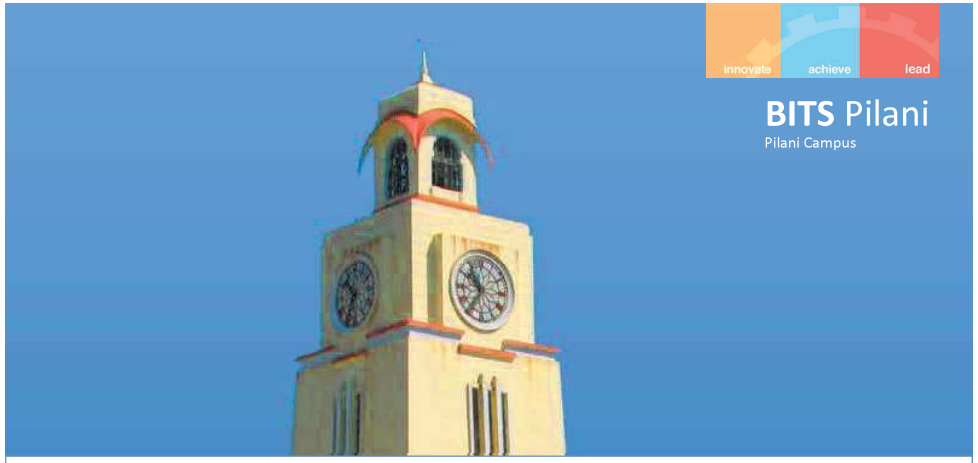
**Disadvantages:**

- Poor Resource planning
- Less Documentation
- Fragmented output



# Application of Waterfall 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



innovate achieve lead

BITS Pilani
Pilani Campus

# Need for Agile Methods



innovate achieve lead

BITS Pilani
Pilani Campus

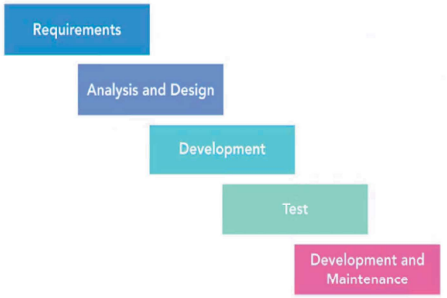
# Traditional Software Development Approaches

# Traditional Software Development Model – Waterfall Model



innovate achieve lead

# 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



innovate achieve lead

# 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



# 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.



# 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

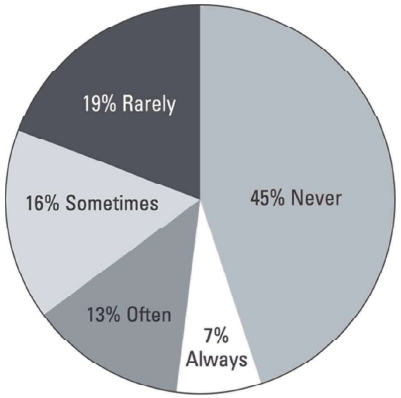
➢ More time should be spent on delivering working software features early and often. And enlisting customer feedback

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



# 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.

Ref: Agile Project Management for Dummies - Mark C. Layton John Wiley & Sons - 2012



# 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.**

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



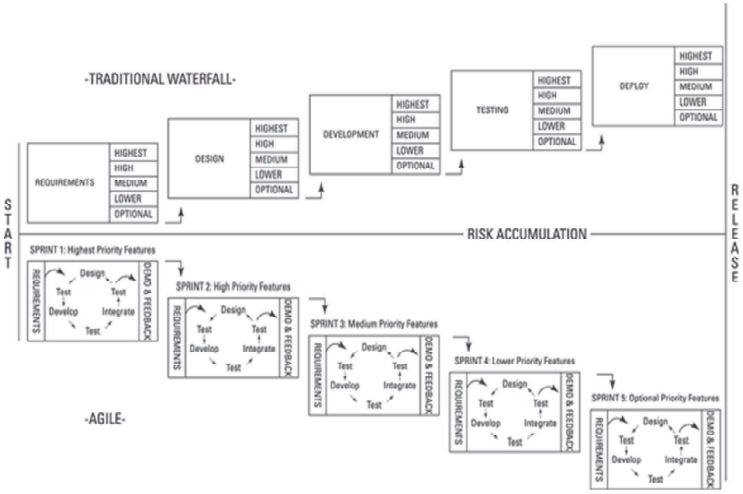
# Definable Work

- 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**.

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



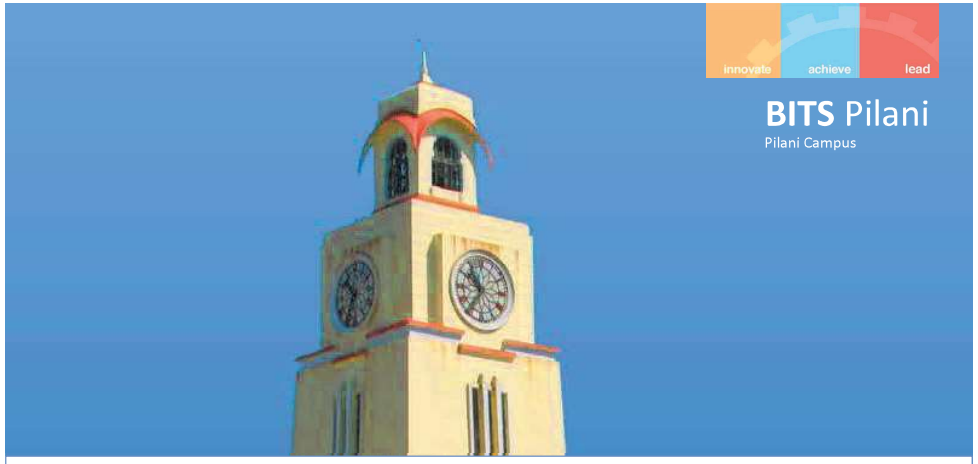
# Waterfall vs agile project




**Mixing traditional project management methods with agile approaches:**

The answer, of course, is **you can.** If you fully commit to an agile approach, you will have a better chance of deriving benefits of Agile project

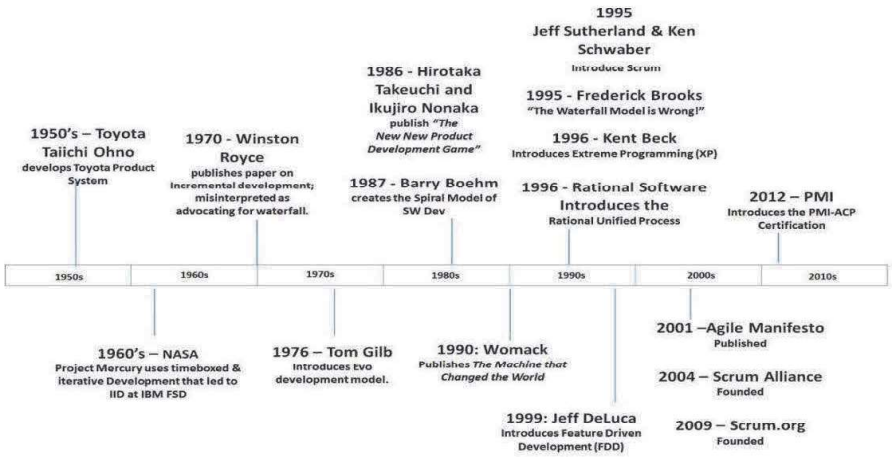
Ref: Agile Project Management for Dummies - Mark C. Layton John Wiley & Sons - 2012




# Evolution of Agile Project Management



# Evolution of Agile Frameworks




Source Material e-learn(Recorded)



# 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 …

- 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***



# 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.**

Ref: Agile Project Management for Dummies - Mark C. Layton John Wiley & Sons - 2012

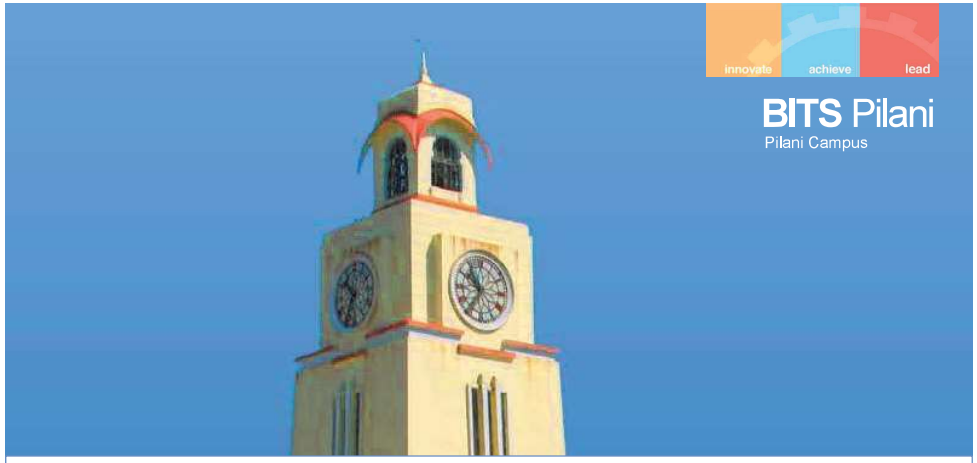


# Why Agile Projects Work Better …

- 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.

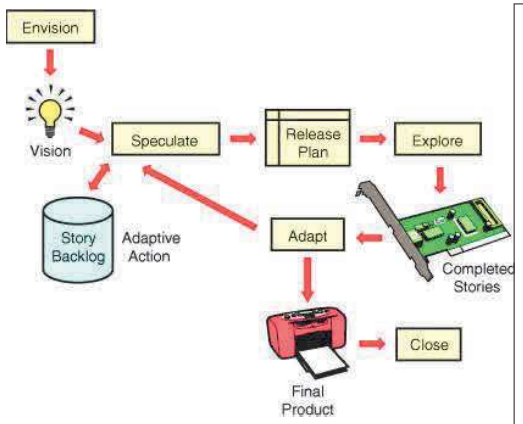
Ref: Agile Project Management for Dummies - Mark C. Layton John Wiley & Sons - 2012






# Agile Project Management Framework

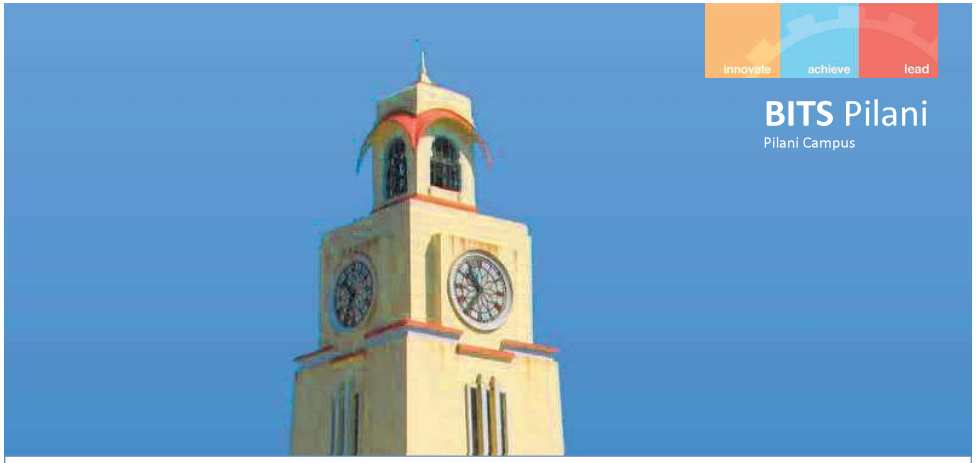
# Agile Delivery Framework (APM)




1. **Envision:** Determine the product vision and project objectives and constraints, the project community, and how the team will work together
2. **Speculate:** Develop a capability and/or feature-based release plan to deliver on the vision
3. **Explore:** Plan and deliver running tested stories in a short iteration, constantly seeking to reduce the risk and uncertainty of the project
4. **Adapt:** Review the delivered results, the current situation, and the team's performance, and adapt as necessary
5. **Close:** Conclude the project, pass along key learnings, and celebrate.

The departure from traditional phase names—such as Initiate, Plan, Define, Design, Build, Test—is significant.

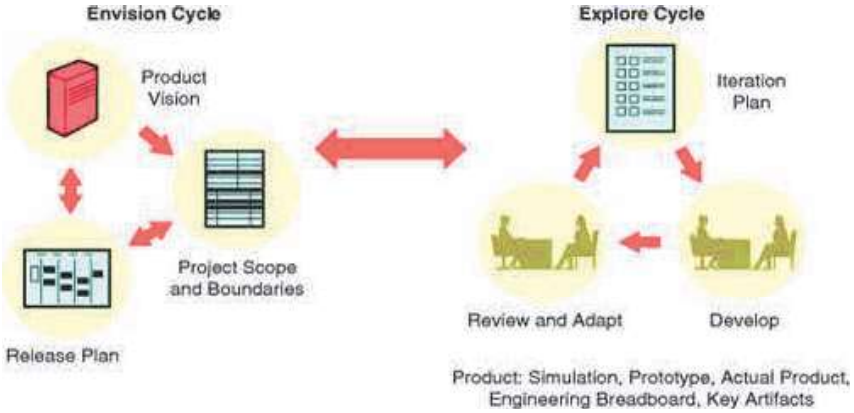





# Benefits & Challenges of Agile Methods



# APM's Envision and Explore Cycles






# 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 …

***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 …

***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 …

***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.

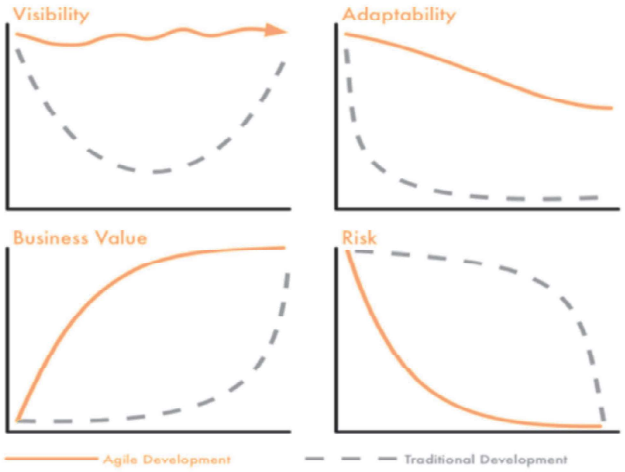
***Constantly measurable quality***: involves creating *active checkpoints* where organizations can assess outputs against both qualitative and quantitative measurements.

***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.



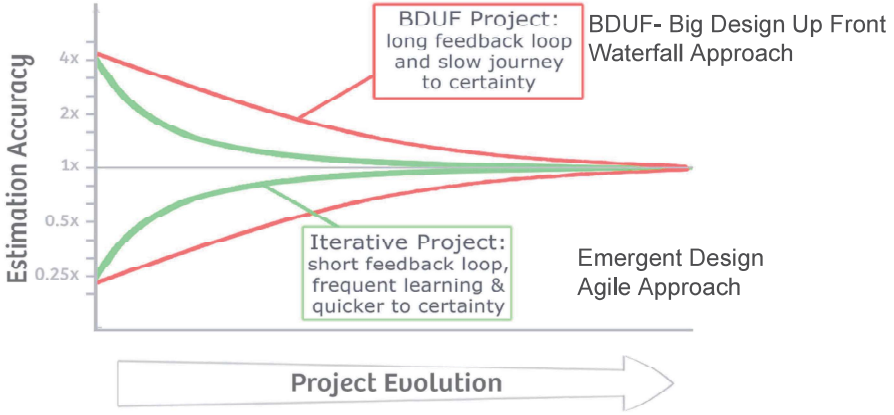
# Other benefits of agile approach




Reference: https://www.beyond20.com/blog/when-to-use-agile-and-when-to-use-waterfall-when-managing-projects



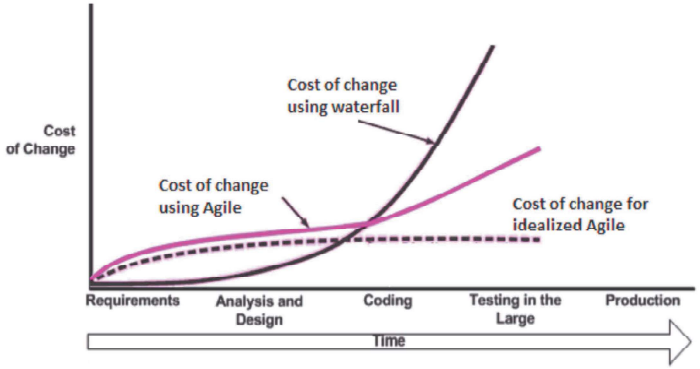
# Cone of Uncertainty




Reference/: https://agilecoffee.com/wp-content/uploads/2016/12/07-cone-of-uncertainty.jpg



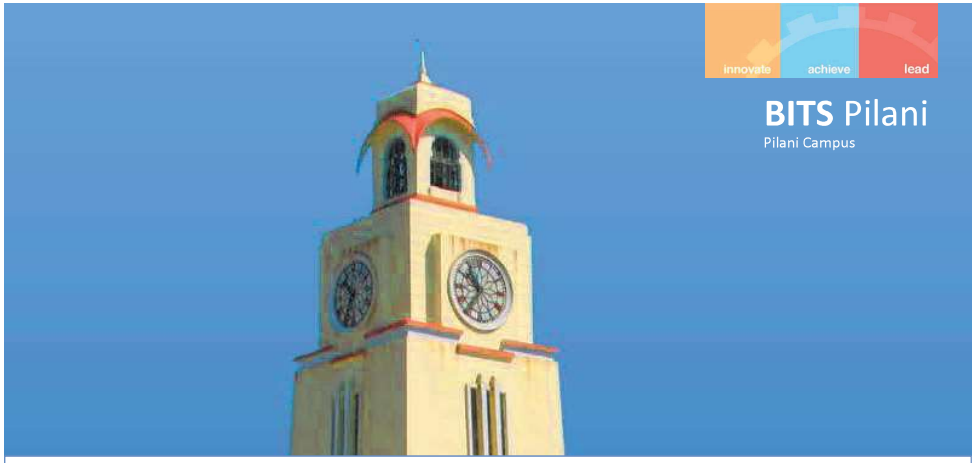
# Cost of change




https://www.researchgate.net/figure/Cost-change-curve-of-traditional-and-agile-methodology-23_fig9_312564218

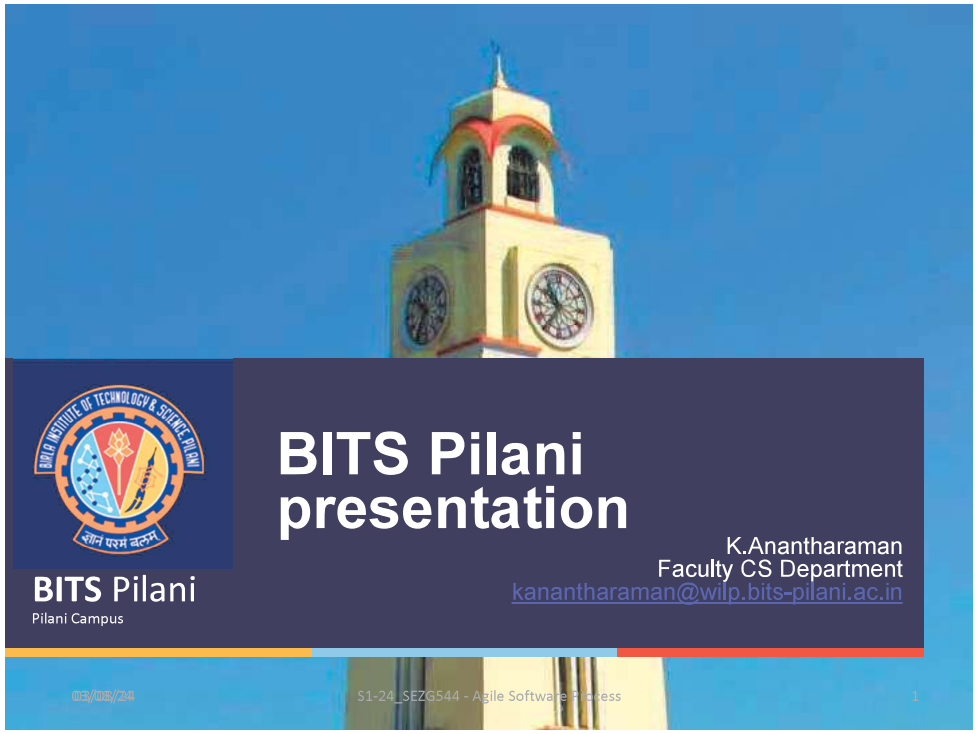
Thank you






# SE ZG544 , Agile Software Process Lecture No. 2 – Module-2 : Agile Software Development





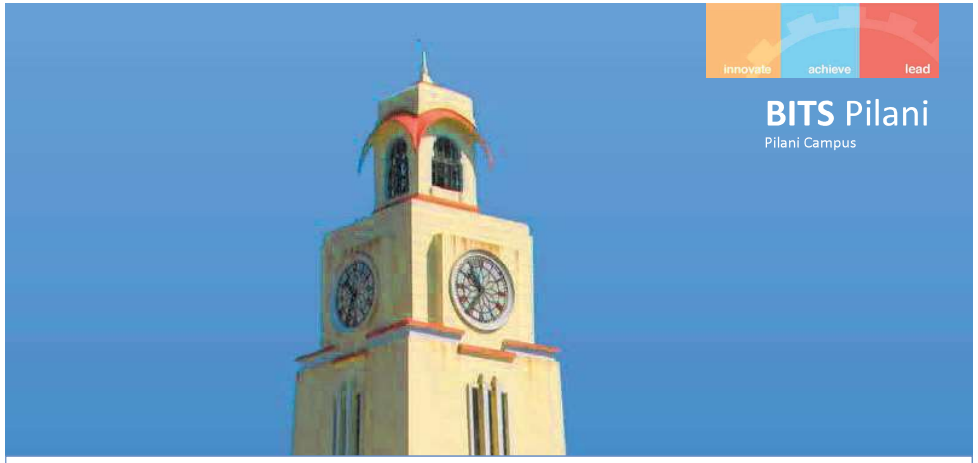

# BITS Pilani presentation

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



## Module-2 Topics

1. Project Development Life Cycle
   a) Predictive life cycle
   b) Iterative life cycle
   c) Incremental life cycle
   d) Agile/Adaptive life cycle
2. Spiral Model
3. Rational Unified Process
4. Early Agile Methods
5. Agile Mindset

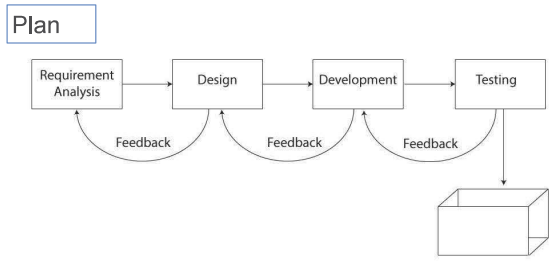



# Project Life cycle models



## 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




- 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/



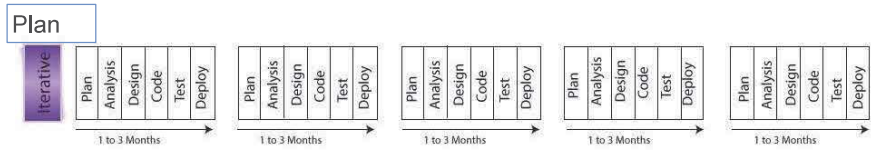
## Life Cycle

- 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



## 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.




- 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

Ref: https://www.izenbridge.com/

# Examples of Iterative Development

- When you are getting a customized coat made
  - You may be required to go for a trail 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

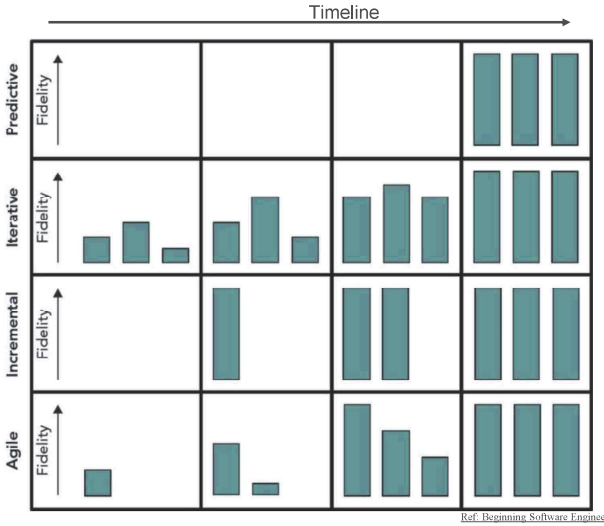
Source: https://www.izenbridge.com/



# Project Life Cycle Models

- Different development approaches add features and increase fidelity in different ways.

Fidelity: the degree of exactness with which something is copied or reproduced.




Goals/Characteristics

Cost, Requirements-Fixed, Single Delivery

Accuracy/Correctness, final single delivery
Intermediate delivery – may no be usable
**Example**: Customized outfit/coat, Website

Accuracy/Correctness, Multiple Delivery,
Intermediate delivery –usable
**Example**: Visit to a restaurant

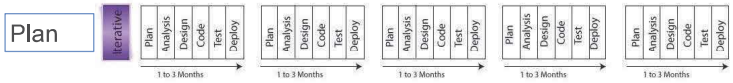
Accuracy/Correctness, Multiple Delivery,
Intermediate delivery –usable
**Example**: Game: Treasure Hunt with clues

Ref: Beginning Software Engineering by Rod Stephens



# 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.

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



# Agile Life Cycle Models

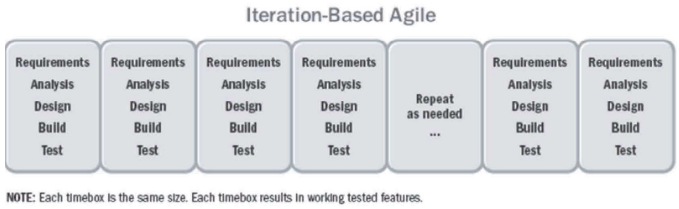
- Iterative
- Flow-Based

# Agile/Adaptive Life Cycle- Iteration Based Agile

innovate achieve lead

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

Plan




Fixed Time box : 1-4 weeks equal duration for each iteration
Goals and Characteristics: Value, Multiple deliveries

Re.f: Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)|



# Project Life Cycles Characteristics

innovate achieve lead

| 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)|

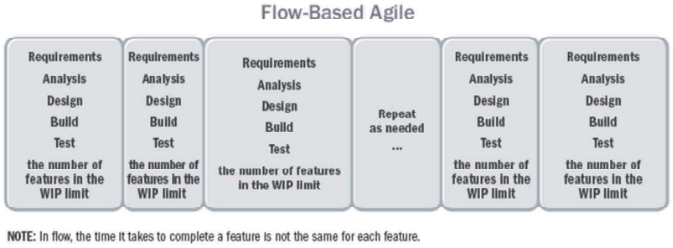


# Agile/Adaptive Life Cycle- Flow-based Based Agile

innovate achieve lead

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

Plan




Variable Time Box :
Goals and Characteristics: Value, Multiple deliveries

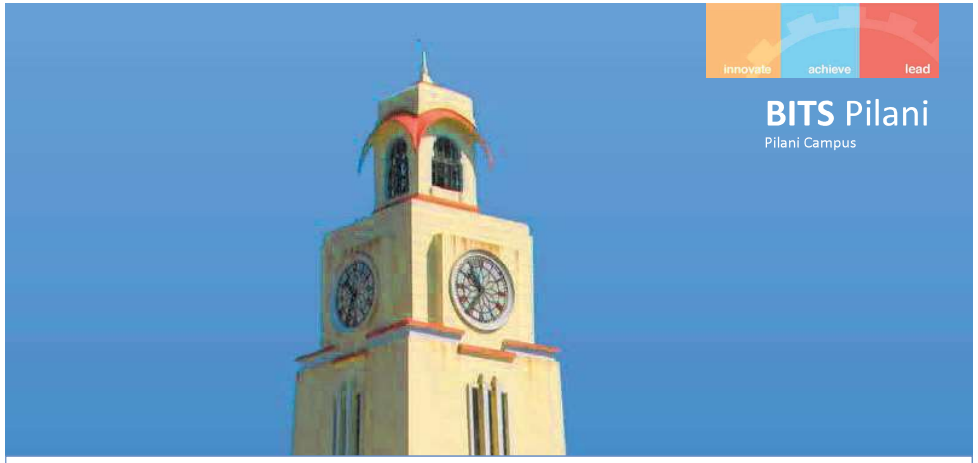
Ref: Agile Practice Guide (ENGLISH) by Project Management Institute Published by Project Management Institute, 2017 (Agile methodologies)|



# Q&A

innovate achieve lead

- Set1




# Some Popular Iteration Models



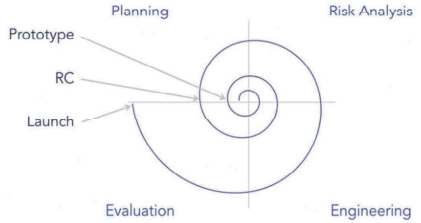
# Rational Unified Process (RUP)

- 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



# Spiral Risk Driven Customer driven Planning



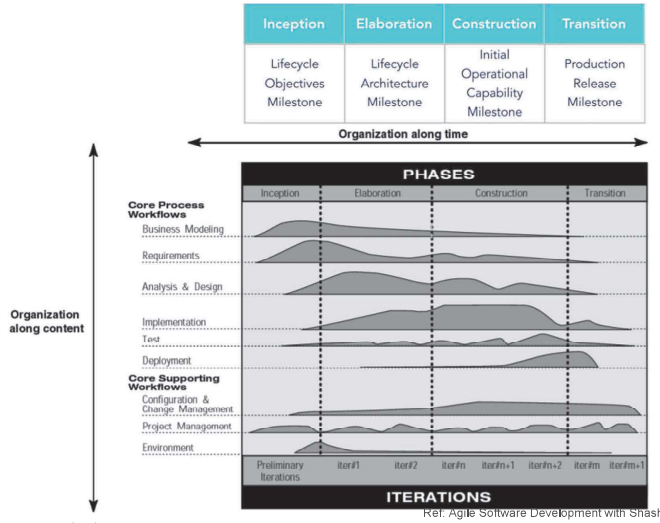

- Developed by Barry Boehm, 1986.
- Easier management of risks (Theme)
- Mix of water fall and iterations
- Y-Axis represents Cost
- X-Axis represents Review
- Prototype-1, Prototype-2 ….
- Operational Prototype
- Final Release

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

Ref: Agile Software Development with Shashi Shekhar,LinkedIn Learning



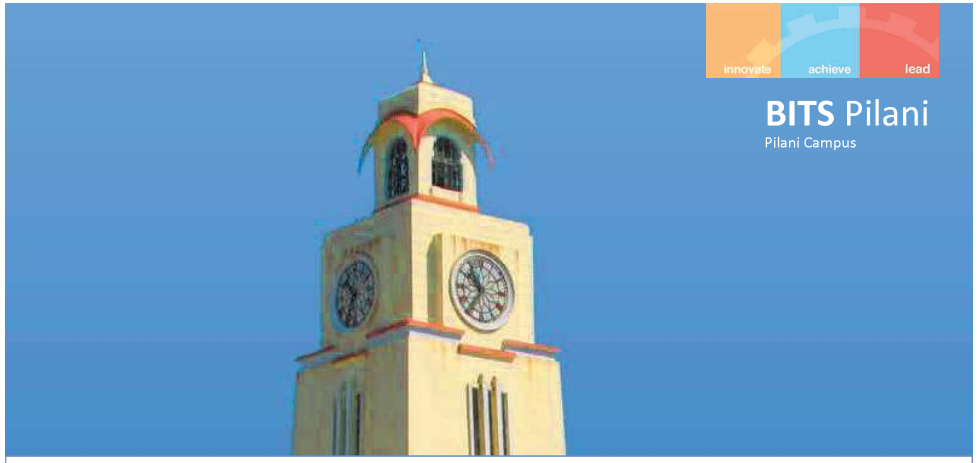
# RUP Iterative Model




X-Axis: RUP Phases, Dynamic

Y-Axis: Organization Process, Static

Ref: Agile Software Development with Shashi Shekhar,LinkedIn Learning




# Early Agile Methods



# 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



# 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



# Crystal Method- Selecting a Model

| Criticality | | | | | | |
|---|---|---|---|---|---|---|
| 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

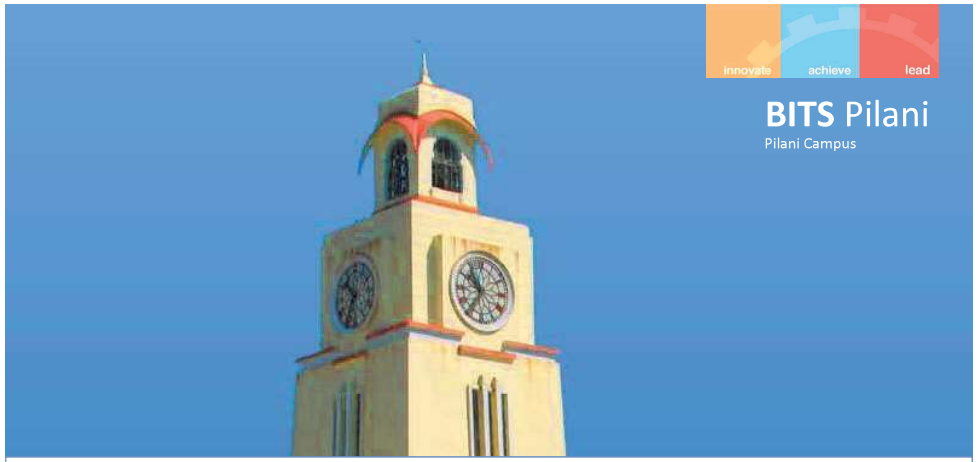
- Crystal methods are people-centric, light-weight, and highly flexible. Focus on People, Interactions,Colloborations.
- Developed by Alistair Cockburn , 1991

Ref: Agile Software Development with Shashi Shekhar,LinkedIn Learning

# Q&A

- Set2






# Project Classifications/Decision making models



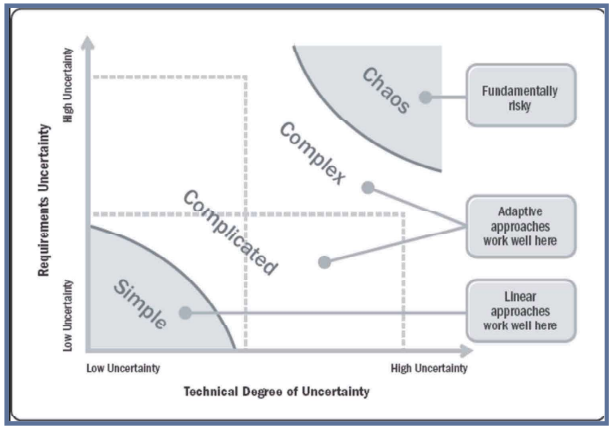
# Decision Making Models

- **Stacey's Complexity Model:** Organizes problems into four categories (Simple, Complicated, Complex, Chaotic) based on their level of clarity and predictability.
- **Cynefin framework:** Helps leaders make decisions by categorizing problems into domains (Simple, Complicated, Complex, Chaotic, Disorder) based on their nature and relationships between cause and effect.

- **Use cases of these Models:**
  - Leaders can use these models to understand the nature of a problem they face and make appropriate decisions.
  - Project managers can use the models to select the most suitable approach based on the complexity of their projects.
  - Teams can use the frameworks to recognize when a problem is shifting from one domain to another, requiring a change in their approach.



# Agile Suitability - Project Environment Stacey's Complexity Model




**Project Examples:**
•**Simple:** Setting up a new employee onboarding process with clear steps to follow.
•**Complicated:** Building a large-scale IT infrastructure with the help of experienced IT consultants.

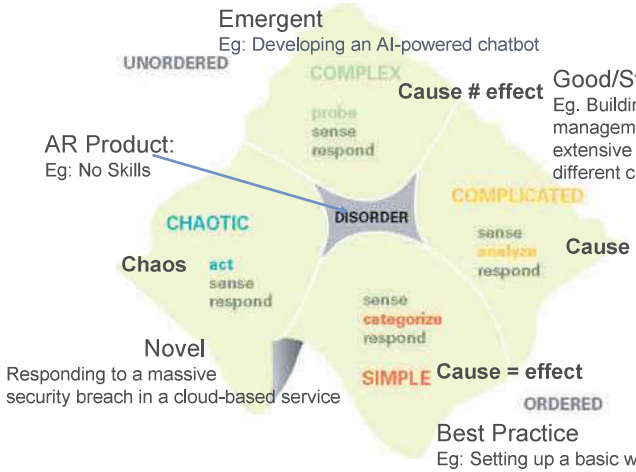
•**Complex:** Developing a new product in a rapidly changing market, requiring continuous iteration and learning.
•**Chaotic:** Crisis management during a natural disaster or cyber-attack.

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

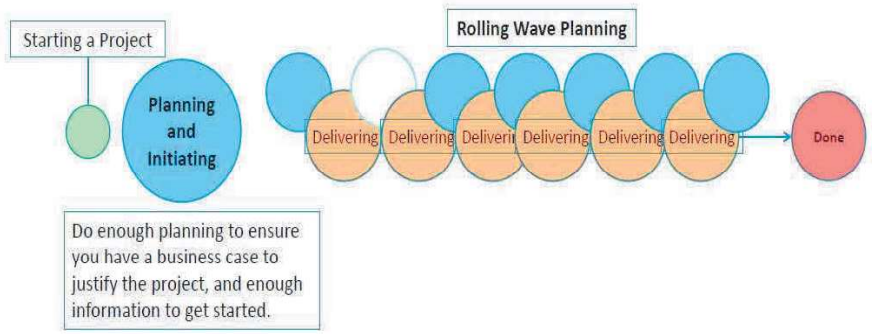
Developed in the early 2000s by David Snodown

Pronounced as: *kun-ev-in*





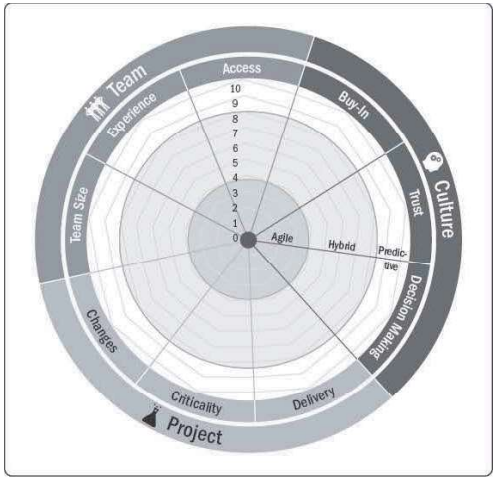

# Rolling Wave Planning or Progressive Elaboration




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



# Agile Suitability Filter



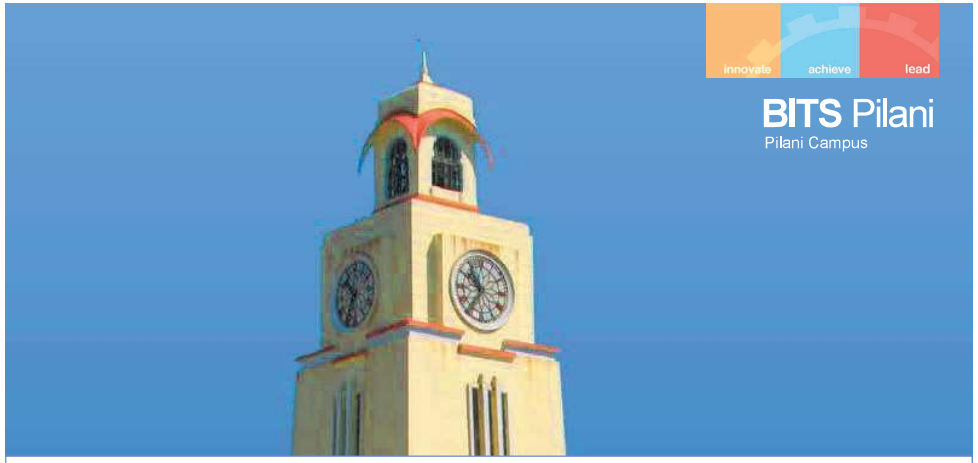

| Attributes | Assessment (wrt Agile) |
|---|---|
| Buy-In | 0-Yes, 5-Partial, 10-No |
| Trust | 0-Yes, 5-Probably, 10-No |
| Decision Making | 0-Yes, 5-Probably, 10-Unlikely |
| Incremental Delivery | 0-Yes, 5-Maybe/Sometimes, 10-Unlikely |
| Criticality | 0-Low, 5-Medium, 10-High |
| Changes | 0-High, 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 |



# Mindset

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

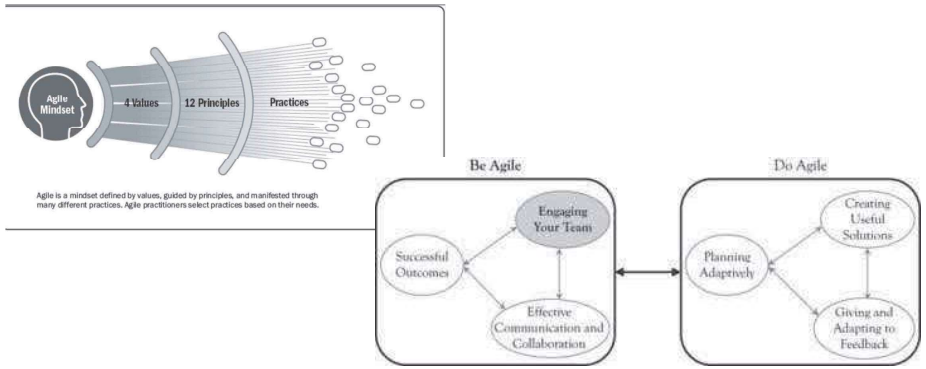
Source: https://www.forbes.com/sites/stevedenning/2019/08/13/understanding-the-agile-mindset/?sh=5a66a5545c17




# Agile Mindset



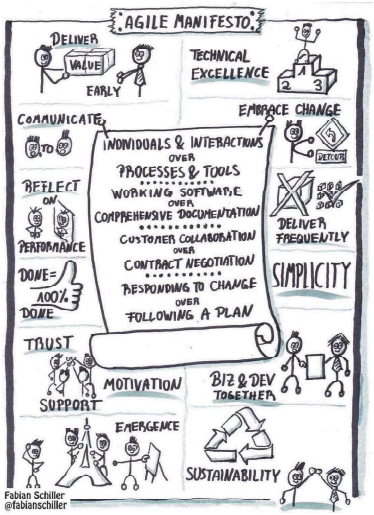
## Agile Mindset (Be Agile and Do Agile)




An Agile mindset implies that an organization or person has absorbed Agile to the extent that it becomes part of their 'identity', i.e. their 'business-as-usual' state and the default way they interact with the world.



## Agile Manifesto & Principles (Basis of Agile Mindset)




- 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
- Then they wrote Agile Principles

**We will discuss this topic in more detail in the next module**



## Agile Mindset …

- It is important to understand that, for **Agile to be successful,** the **right mindset is equally important than merely implementing Agile tools, practices or principles.**
- For example, **having a visual board** does **not necessarily mean that a team is Agile.**
- Experiments have shown that **people can strongly influence** each other to adopt **or reject** an Agile mindset .

# Agile Mindset …

- **Agile is a journey, not a destination**
- Teams become more Agile by **embedding the Agile mindset deeper inside themselves and the organization.**
- This process is **facilitated** by **applying Agile values**, practices, **principles** and so on.



# Agile is not just techniques

- People often prefer to **pick out the elements of a framework they can easily understand** rather than understanding an entire framework and why it should be implemented.
- Some Agile frameworks will only **work effectively if all the key activities, roles and artefacts of the framework are in place.**
  - So, for example, people might think that just prioritizing stories within a backlog is the same as 'being Agile', instead of it being just one part of the whole. Additionally, while prioritization of stories is indeed a core practice in Agile, it is also used in more traditional non-Agile delivery approaches.



# Agile is not just a set of rituals

- Where teams may understand what to do, but don't understand why they are doing it. This will result in **partial success**.
- If this happens teams tend to use Agile practices for a short period of time, but over the **longer term they fall back into their previous ways of working.**
- Because they don't understand why they are doing what they are doing.



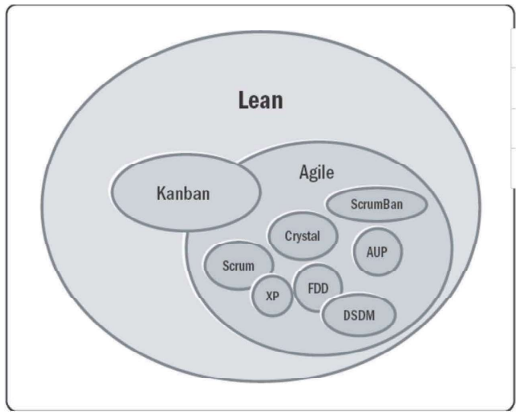
# What is being Agile? Mindset?

| | Fixed mindset | Agile mindset |
|---|---|---|
| *Ability* | Static, like height | Can grow, like muscle |
| *Goal* | To look good | To learn |
| *Challenge* | Avoid | Embrace |
| *Failure* | Defines identity | Provides information |
| *Effort* | For those with no talent | Path to mastery |
| *Reaction to challenge* | Helplessness | Resilience |

- The most important aspect of an Agile mindset is understanding that Agile is neither just a set of rituals that are repeated, nor is it merely based on techniques.

Source: Agile Practice guide, PMI

# Agile is a blanket of many approaches




- Is agile an **approach, a method, a practice**, a **technique**, or a **framework**?
- Any or all of these terms could apply depending on the situation.

Source: Agile Practice guide, PMI



# Q&A

- Set 3



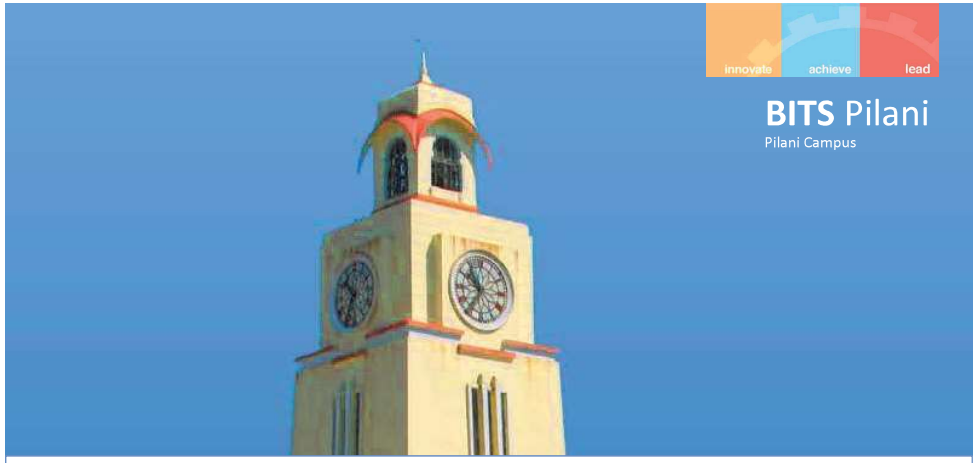
# Two Approaches to fulfill Agile Values and Principles

- **Adopt a formal agile approach** (Designed and Proven)
  - Take time to understand the agile approaches before changing or tailoring them. Premature and haphazard tailoring can limit benefits.
- **Implement changes to project practices** in a manner that fits the project context **to achieve progress on a core value or principle.**
  - Use time boxes to create features
  - Specific techniques to iteratively refine features.
  - Consider dividing up one large project into several releases

➢ The **end goal** is not to be agile for its own sake, but rather to deliver a **continuous flow of value to customers** and achieve better business outcomes.

Source: Agile Practice guide, PMI






# Delivery Environments and Agile Suitability

# Delivery Environments

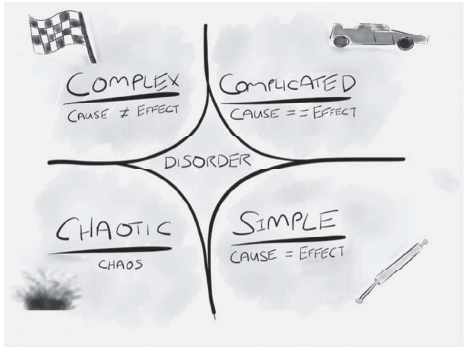
- The environment within which **Project/Product delivery** will occur should largely drive the delivery and **governance framework(s)** that will be implemented.
- For example, in a delivery environment where **high variability** is likely to be encountered (like IT product development), an **Agile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

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



# 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

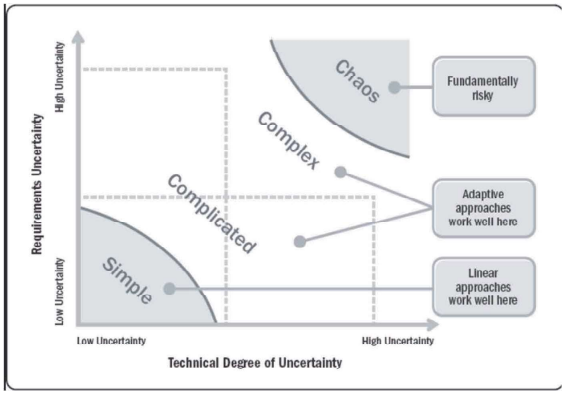



- The central idea of the framework is to offer decision-makers a "sense of place" to view their perceptions in dealing with a situation or problem. Not all situations are equal, and this framework helps to define which response is required for a given situation or problem.

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



# 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 Measey



# 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.

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



Thank you

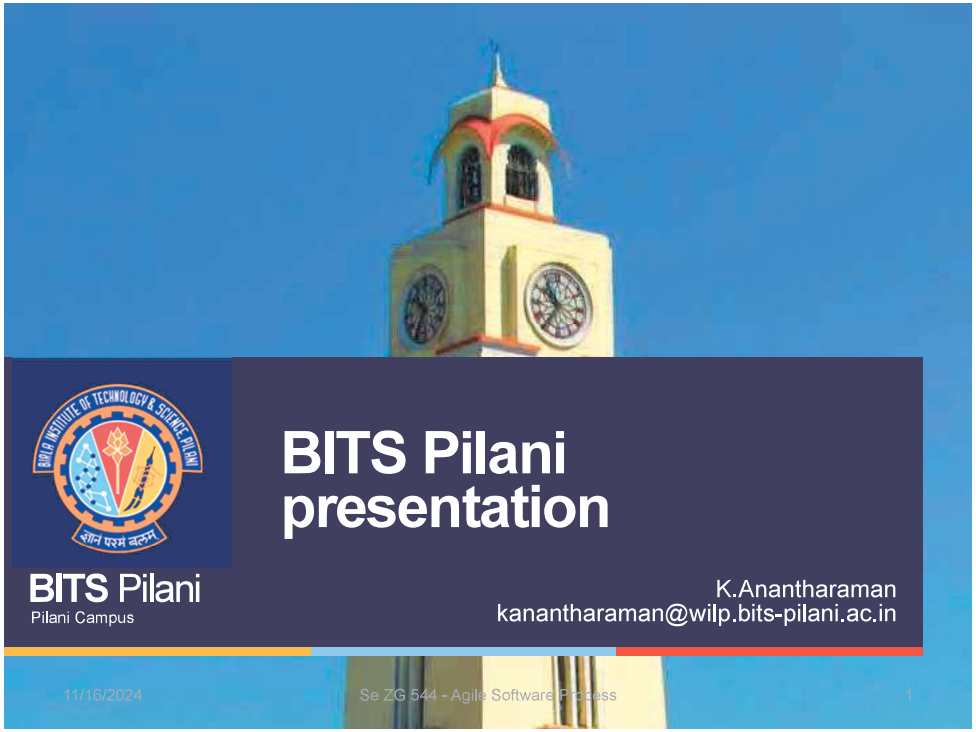


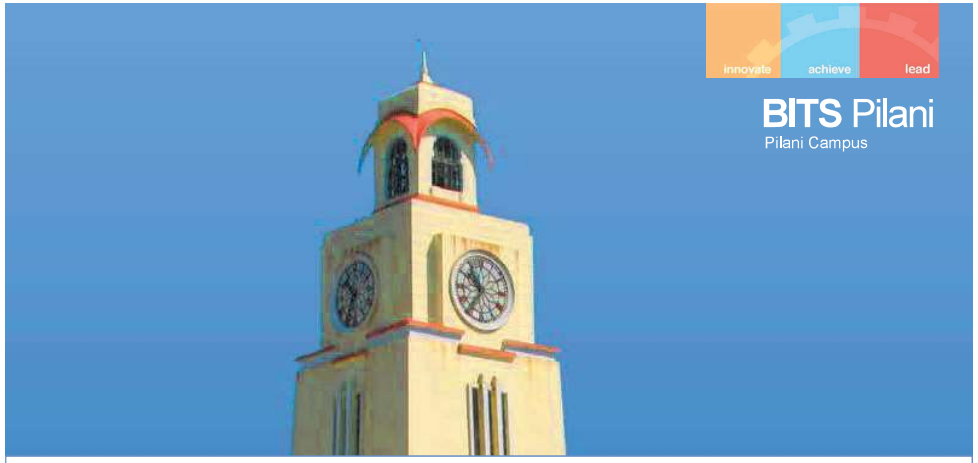
# A Note on Cynefin identifies five domains:

- 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/








# SE ZG544 – Agile Software Process CS3 – Agile Manifesto & Agile Principles

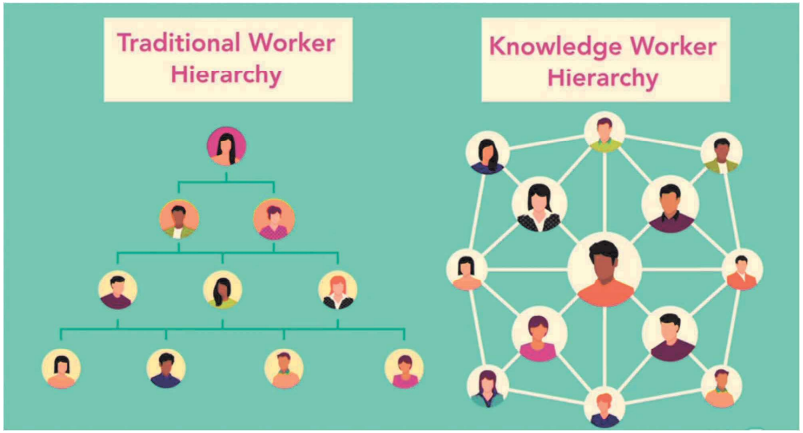


## CS3-Topics

- The rise of knowledge workers
- Agile Manifesto
- Agile Principles
- Team Motivation, Team Dynamics, Soft Skills
- Self Organizing teams, Emergent Design
- Simplicity



## The rise of knowledge workers

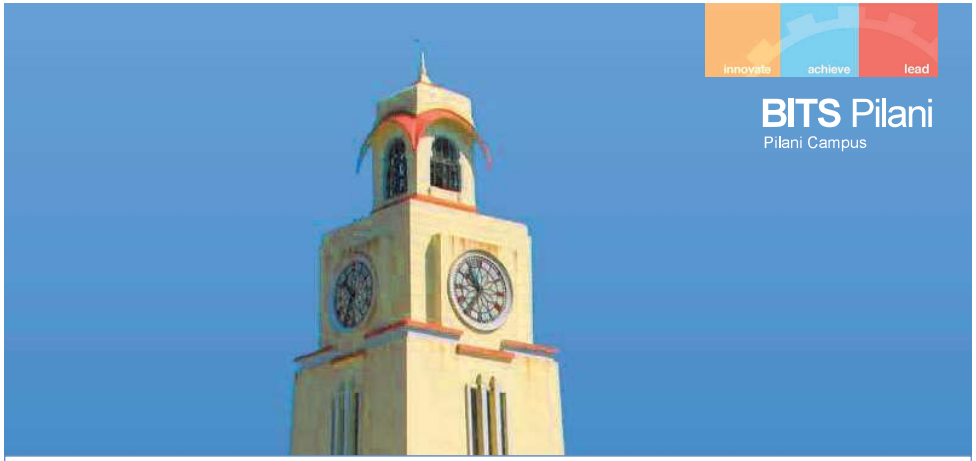



Directive leadership style

Supportive leadership (Servant leadership style)

Source: lynda.com agile-foundations by Doug Rose






# Agile Manifesto

# The Key Contributors

In Feb 2001, 17 new **methodology pioneers** met in Snowbird, Utah, USA.

To share their experiences, ideas, and practices and to suggest ways to improve the world of software development.

After Many discussions, they came up with Agile Manifesto


Image Source: https://udayanbanerjee.wordpress.com/category/agile



# Three Perspectives
## (HOT Perspectives)

- The **H**uman perspective:
  - Cognitive and social aspects, and refers to learning and interpersonal (teammates, customers, management) Process.
- The **O**rganizational perspective:
  - Managerial and cultural aspects and refers to the workspace and issues that extend beyond the team.
- The **T**echnological perspective:
  - Practical and technical aspects and refers to Technical and Coding Practices.

Ref: Hazzan Dubinsky2008_Book_AgileSoftwareEngineering



# Agile Manifesto

**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
**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 | |

*Source:* © 2001 www.agilemanifesto.org

- **Over the years, the Agile Manifesto has become a battle cry for organizational transformation.**



# Agile Manifesto Principles (Year 2001)

- https://agilemanifesto.org/

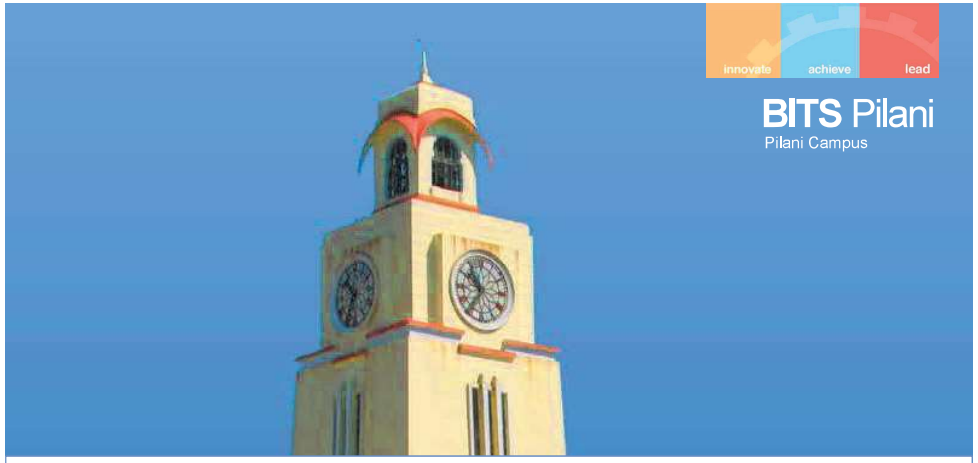
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over Process and tools (H)
- Working software over comprehensive documentation (T)
- Customer collaboration over contract negotiation (HO)
- Responding to change over following a plan (OT)

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

- Each principle supports and supported by other principles
- Redefined roles for Developer, Manager, Customer
- No "Big Upfront" Steps
- Iterative Development
- Negotiated and limited functionality
- Focus on Quality – Achieved through testing

Ref: Meyer2014_Book_Agile




# Agile Principles



# Augmentation of Agile Manifesto

- Agile Manifesto was not actionable
- To support teams making Agile transitions, actionable items were needed.
- The original signatories of Agile Manifesto augmented the four values with 12 Agile principles behind the Agile Manifesto

- **Differentiate between Value, Principle, Practice**
- **4 Agile values ,12 Agile Principles and many Agile Practices**



# 12 Agile Principles(Principles behind Agile Manifesto)

| Satisfy Customer | Embrace Change | Frequent Delivery |
|---|---|---|
| 01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. | 02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. | 03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. |
| **Customer collaboration** | **Support & Trust** | **F2F Communication** |
| 04 Business people and developers must work together daily throughout the project. | 05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. | 06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. |
| **Working software** | **Sustainable Phase** | **Technical Excellence** |
| 07 Working software is the primary measure of progress. | 08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. | 09 Continuous attention to technical excellence and good design enhances agility. |
| **Keep it simple** | **Self Organization** | **Inspect & Adapt** |
| 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. |

https://nicocasel.net/blog/why-i-feel-natural-with-agile-principles



# Agile Principles (Not Official)

**Organizational**

1 Put the customer at the center.

2 Let the team self-organize.

3 Work at a sustainable pace.

4 Develop minimal software:

- 4.1 Produce minimal functionality.
- 4.2 Produce only the product requested.
- 4.3 Develop only code and tests.

5 Accept Change

**Technical**

6 Develop iteratively:

- 6.1 Produce frequent working iterations.
- 6.2 Freeze requirements during iterations.

7 Treat tests as a key resource:

- 7.1 Do not start any new development until all tests pass.
- 7.2 Test first.

8 Express requirements through user stories or scenarios.

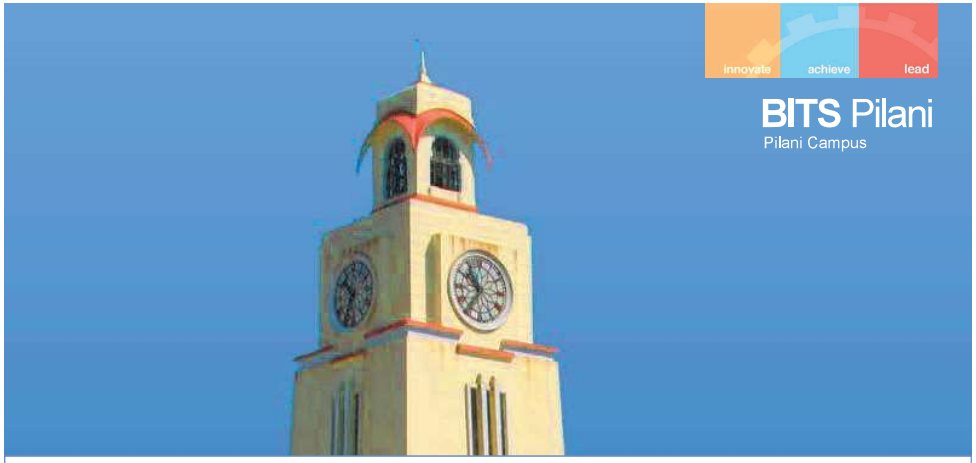
Ref: Meyer2014_Book_Agile

# Agile Practices

- Agile Manifesto → Agile Principles → Agile Practices
- Agile Practices → Project Outcome

- Agile Practices
  - 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
- Agile methods/methodologies
  - Scrum, XP, Kanban, Crystal






# Principle 1



# Quiz

Q1



# Agile Principle#1

**"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."**

- Releasing software early
  - Shipping a working version of software as early as possible. by choosing the features and requirements that will deliver the **most value.** This is the best way to get customer feedback.
- Delivering value continuously
  - The team that truly collaborates with customer has option of making necessary changes along the way. That's what continuous delivery means. --- (vs CCB )
- Satisfying the customer.
  - Plan short iteration, Deliver highest value, Early feedback, Incorporate feedback in next iteration. Collaborate with customer.

Source: The Agile principles By Andrew Stellman and Jennifer Greene

# Agile Principle#2

**"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."**

- Requirements changes due to emerging/new opportunities
- Respond to change instead of tight alignment to plans
- Change satisfies the customer's latest needs and provide the customer with a competitive advantage
- Agile teams embrace change by treating project changes as positive and healthy developments for the project
  - Nobody gets in "trouble" when there is change, We don't sit on the change until it is too late, We're all together.



# Agile Principle#4

**"Business people and developers must work together daily throughout the project."**

- **Business people** is referred to a Product Owner, or anyone who is a proxy between customer and team.
- Emphasize here to have a shared responsibility and accountability, 'work together' stresses on total commitment on both sides.
- Catching misunderstandings early, **clarify requirements just-in-time** and to keep all team members 'on the same page' throughout the development helps in producing successful outcomes.



# Agile Principle#3

**"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."**

- By using time boxed iterations to *deliver working software frequently.*
- Shorter Iterations (~2 wks.) (Smaller releases means fewer chances of bugs , frequent feedback)
- Agile teams constantly adjust the project so that it delivers the most value to the customer
- *We no longer regard a release cycle of "a couple of months" as agile. The industry has evolved to daily or weekly releases.*

Source: https://www.plutora.com/blog/12-agile-principles



# Some list of customer satisfaction issues and Agile Approaches

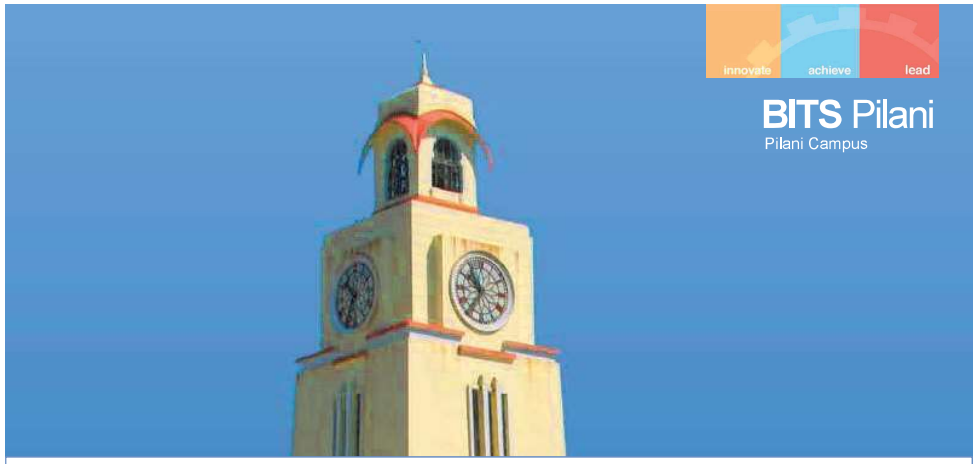
| Customer dissatisfaction issues | Agile Approaches |
|---|---|
| Product requirements misunderstood by the development team | The customer is able to provide feedback just-in-time and also at the end of the sprint, not before it's too late at the end of the project. |
| The product wasn't delivered when the customer needed it. | Working in sprints allows agile project teams to deliver high-priority functionality early and often. |
| The customer can't request changes without additional cost and time. | Agile teams can accommodate change in requirements, and shifting priorities with each sprint, by removing the lowest-priority requirements –offsetting cost. |

Source:T1

# Agile Principle#5

"Build projects around **motivated individuals.** Give them the environment and support they need, and trust them to get the job done"






# Motivated and Talented individuals

Source: Agile Foundations - Principles, practices and frameworks by Peter Measey
McGraw-Hill Education ACP Agile Certified Practitioner Exam by Klaus Nielsen



# Motivated individuals

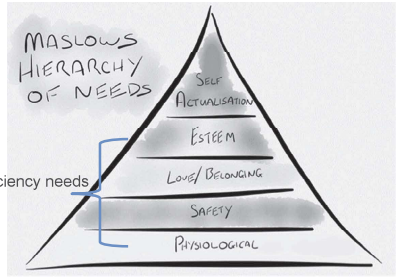
**5th Agile Principle states "Build projects around motivated individuals ….."**

- **Motivation** releases energy and creativity and is an essential component of **high performance.**
- We will look at a few **different approaches to understanding what motivates individuals** and the vital role that talent plays in achieving high performance.
- **We will look at psychological models**



# Maslow's hierarchy of needs (Motivation theory)

- Maslow's hierarchy of needs is a theory in psychology proposed by Abraham Maslow in his 1943 paper still relevant today in 2020.




- Maslow suggested that, once our basic needs are met, our behavior will be driven by meeting higher-level needs.
- He assumed that human beings have the natural propensity to move towards self-actualization (Fulfillment of one own full potential/talents) by satisfying preceding needs.

**Physiological** : Human survival requirements, such as food, water and air, etc., comprise physiological needs. The absence of these will create various psychological symptoms, such as hunger, thirst, discomfort.

**Safety**: Physical safety, economic security, employment, health and wellbeing, and protection against accidents/illness.

**Love/Belonging:** The sense of belonging and acceptance, for instance in working groups, congregations, professional bodies and sports teams, can foster creativity and motivation.

**Self Esteem/Respect** :Sense of contribution, achievement, recognition, freedom and attention.

**Self-actualization:** 'the desire to accomplish everything that one is capable of becoming'. Once individuals have achieved self-actualization they can provide their support to others.

# Management's attitude determines motivation

McGregor's Theory X and Theory Y

| Theory X managers believe that employees... | Theory Y managers believe that, given the right conditions, employees... |
|---|---|
| Hate work | Like and need work |
| Seek money and security | Seek to be involved and realise their potential |
| Have to be forced to work | Drive themselves and work effectively |
| Prefer to be told what to do | Take initiative |
| Are rarely creative | Are naturally highly creative |
| Are selfish | Commit themselves to larger goals |

- An Agile leadership style should be in alignment with McGregor's Theory Y, which views employees in a positive light.
- As in Agile, that puts individuals and teams first, McGregor's research outcomes prove that teams under Theory Y management showed better performance in comparison to Theory X teams.

- Whether employees display Theory X or Theory Y behavior is a consequence of how management treat them.
- This means that generally managers will get what they expect – if they expect and manage for Theory X behavior, they will typically get employees displaying Theory X behavior.



# Daniel Pink,2010 - Motivation comes from autonomy, mastery and purpose

**Autonomy –** people's desire to direct their own lives and to gain control over some (or all) of the four main aspects of work: what, how, when and with whom.

**Mastery –** becoming better at something that matters to an individual. This can be achieved by taking on tasks that allow people to develop skills further. Mastery is fostered by an environment where learning is encouraged and mistakes are tolerated.

**Purpose –** fulfilling a natural desire in people to contribute to a cause greater than themselves.



# Some factors only demotivate

- A researcher Herzberg proposed a refinement to Maslow's and an addition to McGregor's approach
- Hygiene factors comprise:
  - Pay, company policy, quality of supervision/management, working relations, working conditions, status and security.
- Motivators comprise:
  - Achievement, recognition, responsibility, advancement, learning, type and nature of work.

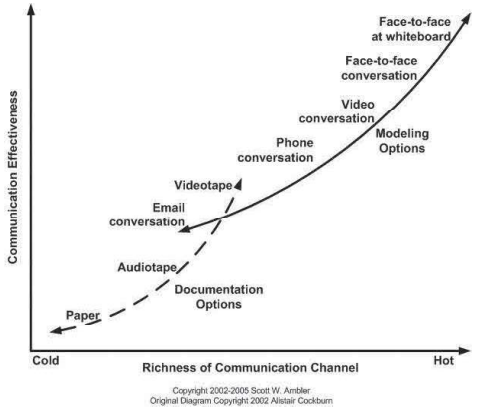


# Talent

- To achieve high performance, motivation needs to be coupled with **talent.**
  - So where does talent come from? Are people born with it or can they acquire it? Matthew Syed argues for the latter, and states that (Syed, 2011):
- Talent comes from purposeful practice
- Practice needs to be purposeful.(Master at something)
  - Not all practice is useful. People only develop when they repeatedly try things that are just out of reach and get quality feedback on their performance.
  - The paradox of excellence is that it is built on necessary failure
  - The learning process is often best facilitated by an expert coach.
- Engaging in purposeful practice leads to high performance – and the opposite is also true.

# Agile Principle#6

**"The most efficient and effective method of information to and within a development is face-to-face conversation."**




**Remote teams:**
- Skype and Hangouts allow us to have remote face-to-face talks.

- One-on-one communication has it challenges
- Number of communication channels: N × (N–1)/2.
- 7 member team = 7*(7-1)/2=21



# Agile Principle#8

**"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."**

- Agile work is intense.
- Regular weekends, night outs, overtime not sustainable
- The entire team is responsible for maintaining sustainable phase
  - Business owners or Product owners
  - The development team
  - Team Lead/Scrum master
  - Organization/Sponsors

Source: https://soulofscrum.com/blog/f/keeping-a-sustainable-pace



# Agile Principle#7

**"Working software is the primary measure of progress."**

- Working software is better than progress reports, because it's the most effective way for the team to communicate what they've accomplished.

- Finished analysis, complete models, or beautiful mock-ups have may be necessary, But have little meaning if they aren't converted into working software.



# 8. Sustainable Phase …

Product Owners (and other interested parties outside the Scrum Team):

- Don't make commitments to the business that don't come from the Development team.
- Don't expect for the Development team to commit to anything longer term than the upcoming Sprint.
- All prioritization comes through the Product Owner…respect that.
- Keep in mind the cone of uncertainty when developing your Product Road Map.

https://soulofscrum.com/blog/f/keeping-a-sustainable-pace

# 8. Sustainable Phase …

Development Team:

- Beware of estimating what you can get done in a sprint
- By taking on too many backlog items into the Sprint Backlog you endanger having the time for creative solutions.
- Cross training skills between team members
- Worst of all the temptation of accepting lower quality for the Product Increment.
- Collaborate with Product owner and prioritize the work

Source: https://soulofscrum.com/blog/f/keeping-a-sustainable-pace



# Agile Principle#9

**"Continuous attention to technical excellence and good design enhances agility."**

- Object Oriented design, Design patterns, Decoupled service-oriented architectures, Containers, Cloud technology and other innovations an tools bring technical excellence to product.
- Well designed code is easy to maintain and extend.
- Constantly lookout for design and code problems and take time to fix those problems.
- Use TDD and pay back Technical Debt



# 8. Sustainable Phase …

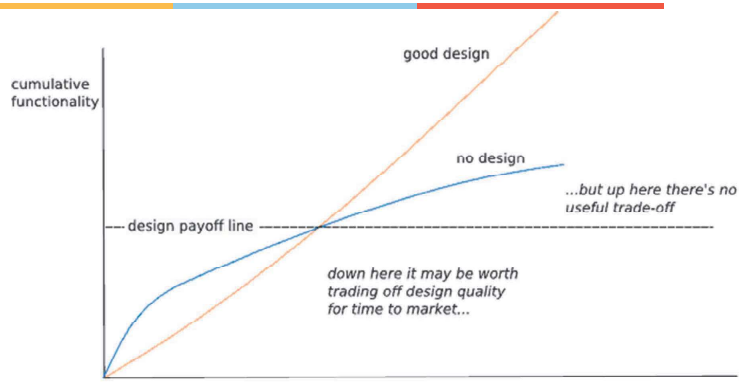
Scrum Masters:

- If you see the team overextending, ask them about this in the Retrospective.
- Make sure there are no external pressures on the team.
- Habitual or frequent overtime indicates an issue that you should investigate and remove as an impediment to team health.

Source: https://soulofscrum.com/blog/f/keeping-a-sustainable-pace



# Technical Excellence

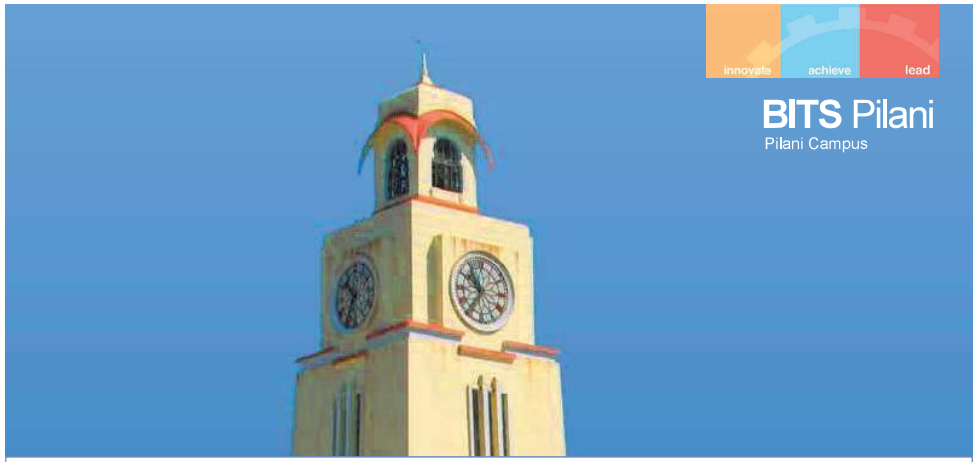



**Technical debt** is a term first coined by Ward Cunningham. It describes the **accumulation of poor design** that crops up in code when decisions have been made to implement something quickly. Ward described it as Technical Debt because if you don't pay it back in time, it starts to accumulate. As it accumulates, subsequent changes to the software get harder and harder. What should be a simple change suddenly becomes a major refactor/rewrite to implement.

Martin Fowler created a pseudo-graph to visualize this:

# Q2






# Simplicity

## The art of maximizing the amount of work not done - 10th Agile Principle



# Simplicity in Agile context

This means:

1. Focusing on ensuring that only the **simplest, leanest and fit-for-purpose product** is delivered, especially when considering lifecycle-driven documentation, and only producing what adds value.

2. Focusing on **maximizing the amount of work not done** when creating the product, i.e., focusing on **simplicity of delivery**



# Fit-For-Purpose product

- This principle is therefore about reducing clutter and keeping the backlog focused on whatever needs to be delivered first
- Breakdown of features that are actually used in a typical delivered system (Standish Groups 2002).
  - **Features always used – 7%** → **Deliver these features first**
  - Features often used – 13%
  - Features sometimes used – 16%
  - Features rarely used – 19%
  - Features never used – 45%
- Agile frameworks have the concept of producing technically fit-for-purpose products.

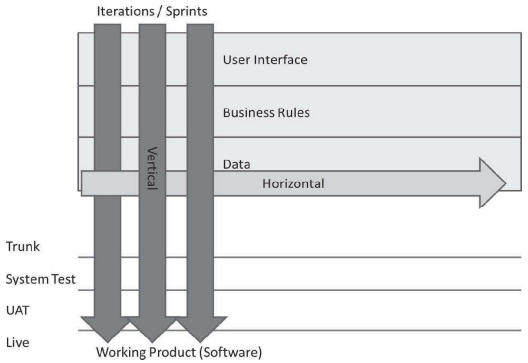
# Fit-For-Purpose Delivery

- This essentially means following Lean Software Development:
  - Eliminate Waste, Build in Quality, Create knowledge, Defer commitment, Deliver fast, Respect people, Optimize the whole
- For example,
- Eliminate waste:
  - Extra stories, stories constantly changing and the buffers created by crossing organization boundaries.
- Build in quality
  - If defects are routinely found in the verification process, the development process is defective.
- Defer commitment :
  - Abolish the idea that development should start with a complete specification
- Optimize the whole:
  - Viewed across the whole value chain - Brilliant products emerge



# Vertical slices

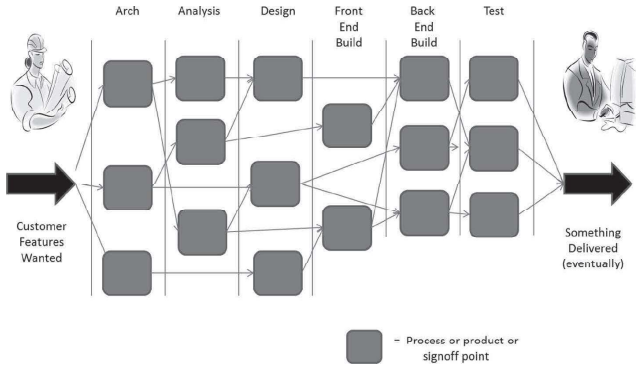
- In an Agile delivery teams focus on producing the highest value stories in vertical slices down the architecture.
- Water fall thinking results in Horizontal slicing






# Silo Delivery

- In a team working with a silo mentality , these unseen boundaries can easily become barriers to effective communication and delivery



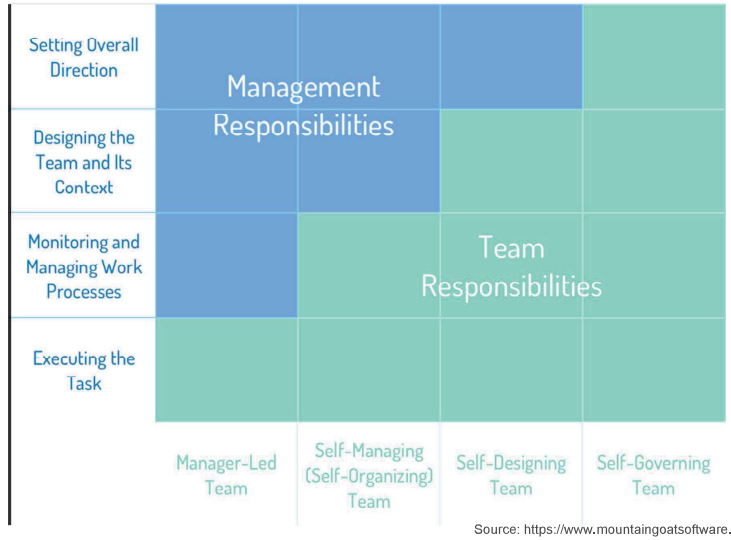



# Agile Principle#11

"The Best Architectures, Requirements, and Designs Emerge from **Self-Organizing Teams.**"


Source: https://www.mountaingoatsoftware.com

# Type of teams




Source: https://www.mountaingoatsoftware.com



# 'Emergent design' – why is it important?

- BDUF: Big Design Upfront
- EDUF: Enough Design Up-front (**Agile team implement this Approach**)
- Upfront design - restrict opportunities to change and improve the design as the product is being developed.
- Wait until the **last responsible moment** to make design decisions
  - You can make decisions based on evidence that is identified as the system is being built. **This typically means that decisions are of a higher quality, because they are not just theoretical decisions based on little evidence.**
  - Example: Ordering a Hardware before lead time. **The last responsible moment is governed by that lead time**, and we have to find out enough about the hardware to make the order by that time, do some experimentations, gather required knowledge etc.



# Self Organizing teams

- Not every agile team will choose to organize themselves the same way
  - Some teams will decide that all key technical decisions will be made by one person on the team.
  - Other teams will decide to split the responsibility for technical decisions along technical boundaries
  - Still other teams may decide that whoever is working on the feature makes the decision but has the responsibility of sharing the results of the decision with the team.
- Making use of the **collective wisdom of the team** will generally lead to a better way of organizing around the work than will relying solely on the wisdom of one personnel manager.

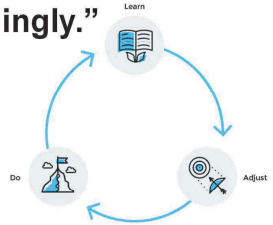


# Self-organizing teams and emergent design

- Self-organizing teams are empowered, within agreed boundaries, to deliver **fit-for-purpose products, in a fit-for-purpose way, within the most effective time scale.**
- Self-organizing team in relation to emergent design is that there will be some **overarching design principles that teams must or should align to the timescale.**
- If teams are **forced to align to an externally defined detailed design** they are unlikely to 'go the extra mile' to try and identify or implement any opportunities to make the design better (**opportunistic design**).
  - It is likely that the only people who can effectively make the right detailed design decisions are team members. Nobody else will understand the evolving design as well as the team does.
  - This ties in with the concept of '**real options**' (Matts, 2007), which means keeping your options open for as long as you possibly can and making a decision when you are in the best position to make it with confidence.

# Agile Principle#12

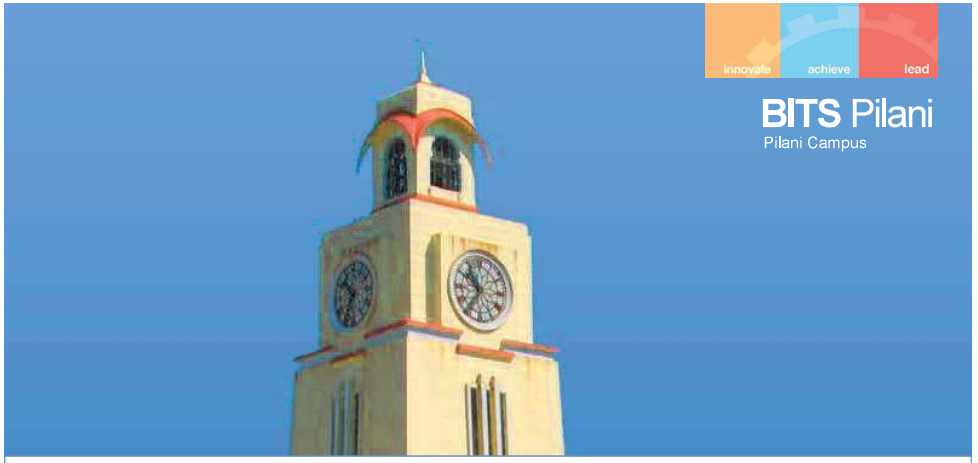
**"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."**




- At regular intervals, self-organizing teams should take the time to look at the way they work and adjust accordingly. No team runs perfectly.
- A mature agile team can identify issues with respect for each other and then take action to improve the process.

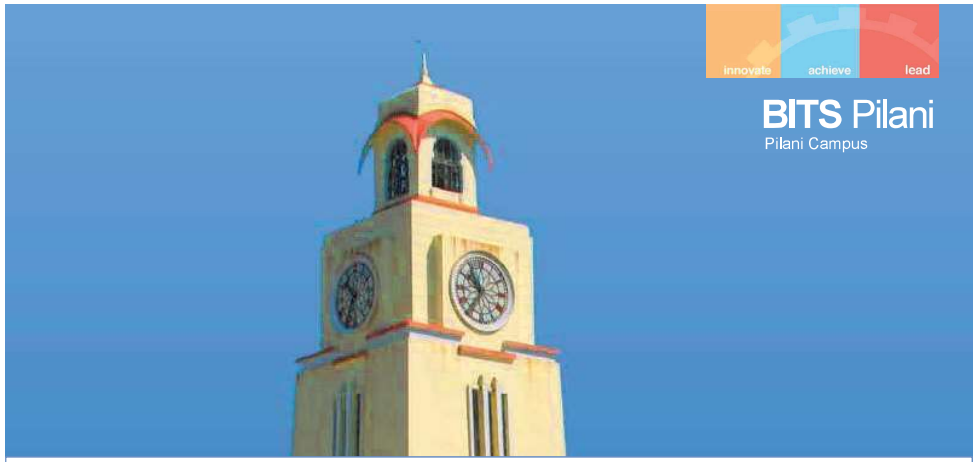
https://www.plutora.com/blog/12-agile-principles






# Team Dynamics and Interpersonal skills

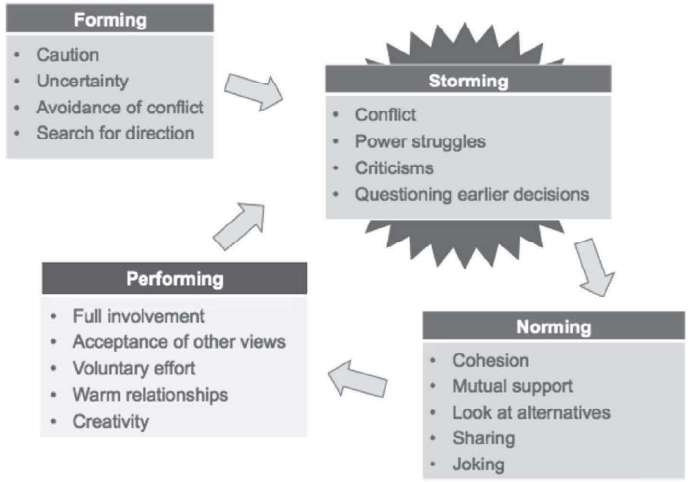





# Emergent Design From Self-Organizing Teams



# Tuckman's theory of team evolution



# Lencioni – the five dysfunctions of teams


Dysfunction team


Functional high performance team



# Emotional Intelligence

- Emotional intelligence, in an Agile team, provides the team with the tools to **make things work and the ability to perform well**.

Mixed Model by Daniel Goleman

| | *Self* | *Other* |
|---|---|---|
| | ***Personal Competence*** | ***Social Competence*** |
| ***Recognition*** | ***Self-Awareness*** | ***Social Awareness*** |
| | Emotional self-awareness | Empathy |
| | Accurate self-awareness | Service-oriented |
| | Self-confidence | Organizational awareness |
| ***Regulation*** | ***Self-Management*** | ***Relationship Management*** |
| | Self-control | Developing others |
| | Trustworthiness | Influence |
| | Conscientiousness | Communication |
| | Adaptability | Conflict management |
| | Achievement-driven | Leadership |
| | Initiative | Change catalyst |
| | | Building bonds |
| | | Teamwork and collaboration |

- The higher the emotional intelligence of the Agile team, the greater are the chances for being successful in a people-oriented environment.



# Interpersonal Skills – Key terms

- **Adaptive leadership** A type of leadership that deals with changes and problem solving
- **Collaboration** The action of working with someone to produce something
- **Conflict resolution** Method(s) to solve conflicts
- **Emotional intelligence** Focused on people and forging strong and supportive relationships
- **Negotiation** Process of reaching the best results
- **Servant leadership** A philosophy and set of practices that enriches the lives of individuals, builds better organizations and, ultimately, creates a more just and caring world



# The Emotional Intelligence Skills Assessment (EISA) framework

- Provides a strong fundamental assessment of emotional intelligence in project managers and Agile team members

| *Factor* | *Comments* |
|---|---|
| ***Perceiving*** | The ability to recognize, acknowledge, and attend to the emotions of one's own self and other team members |
| ***Managing*** | The ability to express emotions in a controlled manner |
| ***Decision making*** | The ability to apply emotions effectively in decision making |
| ***Achieving*** | The ability to generate the emotions that will motivate oneself toward the pursuit of a desired goal |
| ***Influencing*** | The ability to motivate others in the pursuit of a goal, by evoking similar emotions in others as well |

# Collaboration

- Why do we collaborate and what factors contribute to effective collaborations?
  - **We collaborate in order to deliver software (deliverables)**
  - **To foster progress by making decisions (decisions)**
  - **To learn (knowledge).**
- Cockburn and Highsmith (2002) identified collaboration as a combination :
  - **Interpersonal (trust, participation, commitment, respect)**
  - **Cultural (values and principles)**
  - **Structural (organization, technology, and practices) values.**
- Agile practices encourage collaboration and coordination through:
  - Daily stand-up meetings, Daily interaction with the product team, Stakeholder coordination



# Negotiation

Fisher, Ury, and Patton (1991) call their approach "principled negotiation." Their book, *Getting to Yes*, contains four key elements:

- Separate people from the problem
- Focus on interests, not positions
- Invent options for mutual benefit
- Use objective criteria



# Adaptive Leadership

- Adaptive leadership that deals with changes and problem-solving.
- Adaptive leadership focuses on team management from building self-organized teams for developing servant leadership style.
- Adaptive work, such as that embodied by Agile, requires adaptive leadership. Different situations call for different responses, there is a technical or routine response, and there is also an adaptive response

| *Situation* | *Technical or Routine* | *Adaptive* |
|---|---|---|
| *Direction* | Define problems and provide solutions | Identify the adaptive challenge and frame key questions and issues. |
| *Protection* | Shield the organization from external threats | Let the organization feel external pressures within a range it can stand. |
| *Orientation* | Clarity roles and responsibilities | Challenge current roles and resist pressure to define new roles quickly. |
| *Managing Conflict* | Restore order | Expose conflict or let it emerge. |
| *Shaping Norms* | Maintain norms | Challenge unproductive norms. |



# Conflict resolution

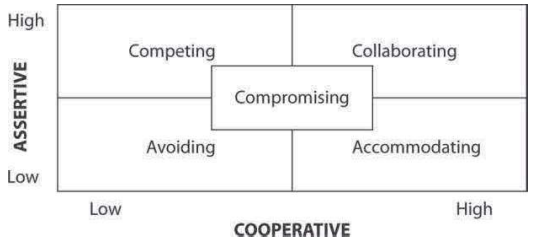
" The steps in conflict management may involve the following activities:

- Conflict identification
- Conflict analysis (who, what, why, when)
- Conflict resolution

# Conflict Resolution Techniques

The Thomas-Kilmann conflict mode and the five general techniques for solving conflicts are fairly similar. His five general conflict resolution techniques are:

- Withdraw/avoid
- Smooth/accommodate
- Compromise/reconcile
- Force/direct (compete)
- Collaborate/problem-solve






# Conflict resolution skills

| | |
|---|---|
| ***The win–win approach*** | How can we solve this as partners rather than opponents? |
| ***Creative response*** | Transform problems into creative opportunities. |
| ***Empathy*** | Develop communication tools to build rapport. Use listening to clarify understanding. |
| ***Appropriate assertiveness*** | Apply strategies to attack the problem, not the person. |
| ***Cooperative power*** | Eliminate "power over" to build "power with" others. |
| ***Managing emotions*** | Express fear, anger, hurt, and frustration wisely to effect change. |
| ***Willingness to resolve*** | Name personal issues that cloud the picture. |
| ***Mapping the conflict*** | Define the issues needed to chart common needs and concerns. |
| ***Development of options*** | Design creative solutions together. |
| ***Introduction to negotiation*** | Plan and apply effective strategies to reach agreement. |
| ***Introduction to mediation*** | Help conflicting parties to move toward solutions. |
| ***Broadening perspectives*** | Use the three articles on running meetings in conflict resolving mode. |

Conflict resolution an important factor of emotional intelligence, which more and more teams foster to improve the teamwork.
Communication, active listening, negotiation skills, and soft skills all play an important part in Agile methodologies, as teams are self-governed and empowered.



# Conflict resolution

The book, Coaching Agile Teams (2010) by Lyssa Adkins, is popular in Agile circles and contains the following

| | |
|---|---|
| ***Level 5*** | "World War" is when talks stop and the parties simply seek to destroy each other. We should do whatever is necessary to solve this conflict before someone gets badly hurt. Desperate measures may be needed and often outside help is employed. |
| ***Level 4*** | "Crusade" is really bad. Protecting one's own group becomes the focus and language is ideological. It is us or them. Winning is the only option. A safe environment must be created to allow time and, hopefully, some shuttle diplomacy to figure out a solution to the conflict. |
| ***Level 3*** | Things may turn worse with a "Contest" conflict where the winning trump resolves the issue. Language may include personal attacks. In this situation we need to accommodate and accept the other party's demands and, hopefully, end the conflict with a negotiation. |
| ***Level 2*** | Conflicts may worsen or start at "Disagreement" where personal protection trumps collaboration and language is guarded and open to interpretation. The situation may be handled by giving the participants needed support and ensuring the empowerment of the participants to find a good solution without fearing for their safety. We need a cool, calm environment. |
| ***Level 1*** | The lowest level of conflict is "Problem to Solve," which we all know. Information sharing and collaboration takes place as language is open and fact based. The way to deal with these kinds of regular conflicts is with collaboration by seeking consensus and the win–win situation. |

| Conflict level | Successful response options |
|---|---|
| Level 1: Problem to Solve | Collaboration. Seeking a win-win situation.<br>Consensus. Learning where every team member's head is with regard to the issue and, in time, arriving at a decision everyone can back. |
| Level 2: Disagreement | Support. Empowering the other to resolve the problem.<br>Safety. Anything that restores a sense of safety, such as collaboration games or regrounding in the team's shared values. |
| Level 3: Contest | Accommodate. Yielding to the other's view when the relationship is more important than the issue. This is a successful short-term strategy only and becomes a liability if used often over the long term.<br>Negotiate. When the "thing" the conflict is about is divisible, such as the use of a shared resource, negotiation can work. Negotiation will not work when the issue revolves around people's values. Values are not divisible, and one person giving in to another in violation of their own values feels like a sellout.<br>Get factual. Gather data about the situation to establish the facts. |
| Level 4: Crusade | Establish safe structures again. Use "shuttle" diplomacy, carrying thoughts from one group to the other until they are able to de-escalate and use the tools available at lower levels of conflict. |
| Level 5: World War | Do whatever is necessary to prevent people from hurting one another. |



# Servant Leadership

The core characteristics of being a servant leader are:

Listening

- Empathy
- Healing
- Awareness
- Persuasion
- Conceptualization
- Foresight
- Stewardship
- Commitment to the growth of people
- Building community

The emphasis here is on four factors
- Get the right people into the team.
- Trust team members rather than requiring them to prove themselves trustworthy
- Let the team select the project approach for project success
- Stand back and let the team do their work.

# Anti Patterns: Agile Manifesto

- The tool makes us Agile, Relentless automation
- Hierarchies
- Over-standardization
- Proxy customers (Business Analysts, Architect acting as customer)
- Considering plans and roadmaps as commitments
- Expecting too much detail
- Not engaging stakeholders



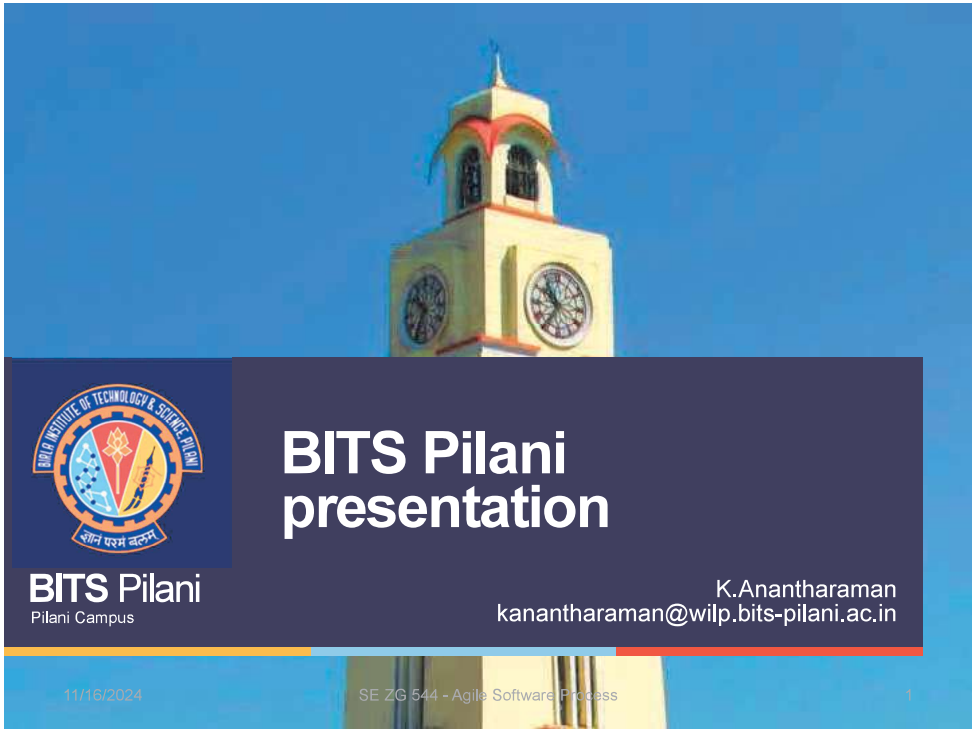
# Anti Patterns: Agile Principles

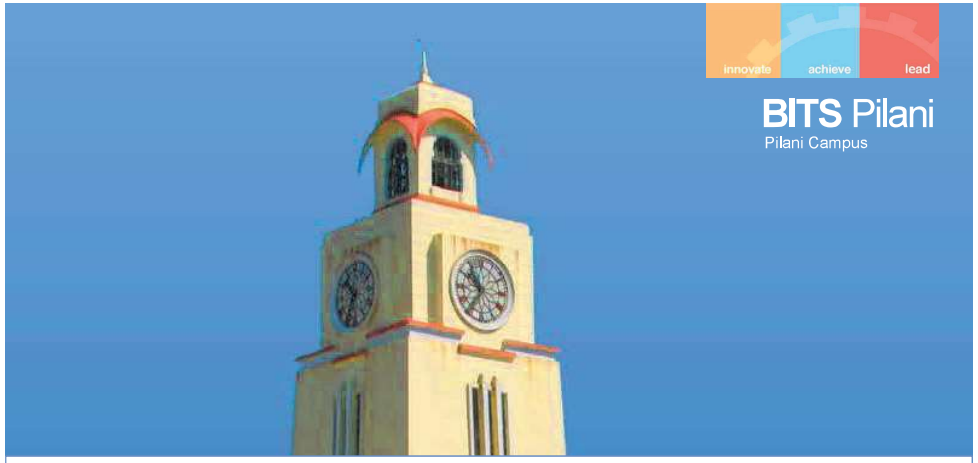
- Out of sight, out of mind - Stakeholders
- Requiring additional documentation or reporting, "We will need this later", Documentation as collaboration, Write only documentation
- One size fits all approach towards team management
- Chasing the metrics
- Ignoring the environment
- Multiple deployment environments
- Detailed story descriptions, Fixed standards or Process, Aiming for Small stories on the backlog
- Restricting who can talk to the customer
- Not considering cultural differences
- Lacking collaboration skills
- Over-complicating things/Future proof everything
- Insisting on Sign-off Process
- "Just in case" development
- Management focus on individuals
- Iterations planned in advance
- Focus on the tasks not the value



# Q3







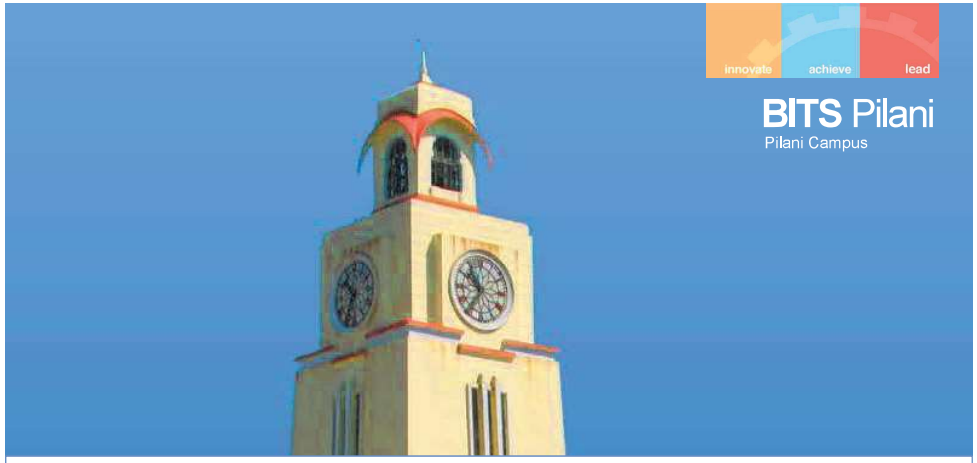

# SE ZG 544 , Agile Software Process Module-4 - Agile Methodologies



## Module-4 Topics

- Agile Methods
- Scrum
- XP
- Lean
- Kanban

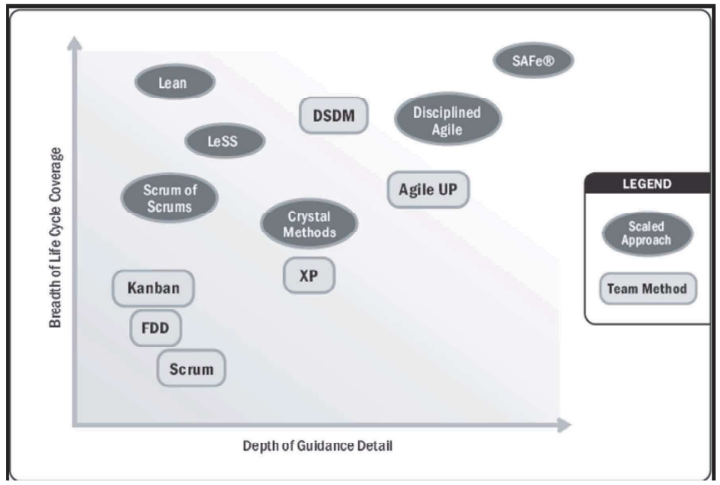




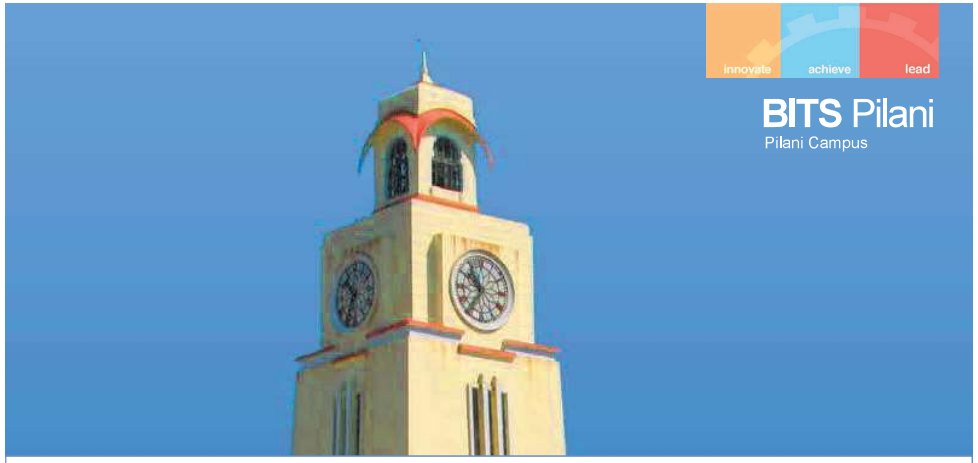

# Agile Methodologies Scrum, Kanban, XP, Lean



## Agile Methods




Source :Agile Practice Guide by Project Management Institute, published by Project Management Institute, 2017




# Scrum



# 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

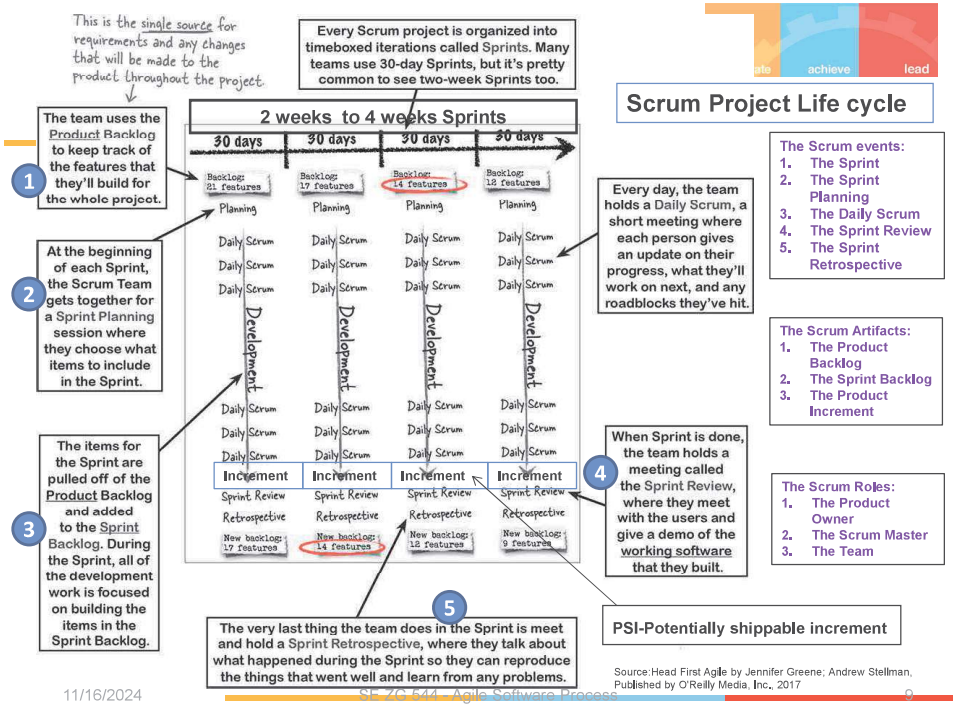
**Life Cycle and Process**

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

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

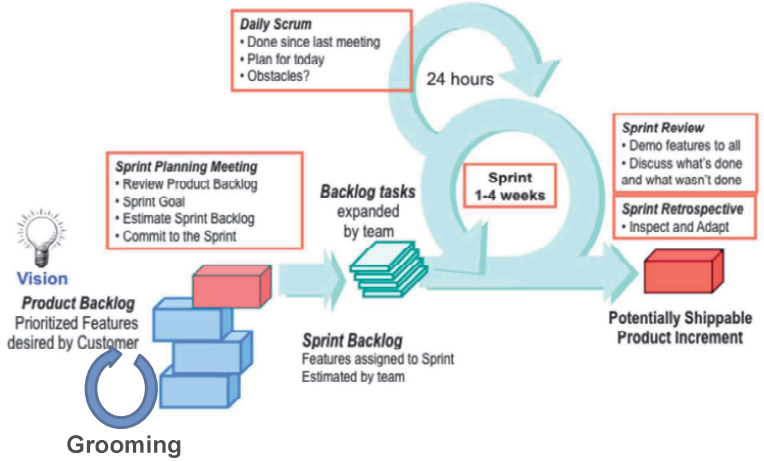


# Scrum Project Life cycle




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

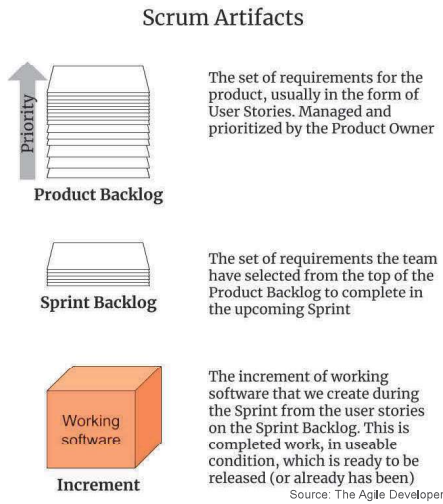
# Scrum Project Lifecycle




Agile project management with Scrum



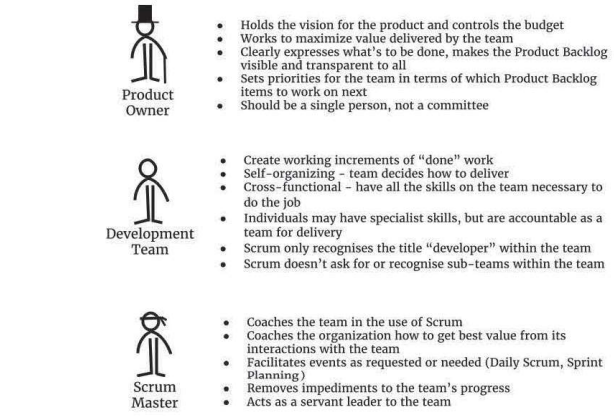
# Scrum Artifacts




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



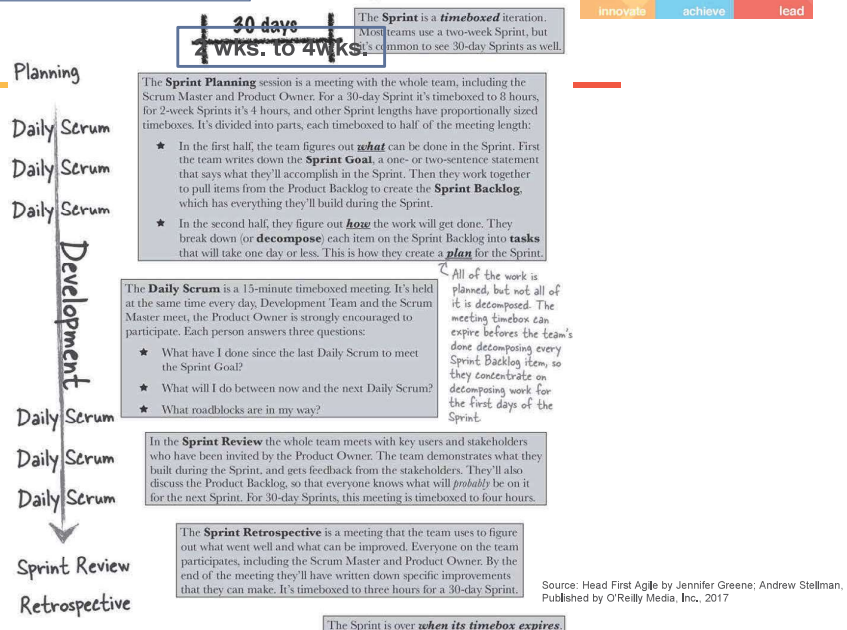
# Scrum Roles




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



# Sprint Events/Ceremonies

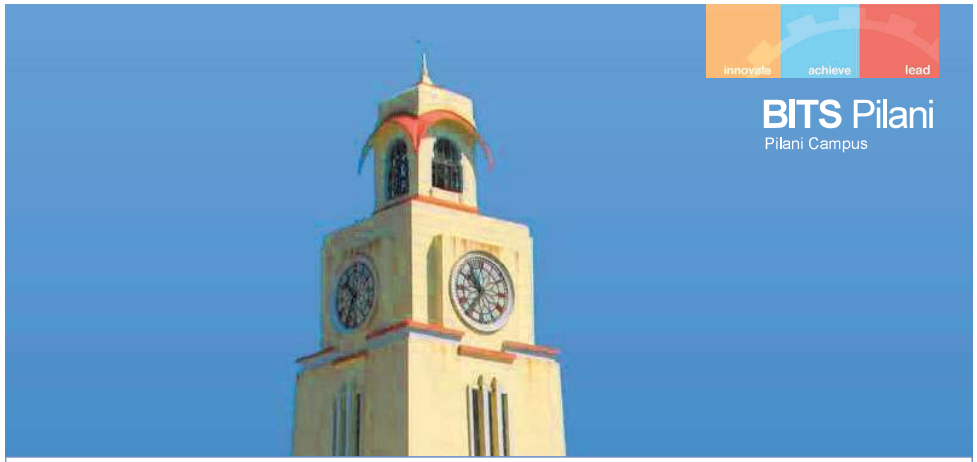



Source: Head First Agile by Jennifer Greene; Andrew Stellman, Published by O'Reilly Media, Inc., 2017

# Quiz

Set1





# XP- Extreme Programming

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

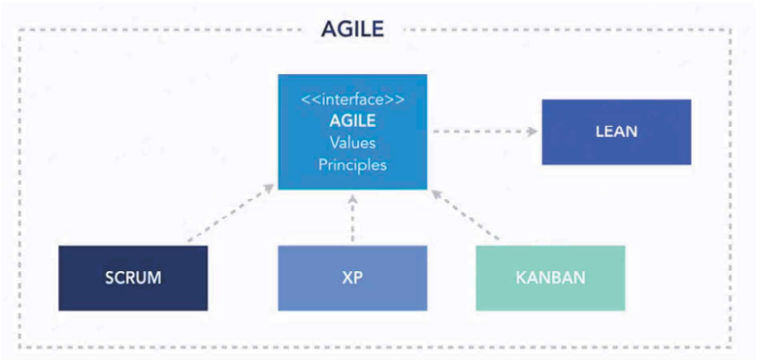


# What is eXtreme Programming?

- 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 :Lynda.com/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

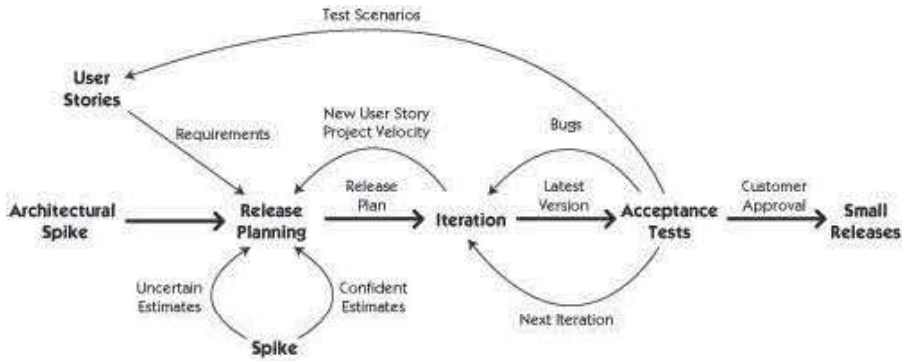


# XP Core Values

- **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.



# A visual process model for XP






# 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



# 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.



# 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



# 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 …

- 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.

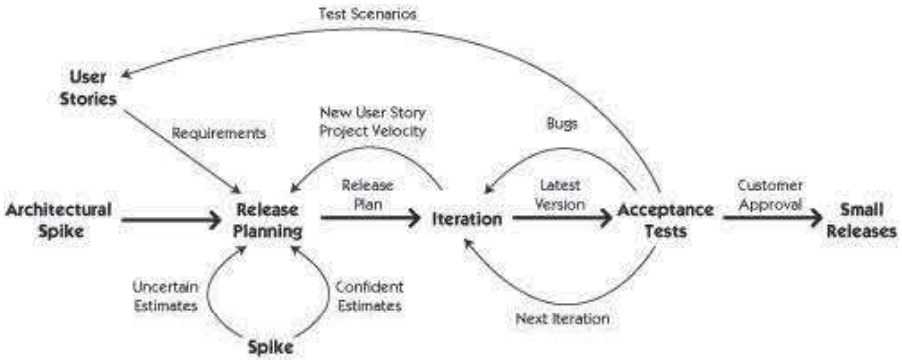


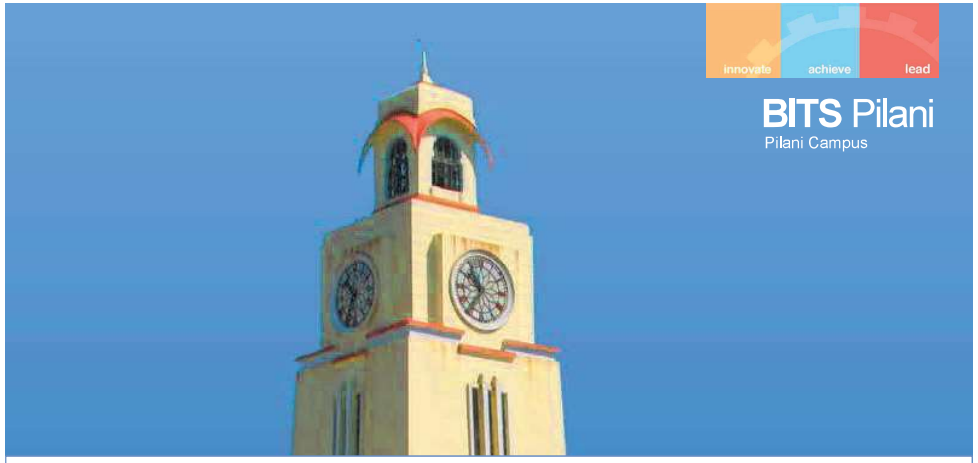
# XP Practices …

- 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



# A visual process model for XP

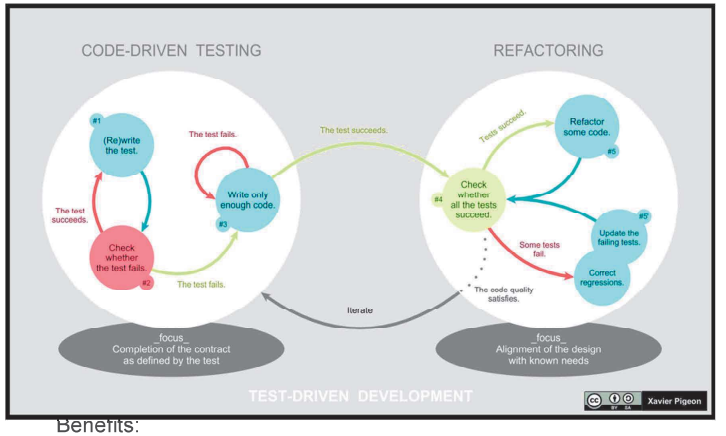





# Test Driven Development (TDD)



# Test Driven Development (TDD) – Detail Process




Tools:

- Junit for Java
- Pyunit for Python

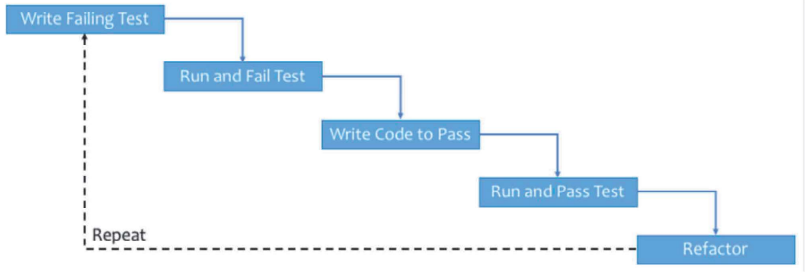
Benefits:

- Improves Code quality
- Shortens feedback loop
- Shorten Development time
- TDD is part of Agile Culture



# Test Driven Development- General work flow

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




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



# "Hello World" TDD Example

**File mytests.py**

```
import unit test from mycode import *
class MyFirstTests(unittest.TestCase):
def test_hello(self):
  self.assertEqual(hello_world(), 'hello world')
```

1.File mycode.py

```
def hello_world():
```

Run mytest.py

Test fails . No return value.

2. Then Add code to mycode.py

```
def hello_world():    return 'hello world'
```

Run mytest.py

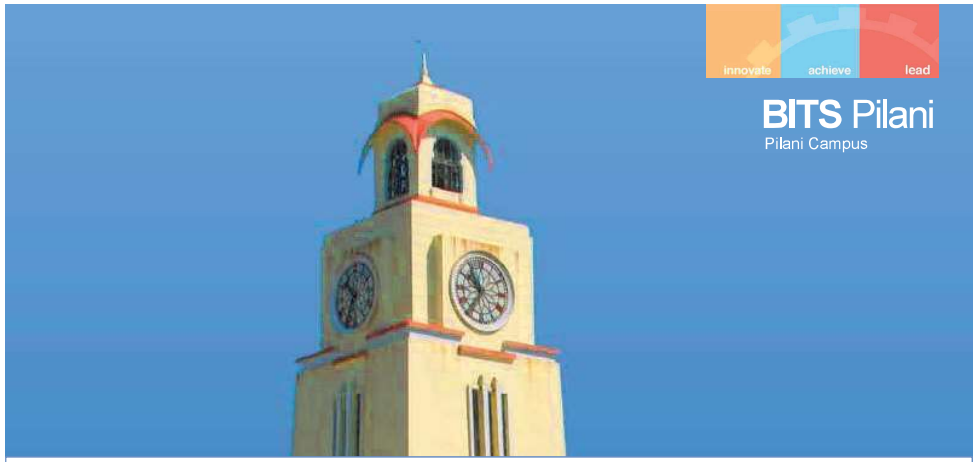
Test Pass.

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

# Quiz

S2






# Lean Software Development

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



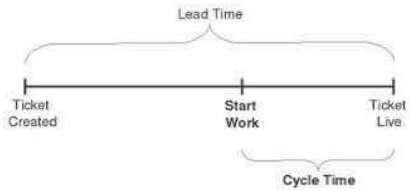
# What is Lean?

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



# Lead time & Cycle Time

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

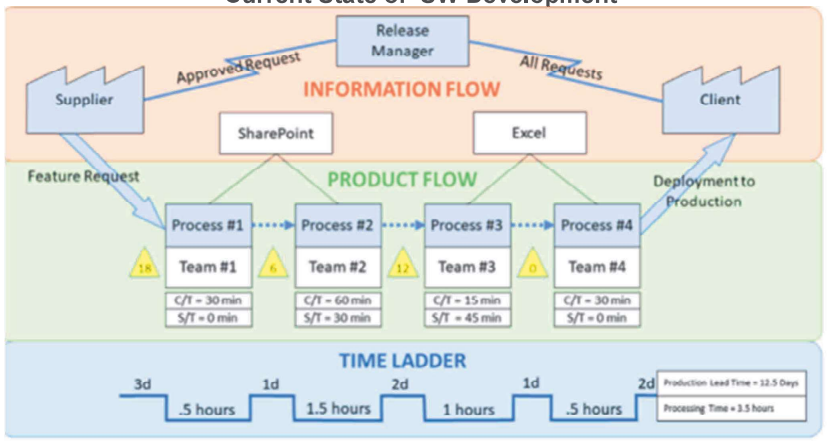


# Value Stream Mapping

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



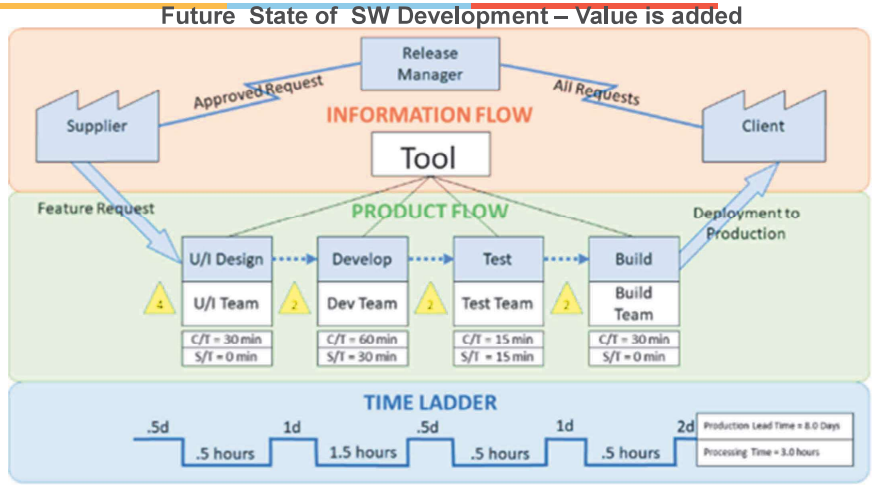
# Example-Value stream Mapping- Initial State




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

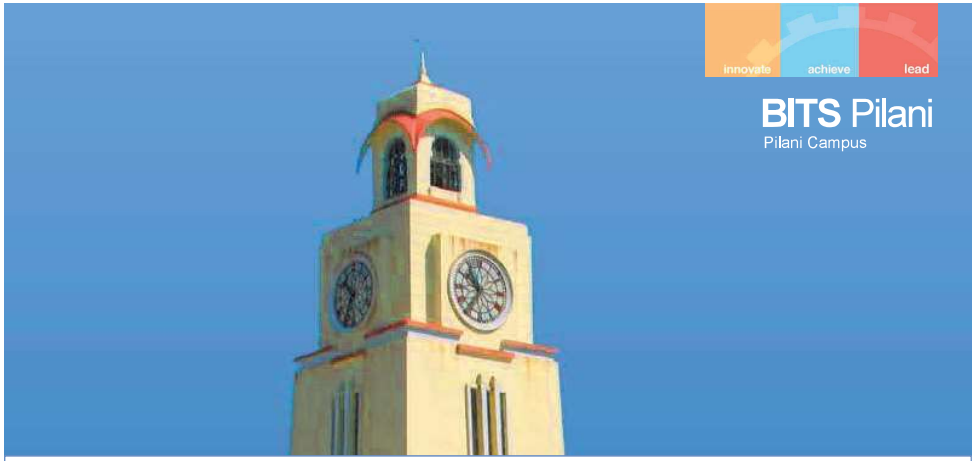


# Example-Value stream Mapping – Future state




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






# 7 Principles of Lean Software Development

# 1. Eliminate Waste

**Type of waste in SW Development**

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



# 2. Build Quality In

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



# 3. Create Knowledge

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



# 4. Defer Commitment

- Don't  make decision/commit if you can differ it at later point in time. Keep options open.
- Continuously collect  and analyze the data or information

## 5. Deliver Fast

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



## 6. Respect People

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



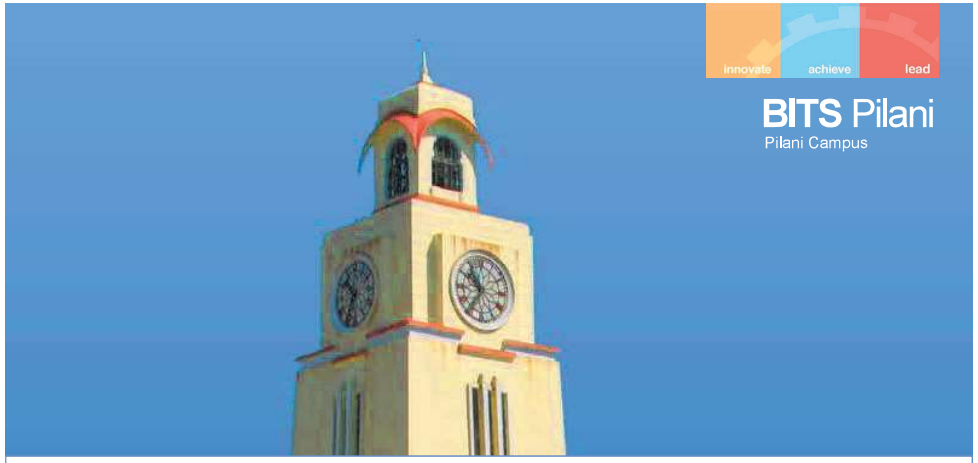
## 7.Optimize the Whole

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



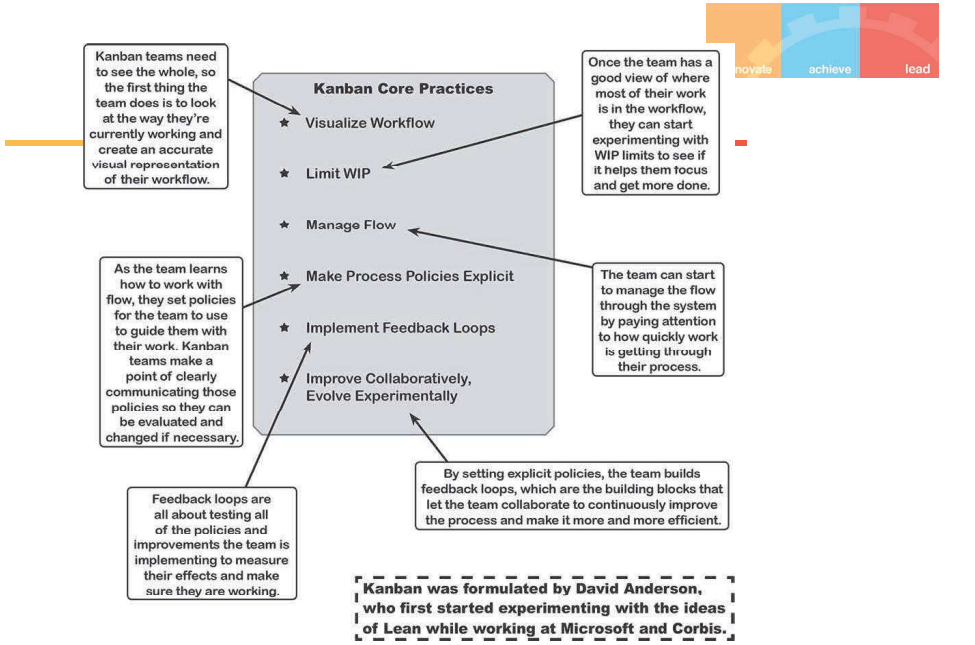
## Lean Principles that are Proven to Work

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




# Kanban






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

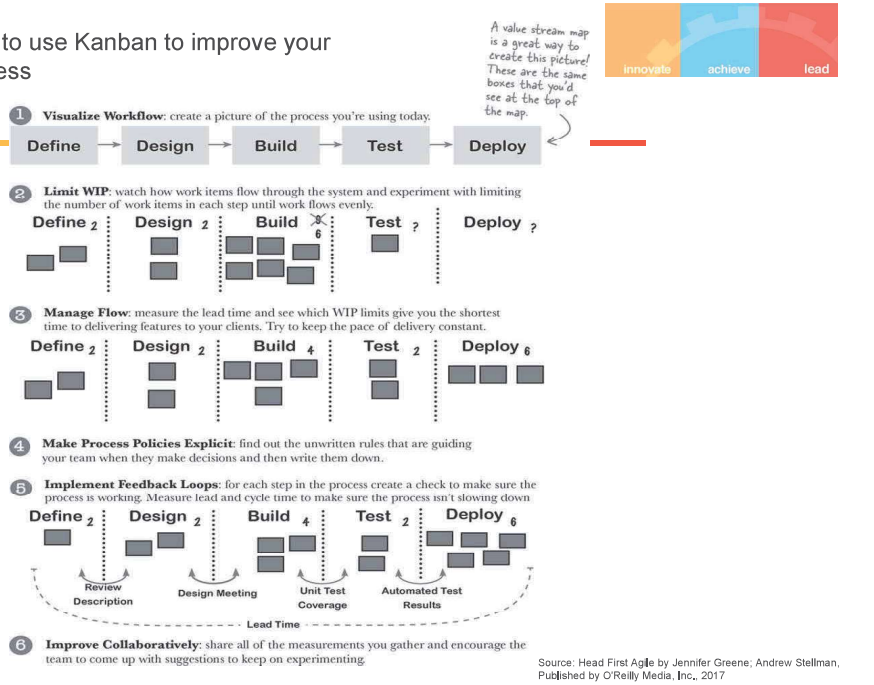


# Kanban

- 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.

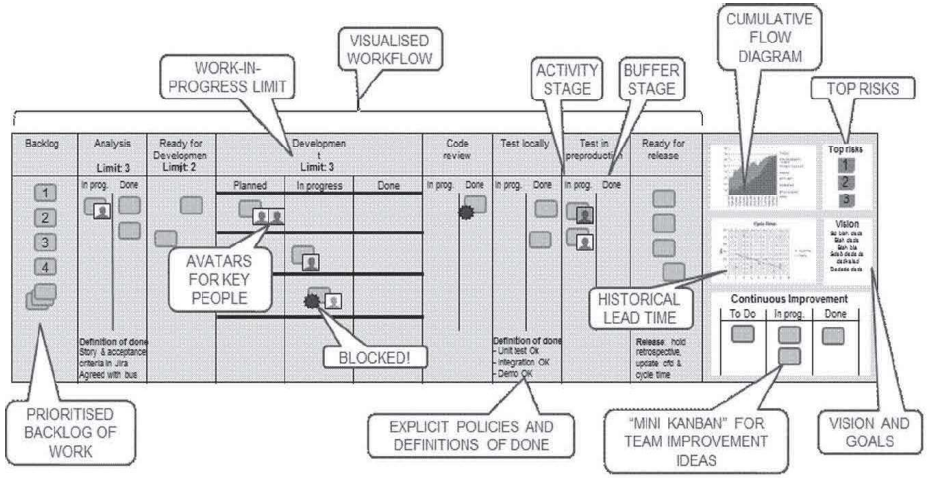


How to use Kanban to improve your process




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

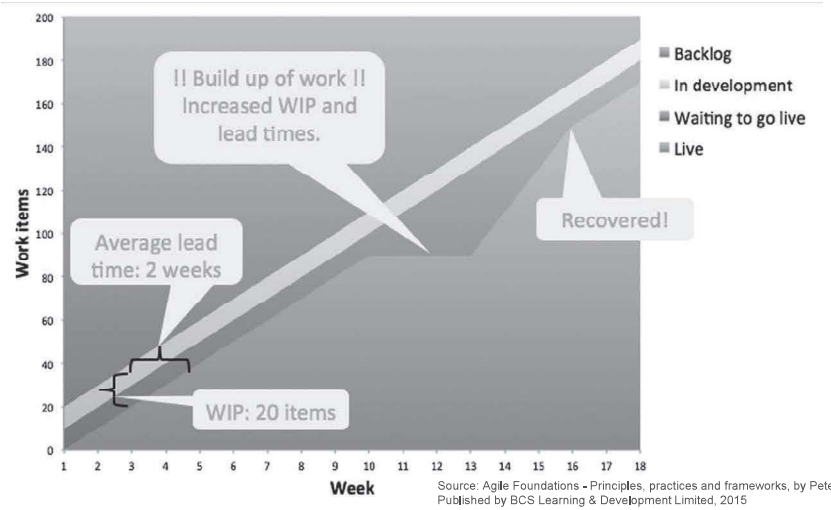
# Typical Kanban board




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



# Cumulative flow diagram (Example-2)

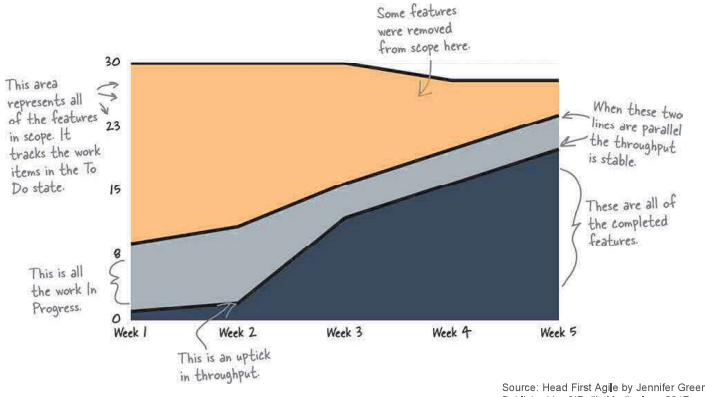



Source: Agile Foundations - Principles, practices and frameworks, by Peter Measey Published by BCS Learning & Development Limited, 2015



# Cumulative flow diagram (Example-1)

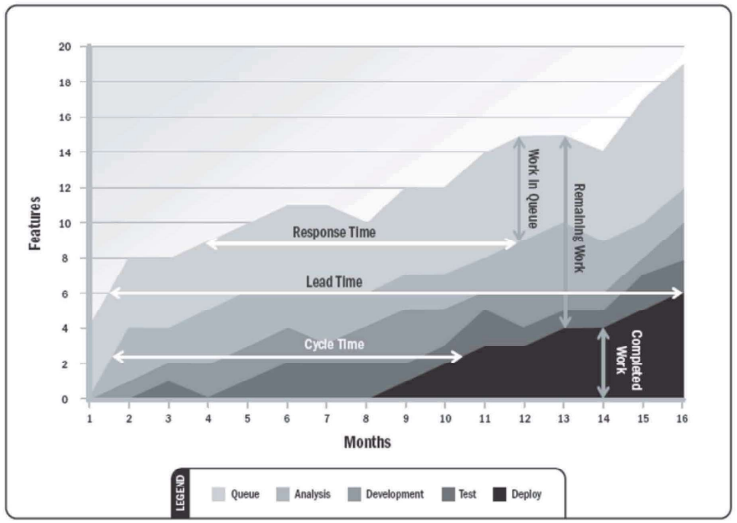
- Kanban teams use cumulative flow diagrams (or CFDs) to find out where they are systematically adding waste and interrupting their flow.




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



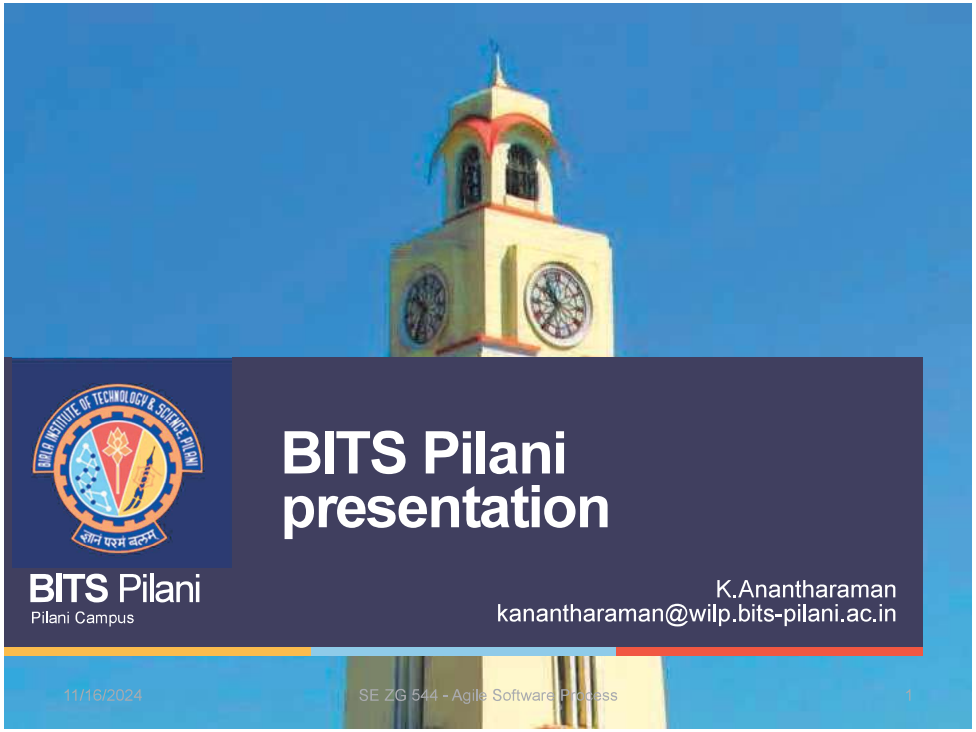
# Cumulative flow diagram (Example-3)



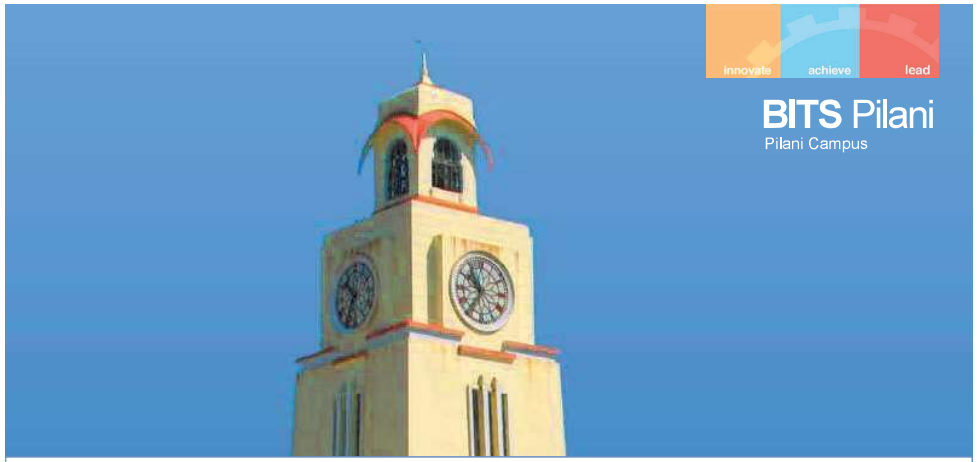
# Quiz

S3



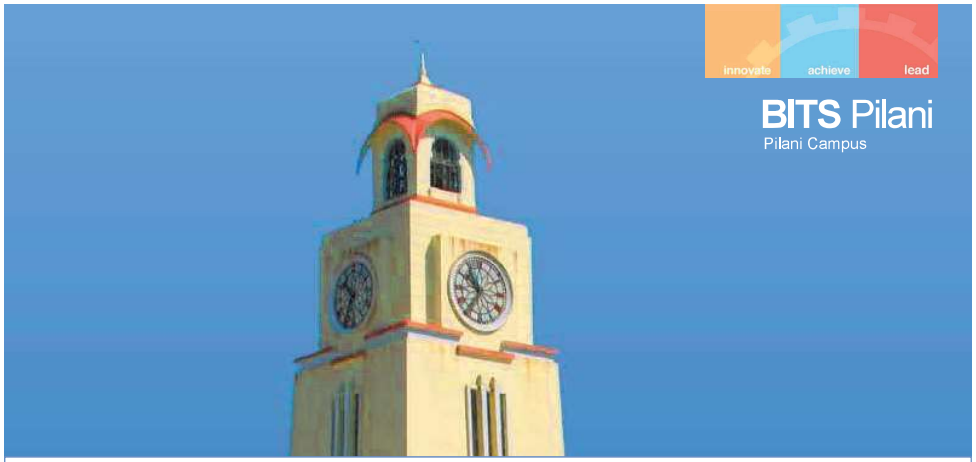





## Thank you






## SE ZG 544 , Agile Software Process Module5 – Agile Requirements & Agile Estimation

# Book References


1. The Agile Sketchpad, O'Reilly Media, Dawn Griffiths, David Griffiths
2. Writing User Stories By Ryan Harper, O'Reilly Media
3. The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010




# 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 ….

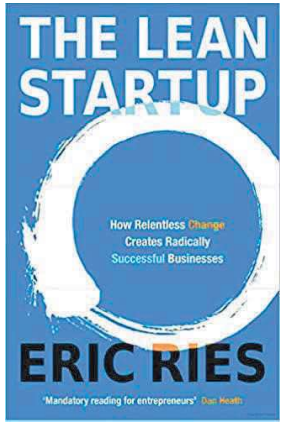


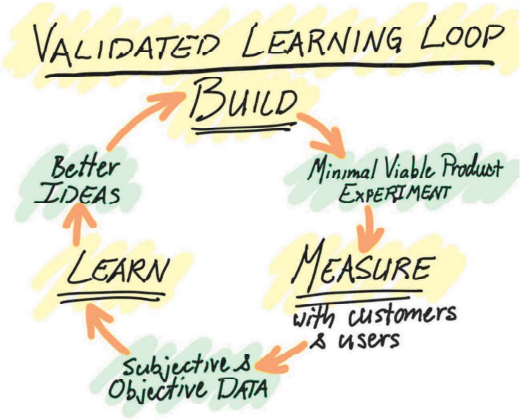
# Requirements Gathering in Agile

- 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.**



# User stories at heart of the build, measure, learn cycle



# Why Write User Stories?

Empathic design is a user-centered design approach

- 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"



# What is the User Story and How it looks?

The User Story Template

As a <type of user>
I want <some goal>
so that <some reason>.

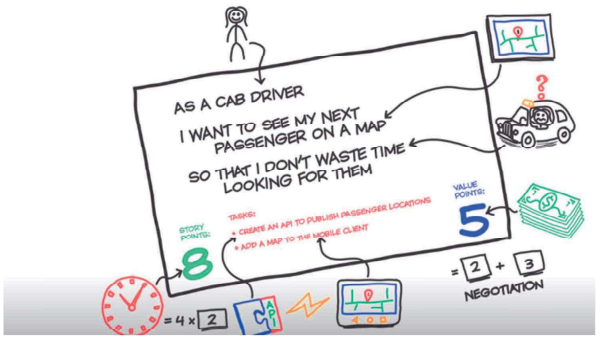
who is this story for
what they want to do
why they want to do it

The user role?

The value (the most difficult part to write in a user story)

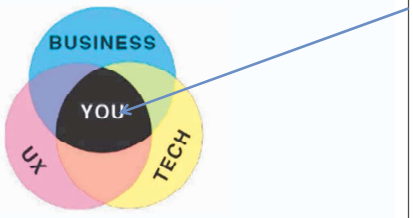
An Example , showing the Front side of the card

User stories are written on Index cards or Sticky notes.
**Why it is written on card?** (Deliberately, not to provide more information, just a promise for a conversation with the customer and the team later.)






# Product Management




- **We need to keep in mind the business, Technology, and User Interface in mind when we write user stories.**
- It is Ok, to help customer to write user stories

In a real situations, User stories are written also by:
- The Agile team
- Other Stake holders



# 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

# 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.



# 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.



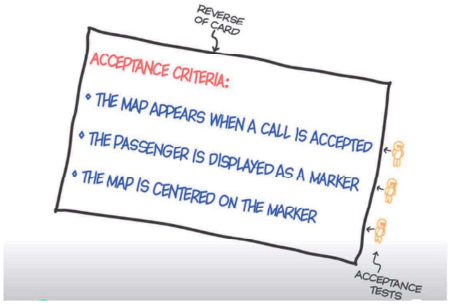
# Writing Acceptance Criteria for User Stories

- 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



# 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."

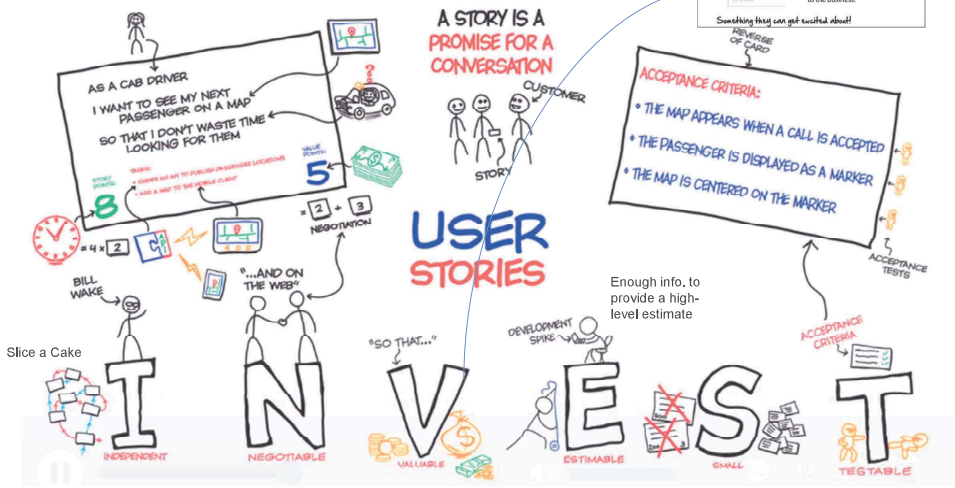
Some possible acceptance criteria:

"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



# Charactertics of a Good User Story (INVEST)






# 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:

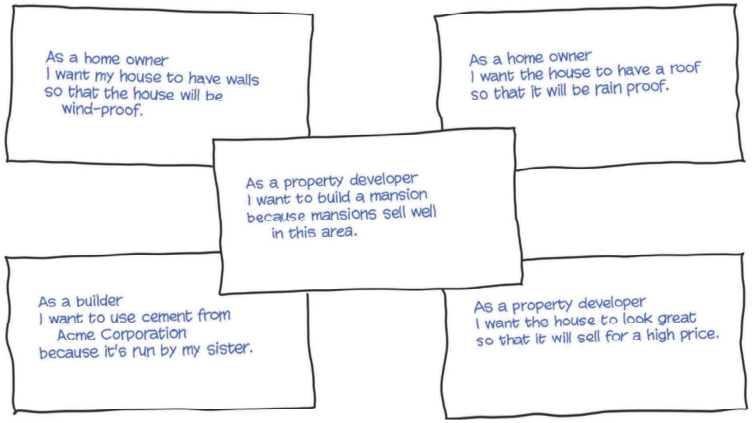
- Independent
- Negotiable
- Valuable
- Estimatable
- Small
- Testable

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



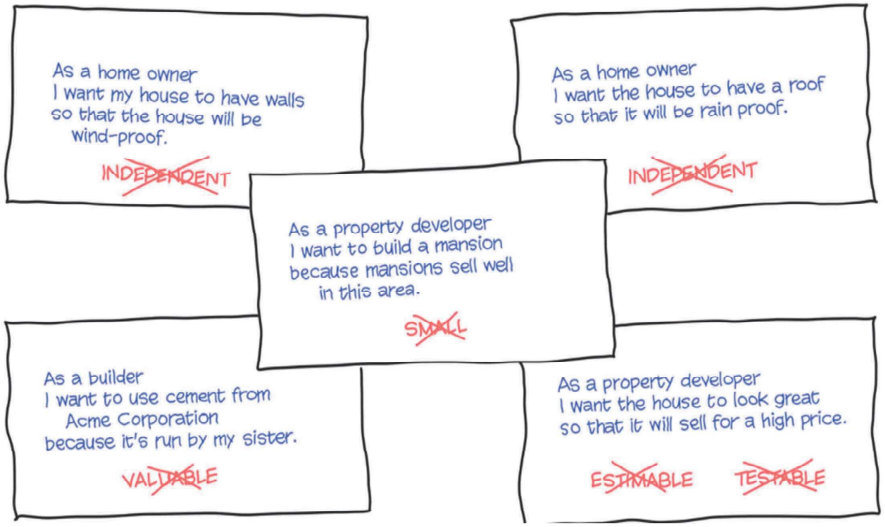
# User Stories – Exercise (Customer is a Property Developer)

Project is to build a House




Apply INVEST test and check if these stories are good or not?

# User Stories – Exercise (INVEST Test)






# 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 superfast

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 superfast can be written like this.

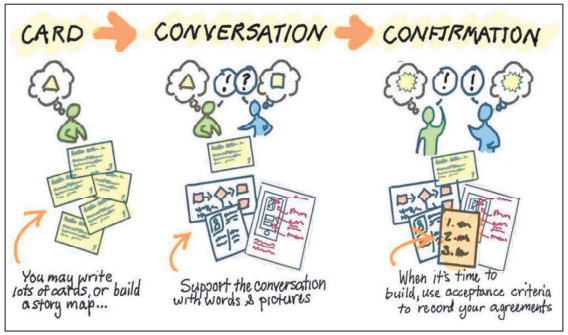
All web pages must load in less than 2 sec → A constraint card



# 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.






# 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/

# How User Stories Promote Empathy

- One benefit of the user story format is that it requires the product manager to think about the feature from the end user's perspective.
- How, when, why, and where the user will interact with the feature will affect how the feature is surfaced.
- What elements are included in the feature, and how the success of the feature will be determined



# 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.

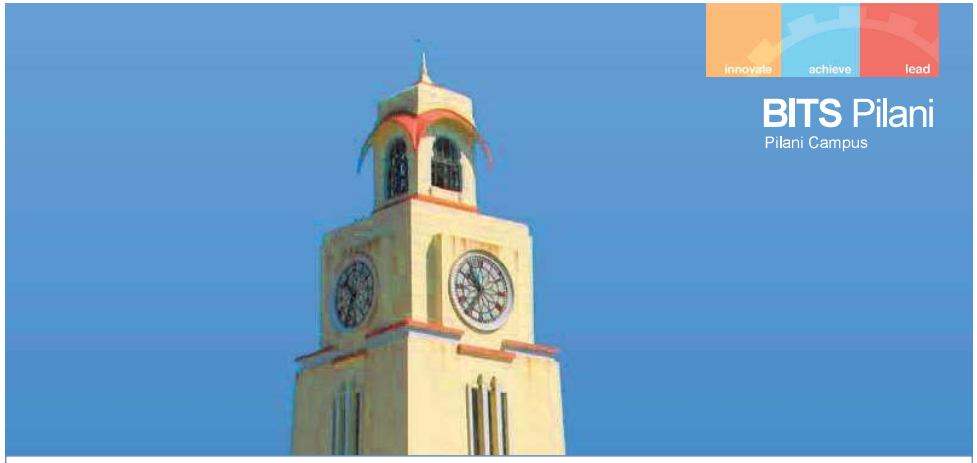


# Example from WebMD apps


This design is not optimal for user, based on the feedback from apps store because no priority order, though the conditions are prioritized

After adding relevance indicator, user able to decide whether to seek medical condition or not






# Agile Estimation

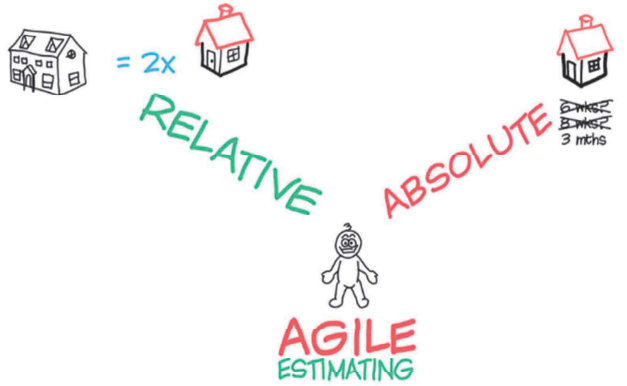
# Absolute Estimation

- 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**



# 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.



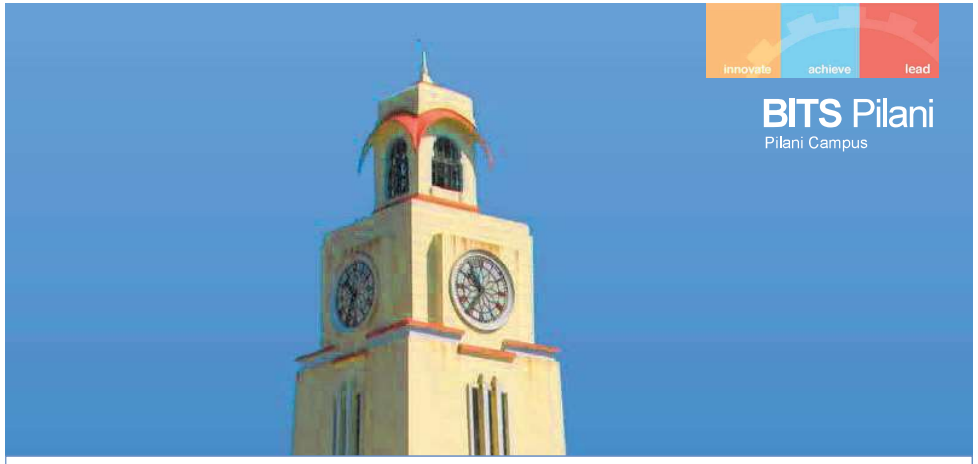
# Relative Estimation

- 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 in 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.



# 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 are we Estimating?

| Product Backlog | |
|---|---|
| As a Cab Driver, I Want to ..... | 3 |
| As a Cab Driver, I Want to ..... | 5 |
| As a Cab Driver, I Want to ..... | 5 |
| As a Cab Driver, I Want to ..... | 8 |
| As a Cab Driver, I Want to ..... | 5 |

| Sprint Backlog | |
|---|---|
| Code the UI | 8 |
| Write Tests | 6 |
| Write DB Stored Procedure | 12 |
| Automate Tests | 5 |

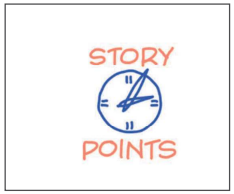
Tasks

Estimated in hrs.

We are talking about these right now



## 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.



# 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.

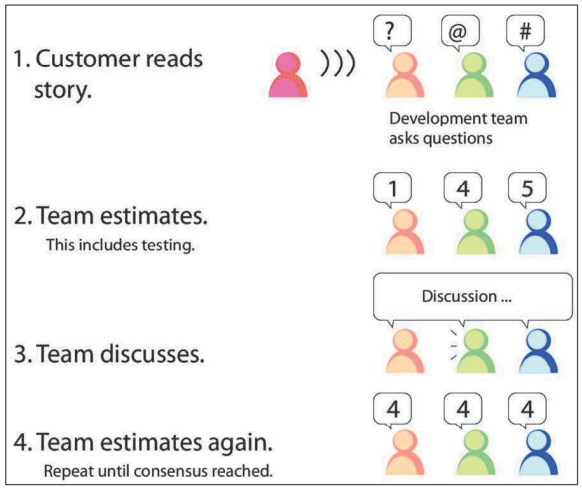


# Predictability of User Stories Estimation

- 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"



# Planning Poker



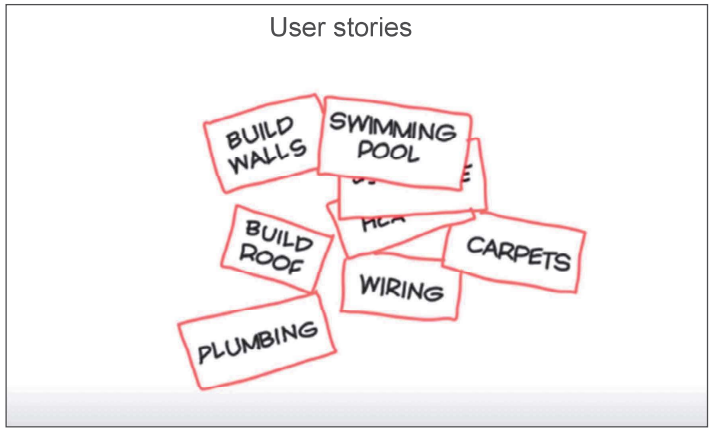


| Estimator | Round1 | Round 2 |
|---|---|---|
| Team member-1 | 3 | 5 |
| Team member-2 | 8 | 5 |
| Team member-3 | 2 | 5 |
| Team member-4 | 5 | 8 |

Image source: https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110

# 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?



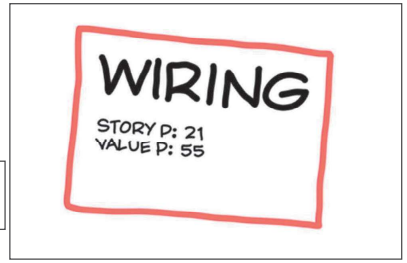



# Value Point

- 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.

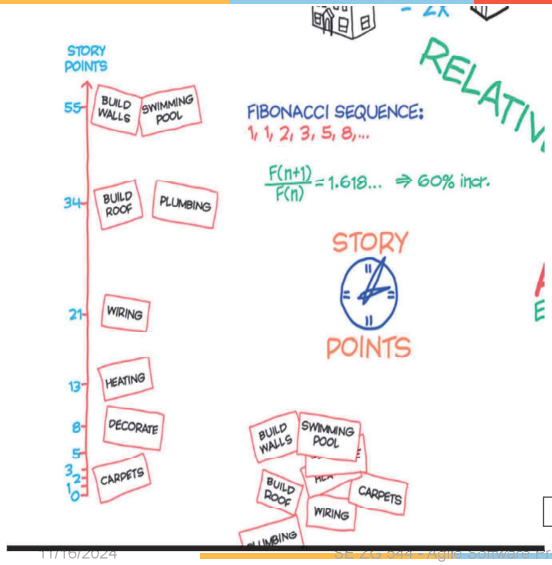
**Story Card**
Story Point : Amount of time/effort
Value Point : Worth to Customer

**Agile is about delivering value early.**






# Story Point Estimation – Example



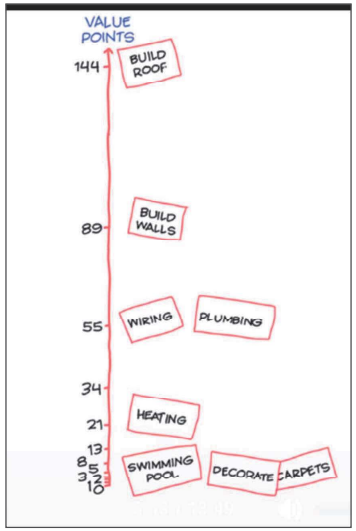

Steps:

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 Estimation – use the previous example

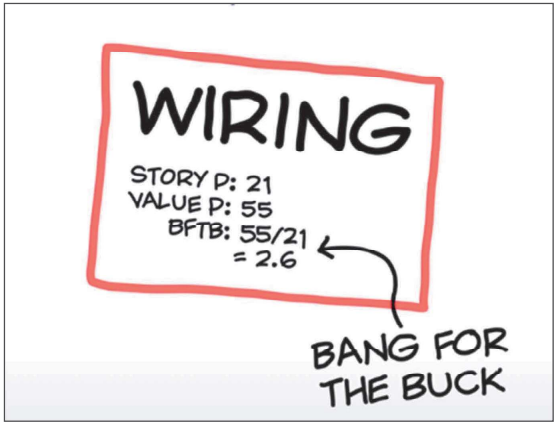



Highest Valued Stories at the top of Product backlog

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



# Velocity

- **Velocity** = Number of story points the team can deliver in an iteration/Sprint (OR)
- The rate at which the team can complete the work within a time frame.
- **Calculating Velocity:**
- For example,

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

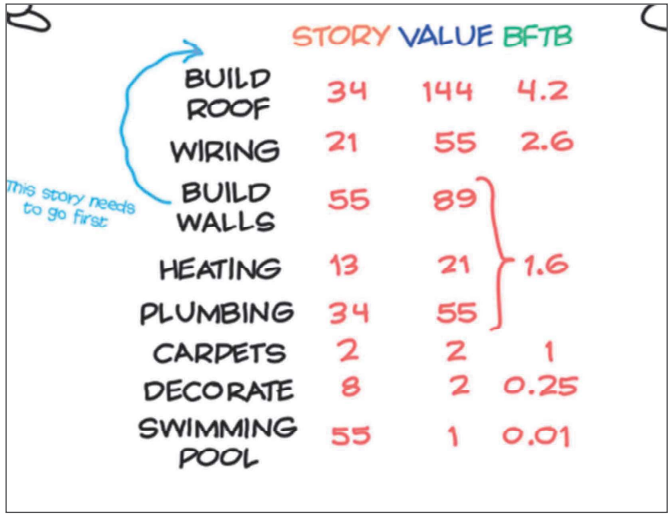
Team Velocity = Average values of the three sprints = (16+15+17) / 3 = 16 Story points per Sprint

- 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.



# How stories are prioritized for each Iteration – by BFTB

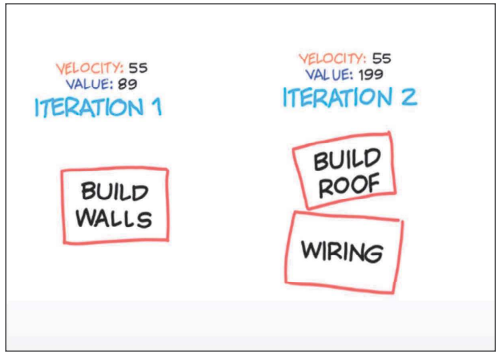
Product Backlog

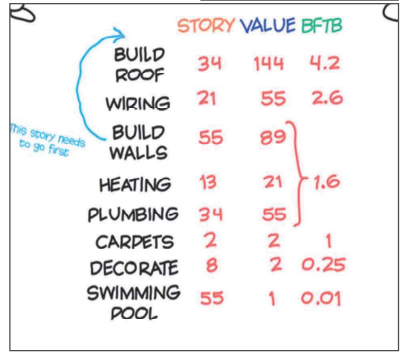





# Highest value delivered in early Iterations

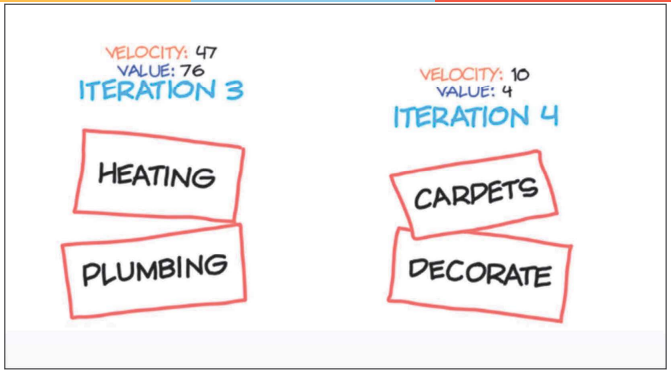
Product Backlog




Suppose, Cost of this iteration-1 is 20000$;
= 20000/89 ~ 225$ Per Value Point.
For Iteration-2 = 225*199 ~ $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.



# Story Points – Real Examples

**Pointing User Stories**

Pointing Rubric at iHeartMedia
(two week development sprints)

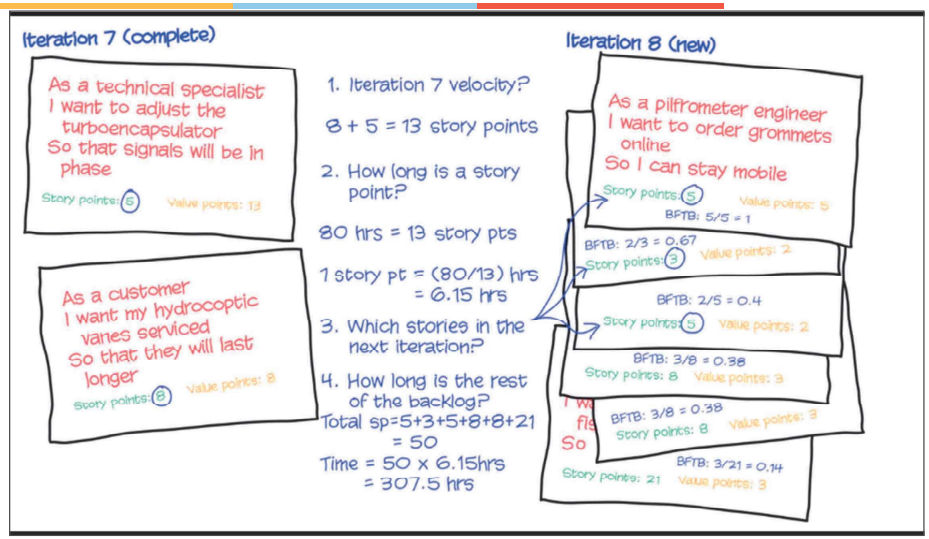
1: Text Change
2: Text Change + Small Functionality Change
3: One Day of Work for One Developer
5: One Week of Work for One Developer
8: Two Weeks of Work for One Developer
13: Two Weeks of Work for Two Developers

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

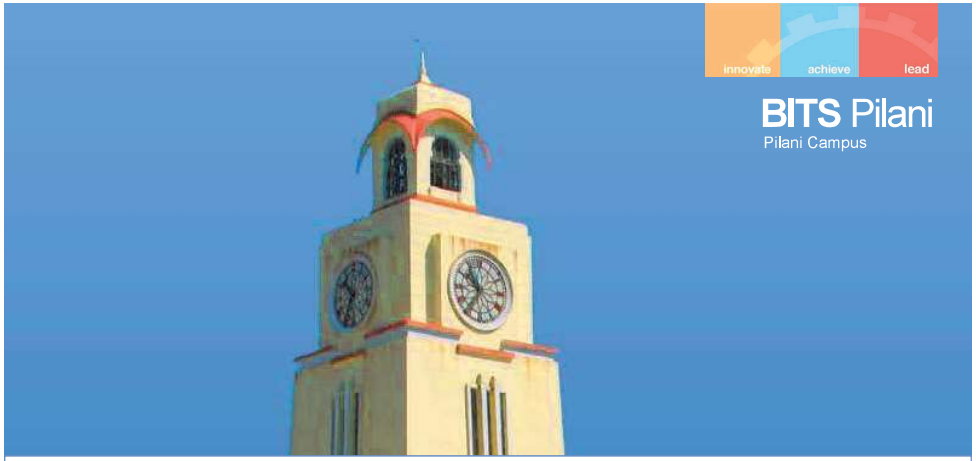
1: Text Change
2: Text Change + Small Functionality Change
3: One Day of Work for One Developer
5: One Week of Work for One Developer
8: One Week of Work for Two Developers
13: Must Be Broken Down Into Smaller Stories



# Estimation Exercise (Assume, 2 week Iteration)






# 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



# 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



# 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 |
| **1point** | Story C | Story D | |



# 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

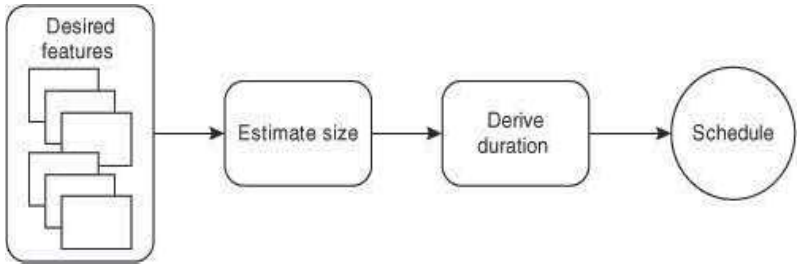
- Level of Effort (LOE) or T-Shirt Sizing
  - T-shirt size," "level of effort" (LOE), or "small, medium, large." (Easy, but lack precision, inability to add up several stories into a meaningful measure.)

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



# 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



# Affinity Grouping

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

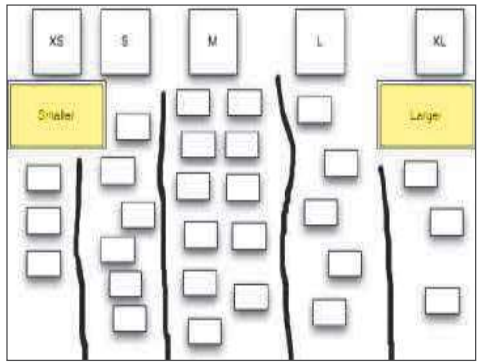


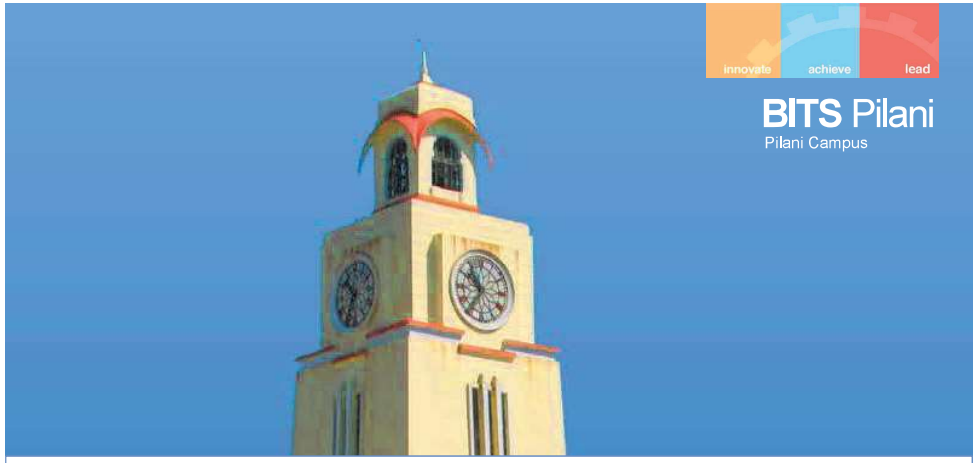

Image source: https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110



# 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




# Supplementary Notes on User Stories





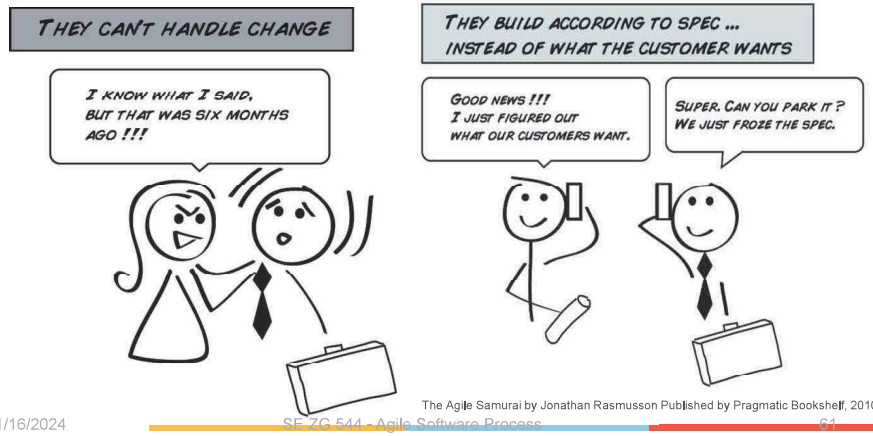
Let's start by looking at how we used to gather requirements and some of the challenges that come with trying to write everything down.





# The problem with upfront requirement documentation

- **Heavy documentation** as a means of **capturing requirements has never really worked** for software projects.
- Here are some other problems teams run into when they rely too heavily on documentation for their software requirements:

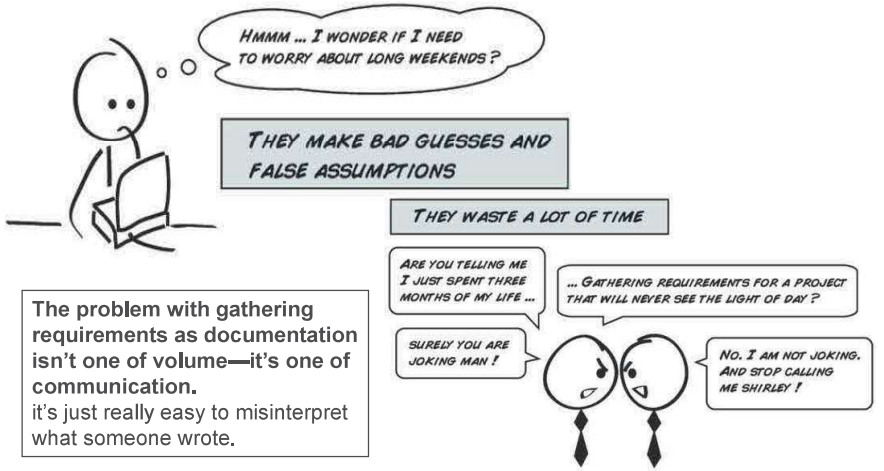



The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010





# More Documentation Issues ...

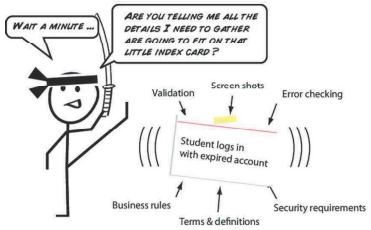


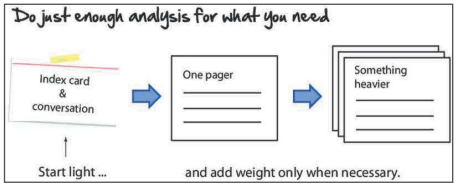

**The problem with gathering requirements as documentation isn't one of volume—it's one of communication.**
it's just really easy to misinterpret what someone wrote.

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010

# What is a User Story?

- Agile **user stories are short descriptions of features** our customer would like to one day see in their software
- Why capture just a few key words and not go to town on the full requirement?
  - Think of a user story **as a promise of a conversation.** At some point, we'll do the deep dive and get in there. But we're not going to do it until we are sure we're going to need it.
  - We **defer diving into the low-level details until later**

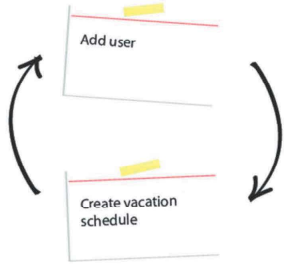


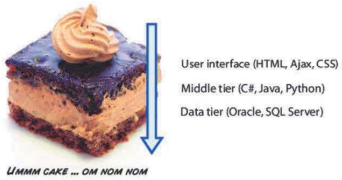




The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# Independent – INVEST acronym




- Things change on projects. What was really important last week can suddenly becomes not so important this week.
- If all of our stories are intertwined and dependent on one another, making trade-offs becomes hard.
- **Characteristic of a really good user story is one that goes from end to end—or as we like to call it, "slices the cake."**
- Slicing our stories from end to end and gathering them by feature enables us to treat the vast majority of **our stories as independent and be flexible on scope when necessary.**

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# 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:

- Independent
- Negotiable
- Valuable
- Estimatable
- Small
- Testable

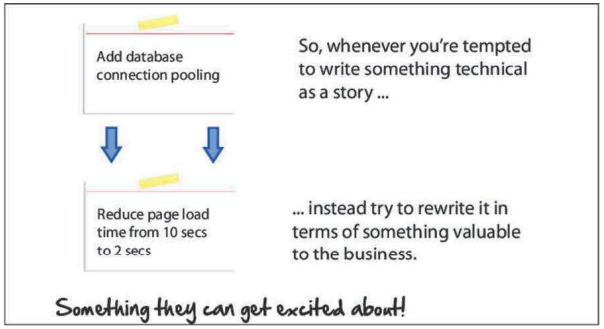
The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# Negotiable – INVEST acronym


- There are always multiple ways to deliver any given story.
- Negotiable stories are nice because they give us the wiggle room we **sometimes need to work within our budgets**

# Valuable – INVEST acronym



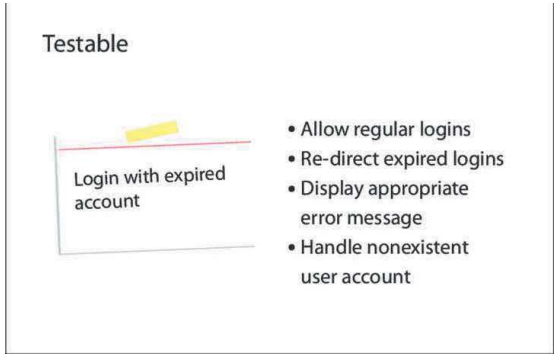

- The important element of a good user story is that it's something of value to our customers.
- What's valuable? Something they would pay for.
- **User stories have to make sense to business**. That's why we always try to write them in simple terms that they understand and stay away from any technical mumbo jumbo.

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# Testable – INVEST acronym

- We like our stories to be testable (as opposed to detestable) because **we like to know when something is working.**
- By writing tests against our user stories, we give the development team a stake in the ground and a way of knowing when they are done.

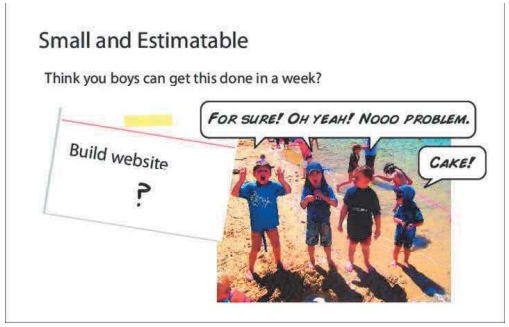



The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# Estimatable & Small – INVEST acronym

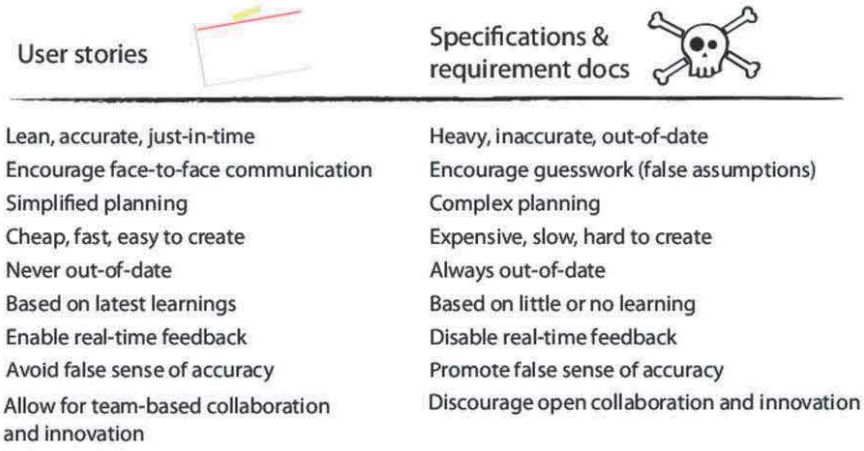
- How do we know a story will fit in within the time frames we have?
  - By **making our stories small** (one to five days)
  - We can ensure they can fit into our one- to two-week iterations, which will enable us to **estimate more confidently**.




The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# User stories vs Documentation for gathering requirements:



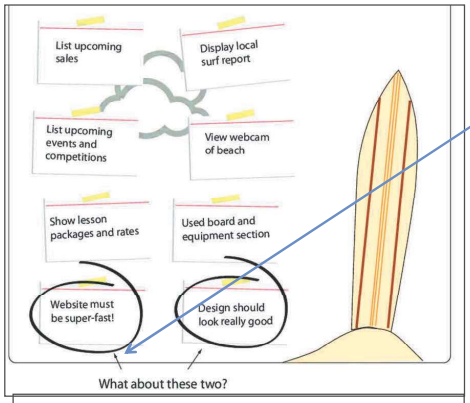

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010

Extracting user stories for an example project,

Big Wave Surf Shop – Website Development



# Extracting User Stories




- **Stories like these, we call constraints.**
- They aren't your typical user stories that we can deliver in a week.
- But they are important because they describe characteristics our customers would like to see in their software.

Sometimes, we'll be able to translate these into testable stories. For example, we could rewrite The website must be super-fast! like this:

All web pages must load in less than 2 sec ← A constraint card

Constraints are important, Capture them on **different colored cards.** Make sure everyone on the team is aware of them, and test for them periodically while you're developing your software.



# Example: Big Wave Surf Shop – Website Development

1. I want the website to be a place for the local scene. Somewhere the kids can come and check out upcoming **events—surf competitions, lessons, things like that.**
2. I need a place to **sell merchandise.** Boards, wet suits, clothes, videos, and things like that. Website have to be easy to use and look really good.
3. I've always wanted **a webcam pointing at the beach.** This way, you don't have to come down to check out the conditions.
   - You can just open your laptop, go to the website, and see whether it's worth getting up. This also means the website has to be fast.

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# The User Story Template

As a <type of user>
I want <some goal>
so that <some reason>.
Value/Benefit

| | |
|---|---|
| **who** | is this story for |
| **what** | they want to do |
| **why** | they want to do it |

For example, using the template, some of our stories for Big Wave surf shop might look like this:

As a surfer who likes to sleep,
I want to check local surf conditions via a webcam
so that I don't have to get out of bed if there is no swell.

As a land-locked Canadian hockey player,
I want to sign up for some adrenaline-pumping lessons
so I can feel the thrill of going "over the falls."

As a grommet looking for the latest surf wear,
I want to see all the latest board shorts and designs
so that I can look stylin' for the Sheilas this summer.

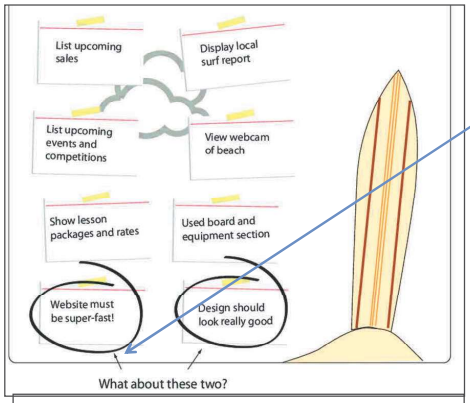
The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010

Extracting user stories for an example project,

Big Wave Surf Shop – Website Development

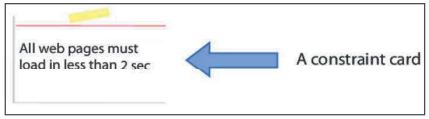


# Extracting User Stories




- **Stories like these, we call constraints.**
- They aren't your typical user stories that we can deliver in a week.
- But they are important because they describe characteristics our customers would like to see in their software.

Sometimes, we'll be able to translate these into testable stories. For example, we could rewrite The website must be super-fast! like this:




Constraints are important, Capture them on **different colored cards.** Make sure everyone on the team is aware of them, and test for them periodically while you're developing your software.



# Example: Big Wave Surf Shop – Website Development

1. I want the website to be a place for the local scene. Somewhere the kids can come and check out upcoming **events—surf competitions, lessons, things like that.**
2. I need a place to **sell merchandise.** Boards, wet suits, clothes, videos, and things like that. Website have to be easy to use and look really good.
3. I've always wanted **a webcam pointing at the beach.** This way, you don't have to come down to check out the conditions.
   - You can just open your laptop, go to the website, and see whether it's worth getting up. This also means the website has to be fast.

The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010



# The User Story Template

As a <type of user>
I want <some goal>
so that <some reason>.
Value/Benefit

| | |
|---|---|
| **who** | is this story for |
| **what** | they want to do |
| **why** | they want to do it |

For example, using the template, some of our stories for Big Wave surf shop might look like this:


The Agile Samurai by Jonathan Rasmusson Published by Pragmatic Bookshelf, 2010

# BITS Pilani presentation


**BITS** Pilani
Pilani Campus

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




## Agile Planning

innovate achieve lead

Harvard Business Review

Diversity Latest Magazine Popular Topics Podcasts Video Store The Big Id

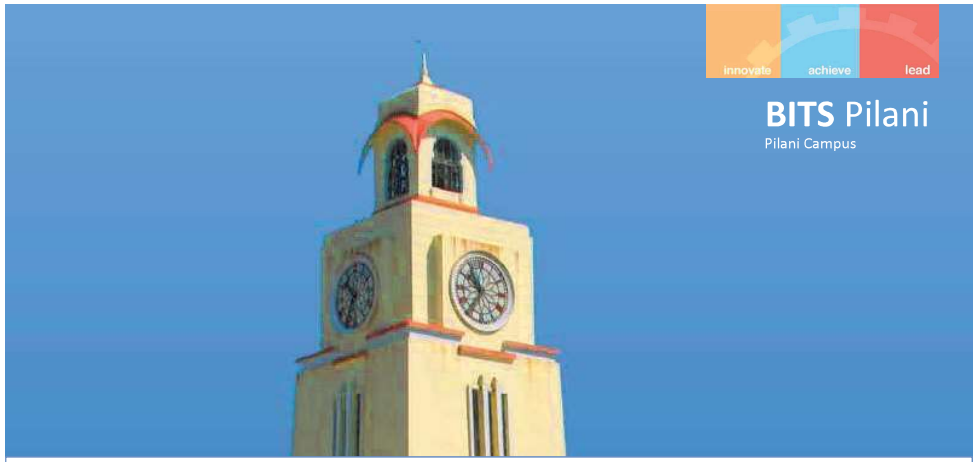
MANAGING ORGANIZATIONS

### Bring Agile Planning to the Whole Organization

by Jeff Gothelf

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





innovate achieve lead

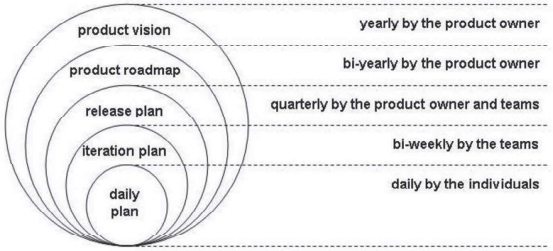
**BITS** Pilani
Pilani Campus

# Module-6 Agile Planning & Release Planning



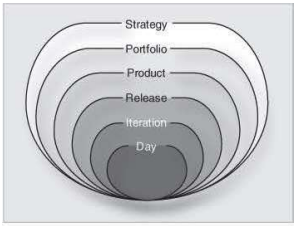
## Agile Planning

innovate achieve lead




Single Product Organization

Many Products or Services 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)




# 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 |

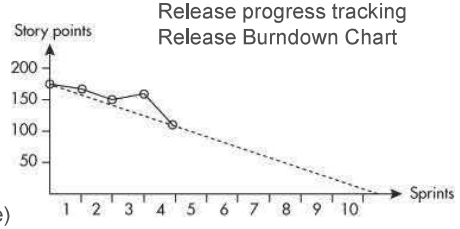
Source: Agile Estimation and Planning by Mike Cohen



# 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)

Release progress tracking
Release Burndown Chart




**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)

Source: Agile Estimation and Planning by Mike Cohen



# High-confidence forecast- Example

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

Sorted Velocity List
16
20
20
25
26
26
26
28

90% Confidence Interval

| Number of Velocity Observations | nth Velocity Observation |
|---|---|
| 5 | 1 |
| 8 | 2 |
| 11 | 3 |
| 13 | 4 |
| 16 | 5 |
| 18 | 6 |
| 21 | 7 |
| 23 | 8 |
| 26 | 9 |

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

Use the nth Lowest and the nth 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 (Question)

- 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 ~ 6 Sprints
- Sprint 1-2 = 15 points; Sprint 3-7 = 20; Total number of points that ca be delivered = 130

- Velocity low = 18; Number of Iteration Required = 140/18 ~ 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 ~ 7 Sprints
- Sprint 1-2 = 15 points; Sprint 3-7 = 18; Total number of points that ca be delivered = 120



# Creating a Release Plan Exercise

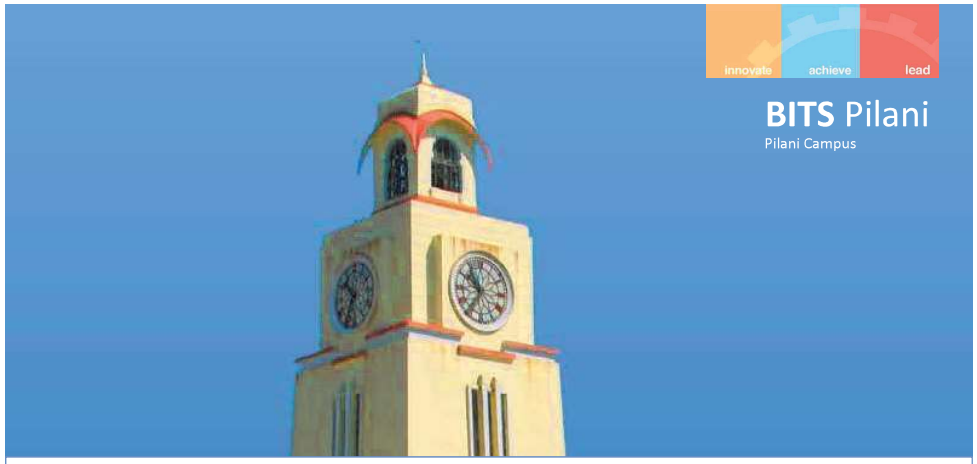
2. What is the maximum and minimum timeline and number of points that can be completed if the budget is $140000 and the cost of a sprint is $20000?

- 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 =5.5 ~ 6 sprints

- Sprint 1-2 = 15 points; Sprint 3-7 = 20 points per sprint ; 20* 5 sprints = 100 points
- Total number points that can be delivered = 30+100 = 130 points

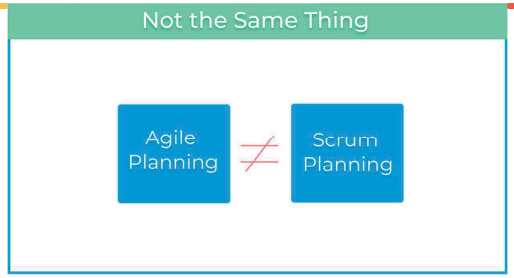
- Velocity low = 18;
- Number of Story points completed in first two sprints= 30
- Remaining Story points = 110
- Number of iteration required to complete the 110 points = 110/18 =6.1 ~ 7 sprints
- Sprint 1-2 = 15 points; per sprint = 5*18 = 90 points
- Total number points that can be delivered = 30+90 = 120 points
- 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



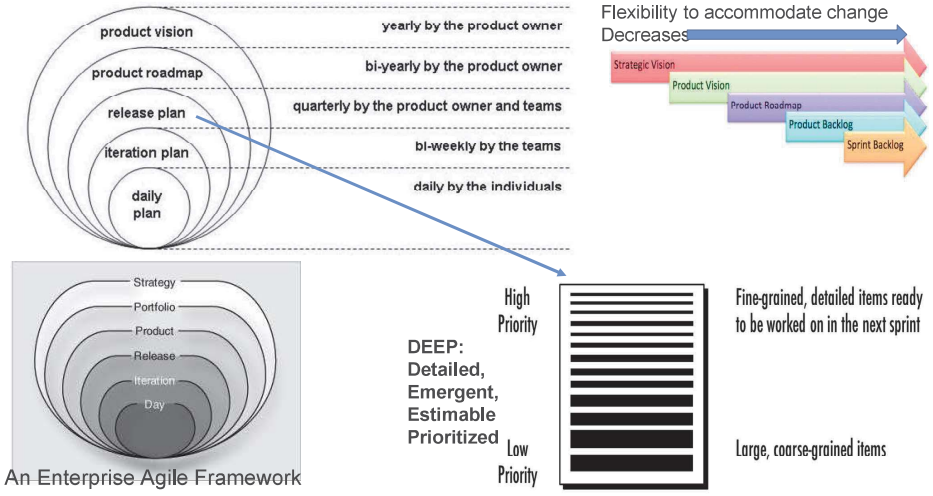
# 5-Levels of Agile Planning (Product Planning)

- 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

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



# 5-Levels of Agile Planning (Product Planning)

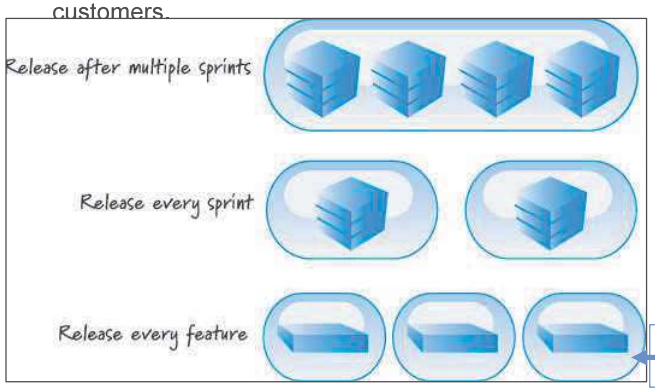



Ref: 5 Levels of Agile Planning: From Enterprise Product Vision to Team Stand-up by Hubert Smits



# 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

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

# 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/



# 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/



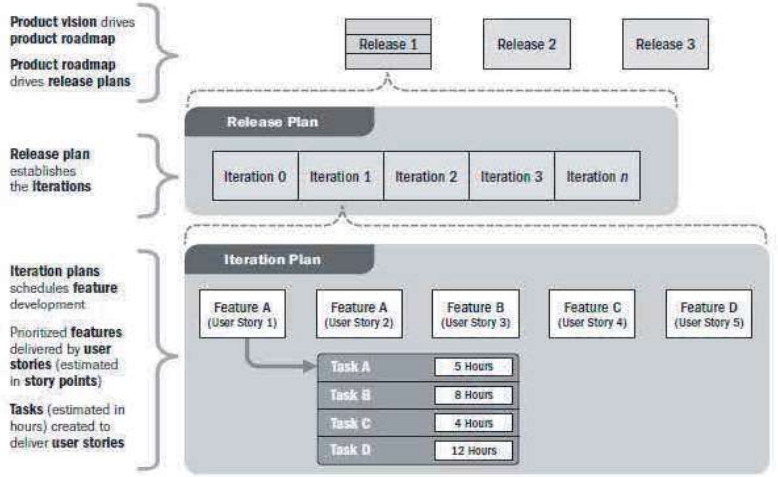
# Agile Release Planning …

- 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/

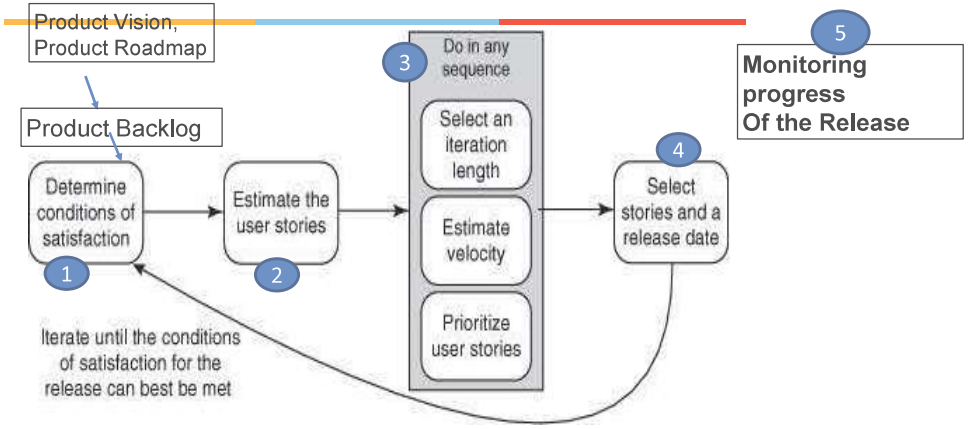


**Relationship between product vision, product roadmap, release planning, and iteration planning.**




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.

Source: Agile Estimation and Planning by Mike Cohen



# 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.

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



# 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 |

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



# 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.

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

# 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 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 terations 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 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 thump: 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

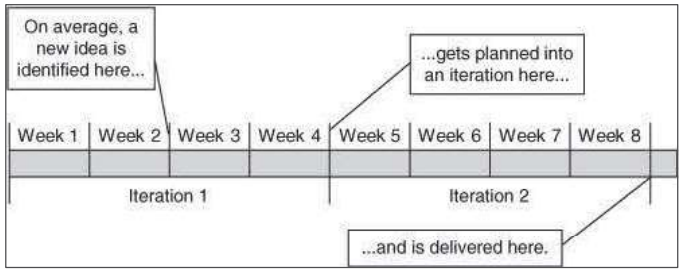


# The ease of getting feedback

- 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.




- It takes an average of 1½ iterations to go from idea to software.

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



# 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.

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.

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



# 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.

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

# 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.
- “Rhythem is a significant factor that helps achieve a sustained pace”



# 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.



# 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 interation 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 × 2 + 1.”
  - During the one-week iteration, however, the team chooses its own work.



# 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



# 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 |

| Product Backlog | |
|---|---|
| As a Cab Driver, I Want to ..... | 3 |
| As a Cab Driver, I Want to ..... | 5 |
| As a Cab Driver, I Want to ..... | 5 |
| As a Cab Driver, I Want to ..... | 2 |
| As a Cab Driver, I Want to ..... | 5 |

60hrs

60hrs

22hrs

60hrs

| Sprint Backlog | |
|---|---|
| Code the UI | 10 |
| Write Tests | 8 |
| Write DB Stored Procedure | 12 |
| Automate Tests | 10 |

40hrs

Velocity = 3+5+5+2+5=20 Story Points

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.

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



# 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

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

## 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 | nth 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
28

90% Confidence Interval

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

Source: Succeeding with Agile SW development by Mike Cohen

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



## 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 | | | | | |
| Depend-encies | | | Imaging library | | | | | |
| Releases | | | | Alpha: Calls, basic text messages | Holidays | Beta: Con-ference 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



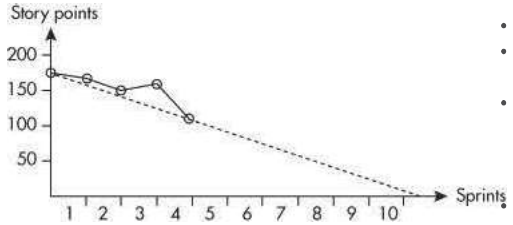
## Creation a Release Plan

- 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.

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



## 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 it's USP(s)?,What are the business goals?
2. Phrasing the product vision in one inspiring sentence.
   - Examples: Google's companies 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 – An Example

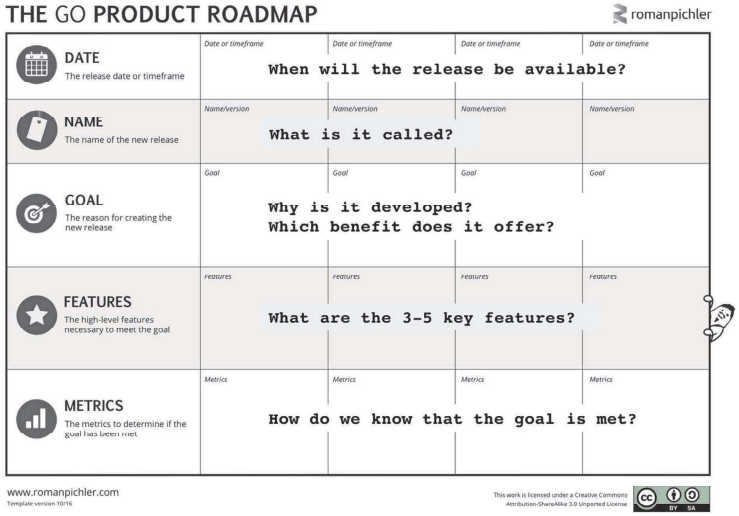
Develop a new dance game for girls aged eight to 12 years. The app should be fun and educational allowing the players to modify the characters, change the music, dance with remote players, and choreograph new dances.

| | 1st quarter | 2nd quarter | 3rd quarter | 4th 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 |
| | • Basic game functionality<br>• Multiplayer<br>• FB integration | • Purchase dance moves<br>• Create new dances | • New characters and floors<br>• Enhanced visual design | • Street dance elements<br>• Dance 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/



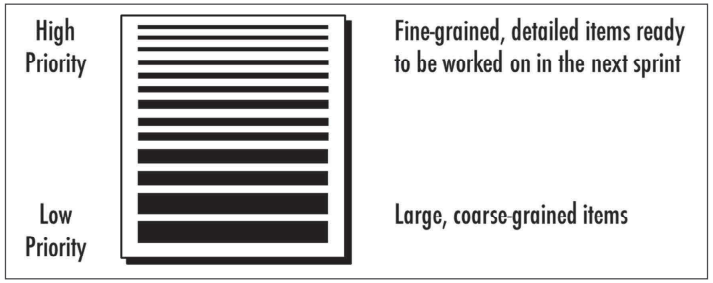
# Goal Oriented Roadmap






# 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.




The product backlog is the responsibility of the product owner who cares about content, availability, and priority of the backlog.

* 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/




# Project Trade-off Matrix

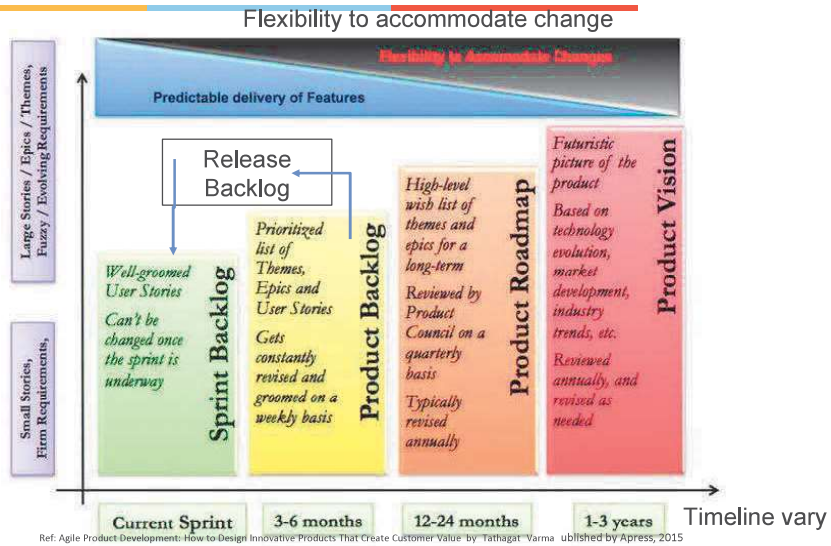
| | Fixed | Flexible | Accept |
|---|---|---|---|
| Scope | X | | |
| Schedule | | X | |
| Cost | | | X |

- 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



# 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 ublished by Apress, 2015



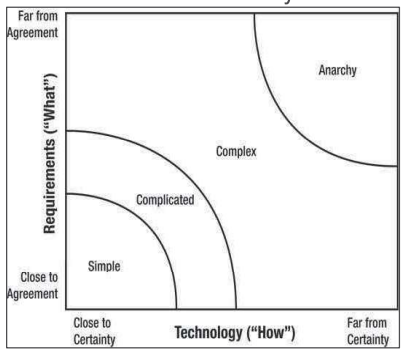
# Exploration Factor

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

| Product Requirements Dimension | Product Technology 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% |

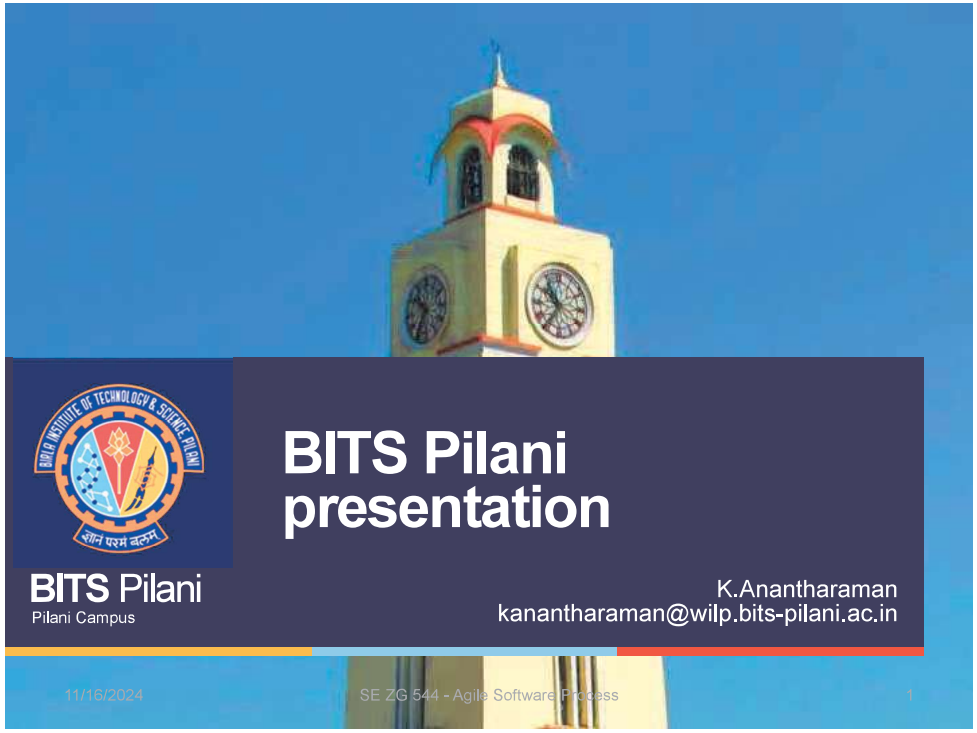
Stacey Matrix



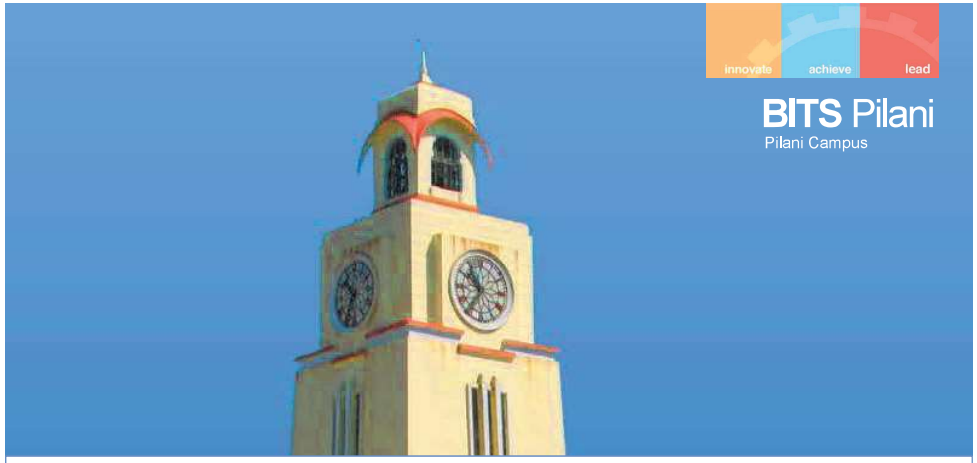

Ref: Agile Project Management: Creating Innovative Products, Second Edition by Jim Highsmith Published by Addison-Wesley Professional, 2009

End








# SE CZ 544 , Agile Software Process Module – 7 Scrum Iteration Planning



## References

1. Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012
2. LinkedIn Learning - Agile Software Development: Scrum for Developers with Shashi Shekhar.

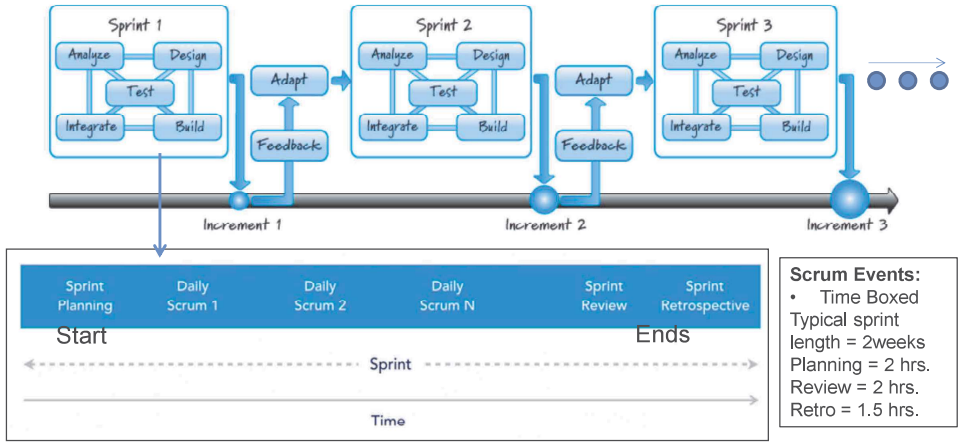
# Topics

- Sprint Overview
- Sprint Planning
- Additional notes
  - Time Boxing
  - Scrum Practices
  - Short Sprints
  - Capacity planning
  - Selecting Product Backlog Items (PBI)
  - Acquiring Confidence
  - Definition of Done
  - Definition of Ready



# Sprint Overview

- Scrum teams build products in an iterative and incremental manner. Each time box iteration of work is called a sprint.




**Scrum Events:**
- Time Boxed

Typical sprint length = 2weeks
Planning = 2 hrs.
Review = 2 hrs.
Retro = 1.5 hrs.



# Potentially Shippable Product Increment (PSI)

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



# Sprint Practices

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

# Spike - Story

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



# Sprint Planning

- Each sprint begins with an event called sprint planning.(Time boxed :4 hrs. for 2 weeks Sprint)

- Sprint Planning is broken into two parts:
- **Part -1 :** Team defines the Sprint Goal: (What Part)
- **Part -2 :** Team defines how they build the product increment. ( How Part)
- **Participants:**
  - Entire Scrum Team, Product Owner, Scrum master, other Stakeholders to figure out how to maximize business value of the work that needs to be done in the current sprint.



# Sprint Planning – Part-1

- Team defines the Sprint Goal

- **What is the Sprint Goal?**
  - Defines what they are going to build in the sprint.
  - Describe business purpose and value of the sprint.

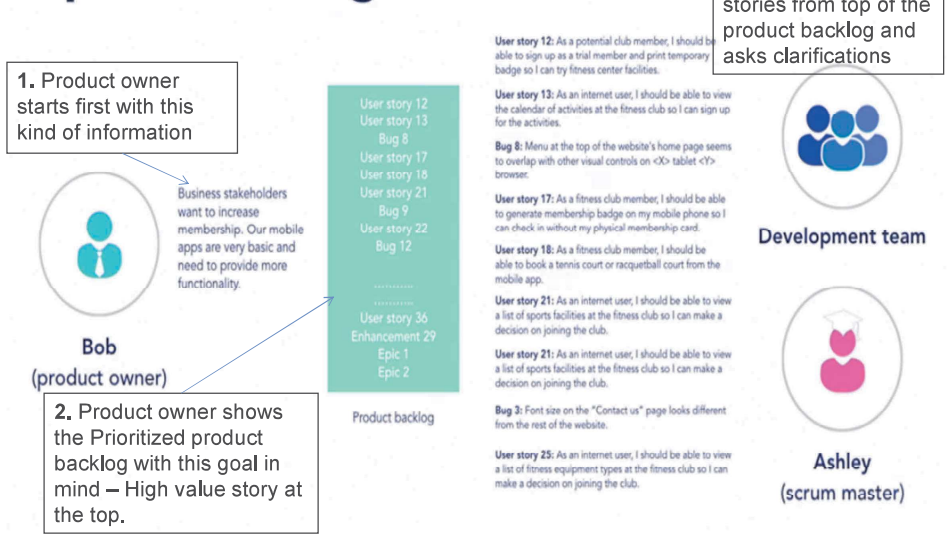
- **Sprint Goal Benefits:**
  - Inspire the team and gives focus.
  - Facilitates prioritization and effective teamwork;
  - Easier to obtain and analyze feedback
  - Helps with stakeholder communication.



# Examples of Sprint goals

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

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






# Sprint Planning –Part -1 Key Takeaways

- Development team pulls items based on product backlog priority, team velocity, team capacity
- This is the *what* of what the team will work on
- Architectural spikes should also be accounted for in the total volume of work selected



# What the Development team pulled?


- The Dev. Team review the items pulled for the sprint based on the initial sprint goal.
- Product owner provides additional clarifications on a few items.

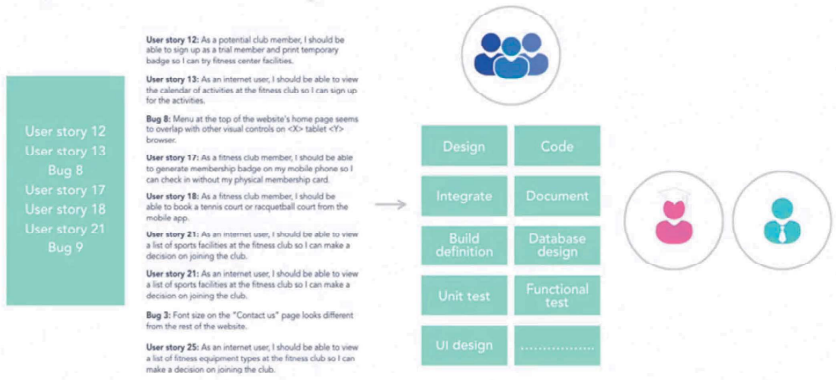
- **The development team pulls items from the backlog based on their availability, past performance history, and team capacity.**
- **Team capacity is determined based on Mean Velocity or Available hours**
- The team then continues to define the sprint goal, which is the short executive summary of what the team wants to accomplish in this sprint.
- The Sprint goal is finalized after completing the Part-2 of the planning



# Sprint Planning – Part-2 – The How

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

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

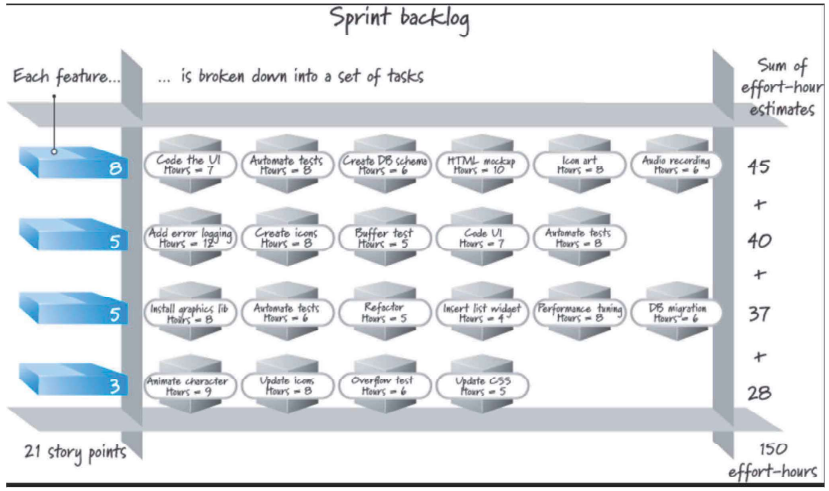



Note: This is not an elaborate plan for the entire sprint. This is just a collection of tasks for the next few days. More planning will be done by the development team as they finish tasks and learn more about the tasks at hand.- **em-pir-i-cism, Last responsible moment.**

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



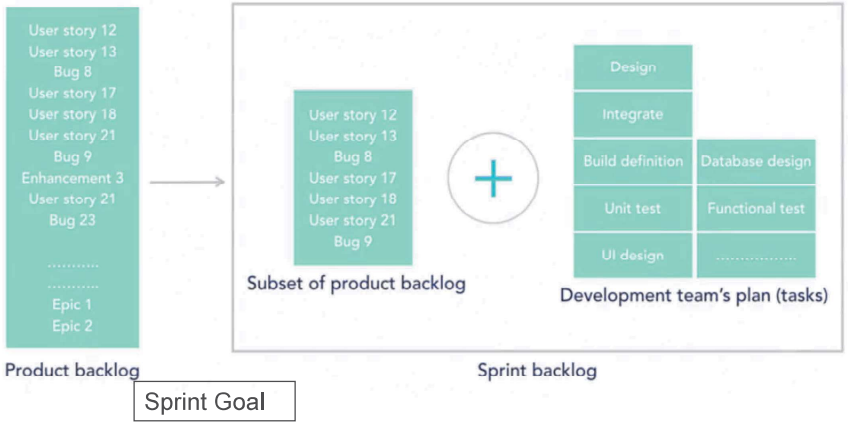
# Sprint Backlog






# The Sprint Planning Ends with Sprint Backlog + Team Commitment + Refined Sprint Goal

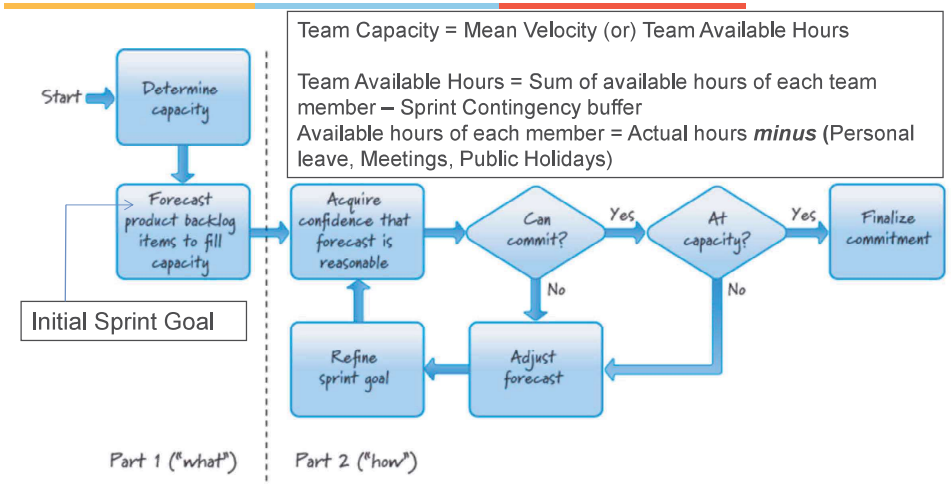
- Most Team break all user stories pulled for the sprint into tasks and estimate in ideal hours. Then compare with team capacity to establish confidence

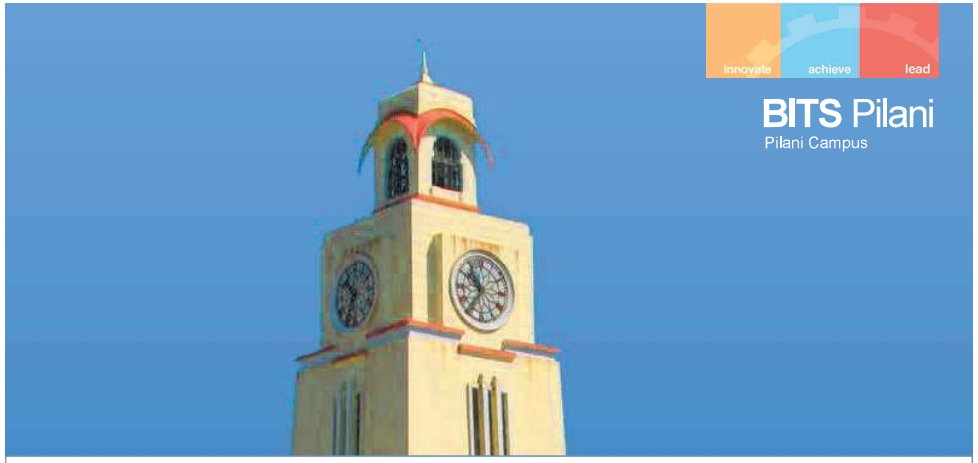



Sprint Goal



# The Process flow

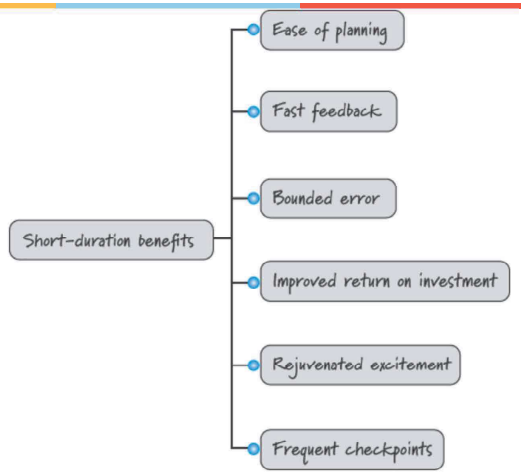





# Sprint Planning – Additional Notes

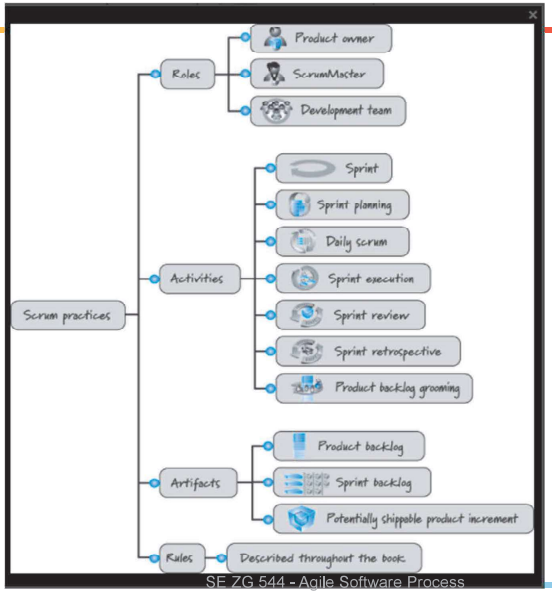


## Short Sprints



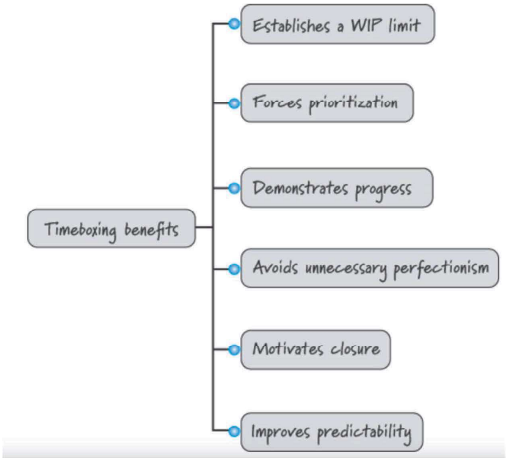



## Scrum Practices

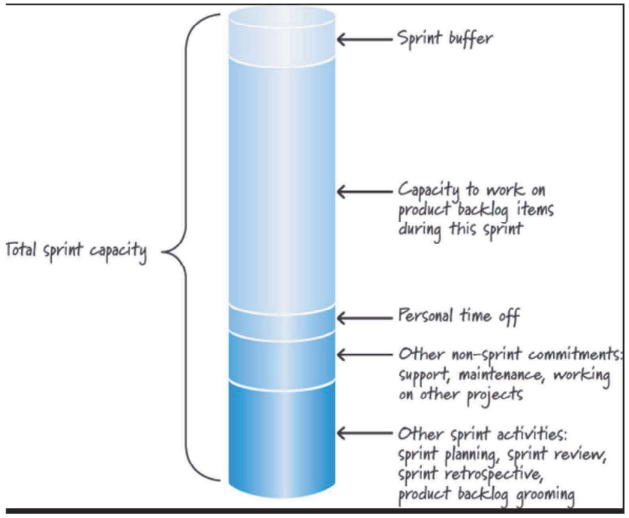





## Time Boxing



# Development team capacity in a sprint






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

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

Caution: Taking 197 hours of work because it would leave no sprint buffer. Better strategy: > 140 hrs. and < 197 hrs.

- If all team members are available full time and no personal holidays – Capacity = (Available Capacity) – (Total Days for Scrum activities ) – Sprint buffer (Assume 5%)

= 400 – 80 – 20 = 300 hours – plan for 320-300 hours of capacity



# Capacity in Story Points

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



# Selecting Product Backlog Items

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

# Examples of Product Backlog Items (PBI)

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



# Acquiring Confidence …

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



# Acquiring Confidence

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



# Definition of Done

- An Example

| Definition of Done |
|---|
| ❑ Design reviewed |
| ❑ Code completed<br>❑ Code refactored<br>❑ Code in standard format<br>❑ Code is commented<br>❑ Code checked in<br>❑ Code inspected |
| ❑ End-user documentation updated |
| ❑ Tested<br>❑ Unit tested<br>❑ Integration tested<br>❑ Regression tested<br>❑ Platform tested<br>❑ Language tested |
| ❑ Zero known defects |
| ❑ Acceptance tested |
| ❑ Live on production servers |

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

# Definition of Done …

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



Thank you



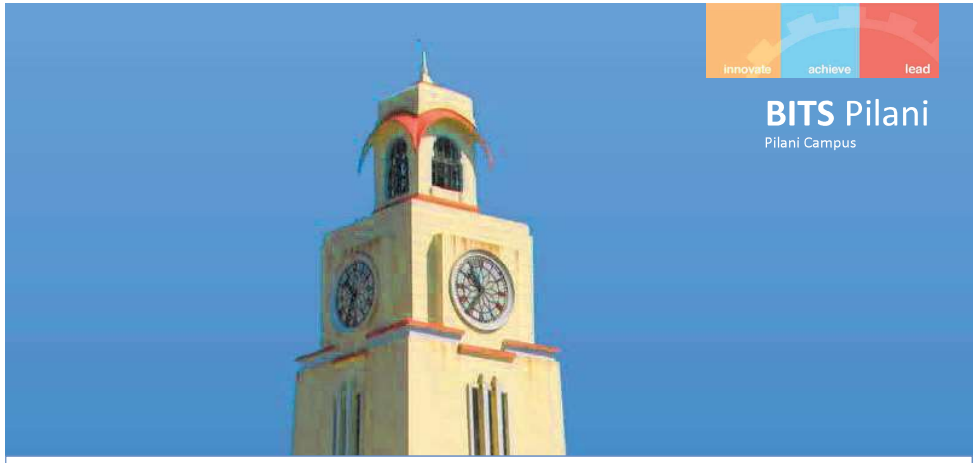
# Definition of Ready

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

Example: Definition of ready

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





BITS Pilani
Pilani Campus

# Module8- Executing a Sprint



## Daily Scrum

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

15-minutes everyday
Same place, same time

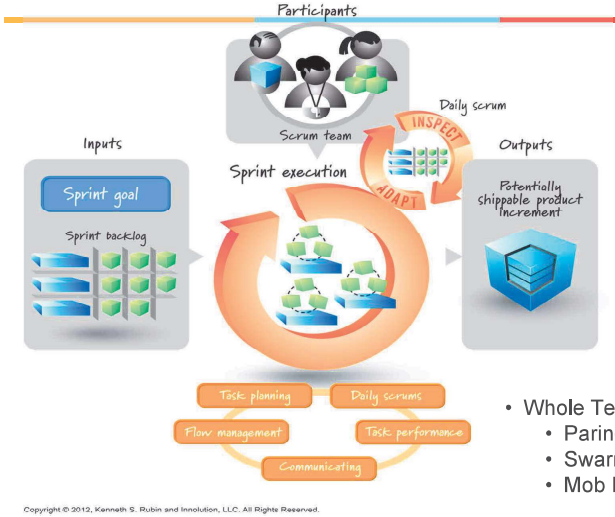
**3 important questions in Daily Scrum**

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

Ref: GenMan Solutions



## Sprint Execution

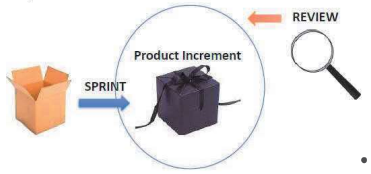





- Whole Team
  - Paring
  - Swarming
  - Mob Programming



## Sprint Review

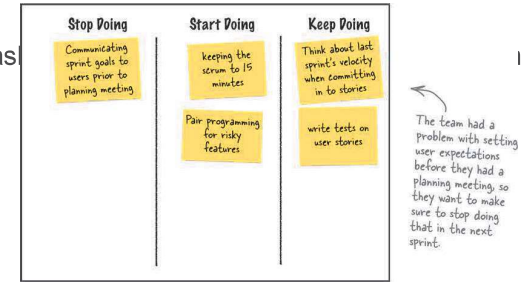



- Duration: Max 4 hrs. or less for 4 week sprint
- Products invites all attendees
- Development Team Demo the completed items.
- Product owner verifies and lists completed and incomplete items.
- Development team share sprint experience and highlights challenges.
- The Product Owner updates the Product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

Ref: GenMan Solutions

# Sprint Retrospective

- Agenda:
  - Set the stage, Gather Date ,Generate insights, Decide what to do, Close
- Simplest Approach
- Each team member is asked what they should:
  - Start doing
  - Stop doing
  - Continue doing




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

Ref: Head First Agile by By Andrew Stellman and Jennifer Greene



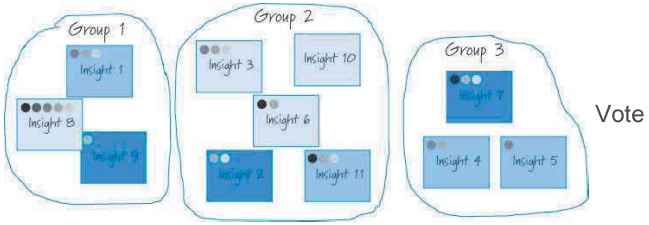
# Communicating: The Progress

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



# Sprint Retrospective

- Group insights




Vote

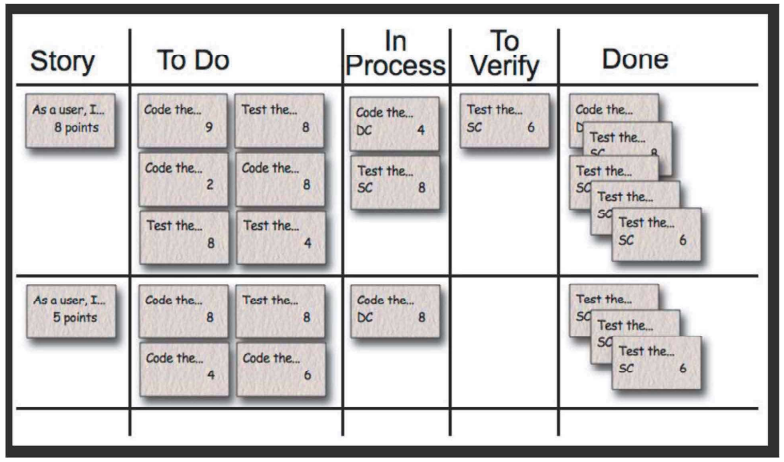
Determine Actions

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

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

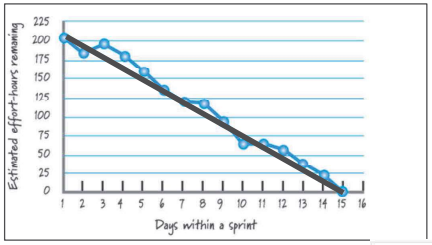


# Sprint Task Board




Ref: Agile Estimation and Planning by Mike Cohn

# Sprint burndown chart



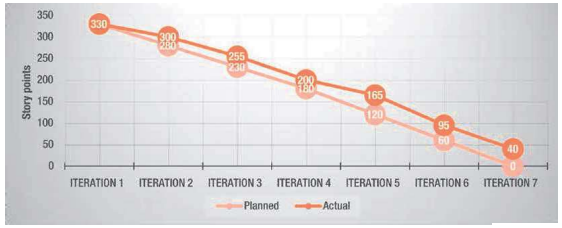


Ref: Essential Scrum: A Practical Guide to the Most Popular Agile Process by Kenneth S. Rubin Published by Addison-Wesley Professional, 2012

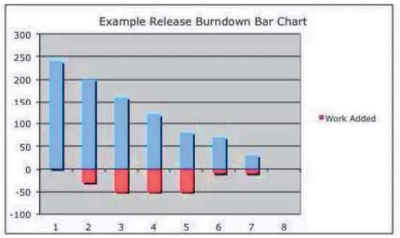


# Release-Charts

Release Burndown chart

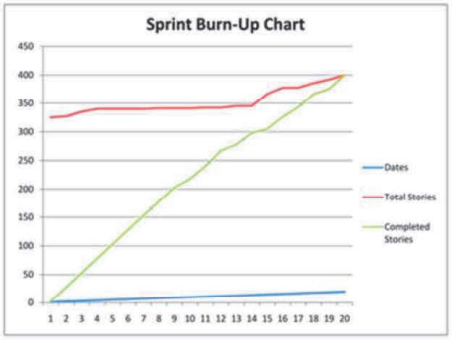


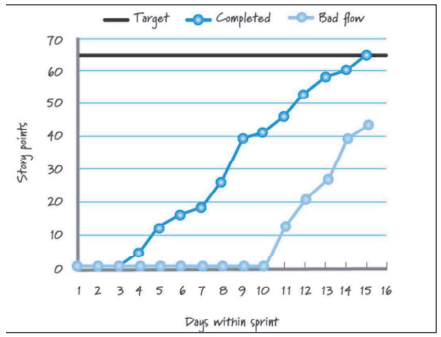

One dimensional view- Planned vs Actual






# Sprint burnup
# Story Points <> Days



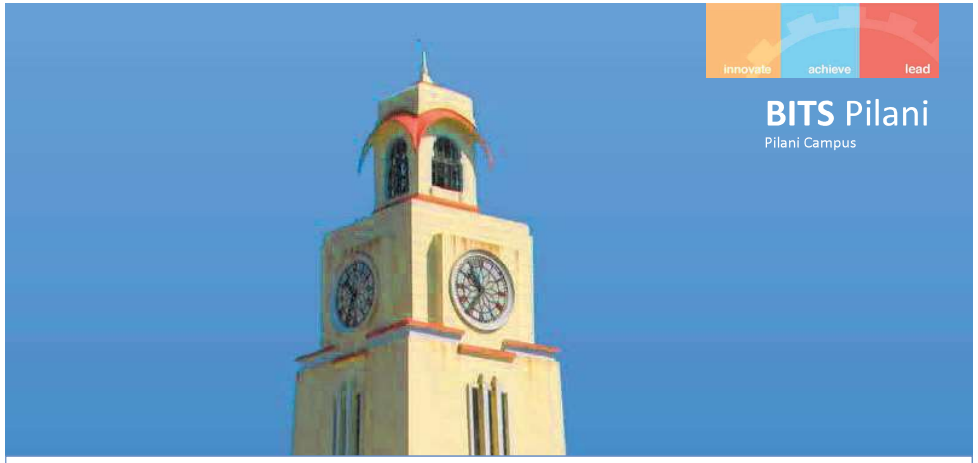




The one advantage of burnup charts over burndown charts is that it can depict change of scope.

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



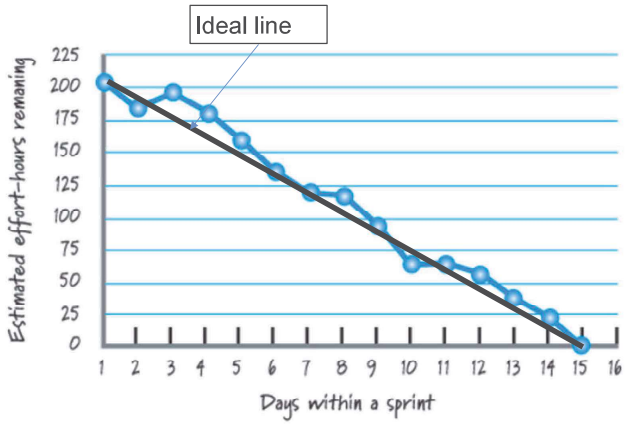
# Quiz




# Additional Notes



# Sprint burndown chart- Effort




X-Axis: Represents the days within a sprint.
Y-Axis: Remaining estimated effort-hours
Should be updated every day
•Charts shows the trend – likelihood of completing the work by end of the sprint

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

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



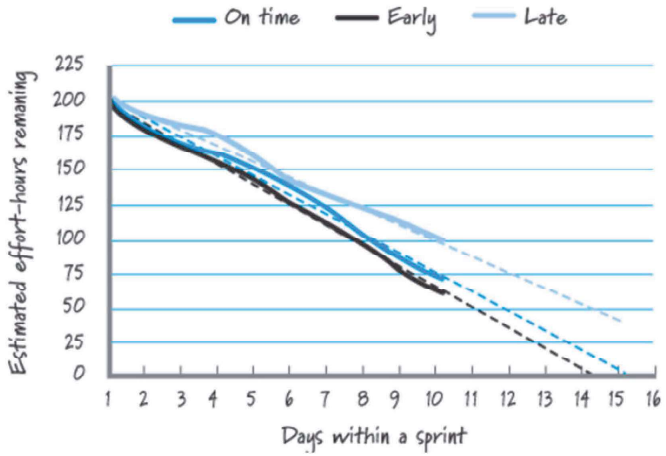
# Sprint Execution: The Process

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

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

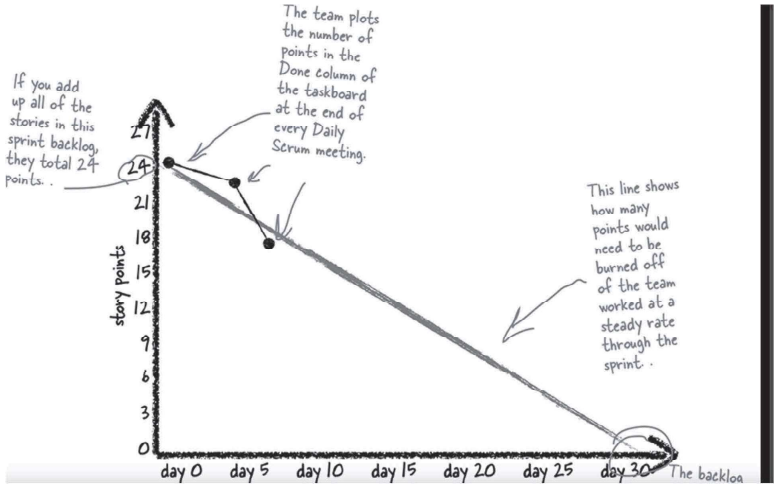


# Sprint burndown chart with trend lines




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

# Sprint burndown chart- Story points




Ref: Head First Agile by By Andrew Stellman and Jennifer Greene



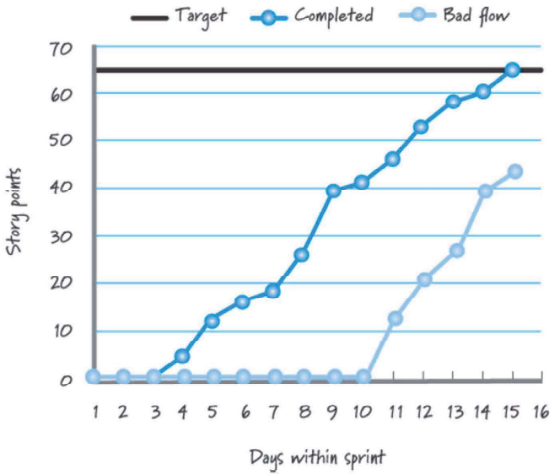
# Sprint Execution: Daily Scrum team meeting

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

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



# Sprint burnup chart




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

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



# Sprint Execution

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

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

# Sprint Execution: Timing

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

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



# Sprint execution: Process Task Planning

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



# Sprint Execution: Participants

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

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

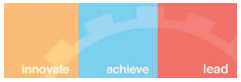


# Sprint execution: Process Flow Management

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

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

# Flow Management: Parallel Work and Swarming



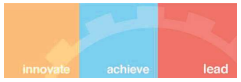

- The team must determine how many product backlog items to work on in parallel, in other words, at the same time. Working on too many items at once slows the team down. But working on too few items at once is equally wasteful. To find the proper balance, I recommend that teams work on the number of items that leverages but does not overburden the T-shaped skills and available capacity of the team members.

- The goal is to reduce the time required to complete individual items while maximizing the total value delivered during the sprint. Another name for this approach is swarming. Swarming is when team members with available capacity work together to complete one unfinished item rather than moving ahead to work on new items. This doesn't mean teams should always work on only one item at a time—the actual number of open items at any one time is highly dependent on context. Teams will have to experiment to find the balance that maximizes the value they deliver each sprint.

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



# Flow Management: How to Organize the work?




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

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



# Flow Management: Which Items to start


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

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



# Flow Management: What tasks need to done?


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

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

# Flow Management: Who does the work?

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



# Communicating: The Progress

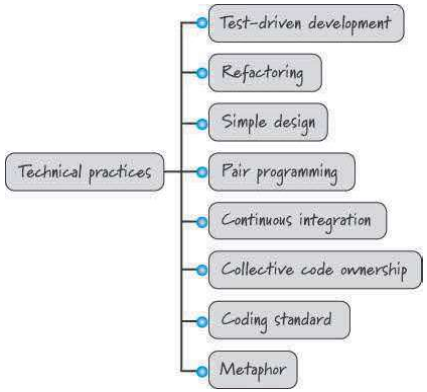
- Most teams use a combination of a task board and a burndown and/or burnup chart as their principal **information radiator.**

- **Information radiator:**
- A visual display that presents up-to-date, sufficiently detailed, and important information to passersby in an easy, self-interpretable format.



# Task Performance: Technical Practices

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




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



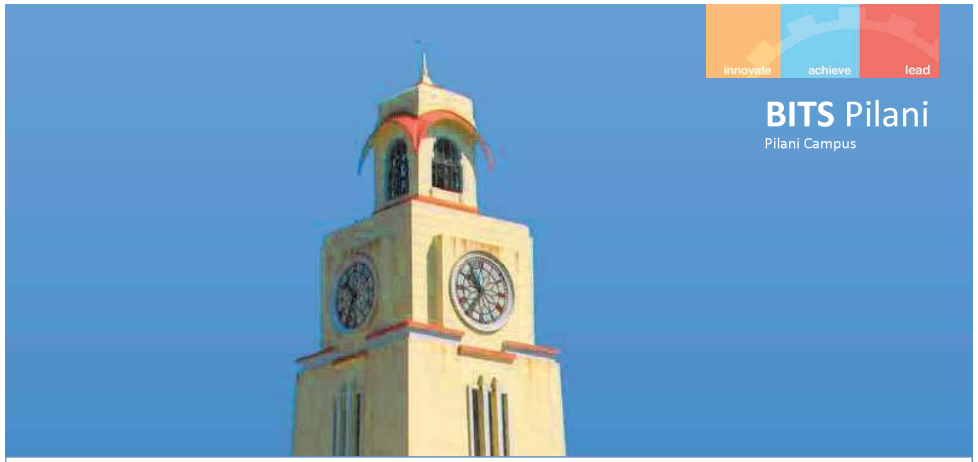
Thank you

# BITS Pilani presentation


**BITS** Pilani
Pilani Campus

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



**BITS** Pilani
Pilani Campus

# SE CZ 544 , Agile software processes – Agile Metrics and Tools



## Agile Metrics

- Examples:
  - Velocity, Lead time, Cycle time, Charts, Escape defects and so on.
- Helps to assess the quality of a product and track team performance.

- The Concept:
  - Define Metrics that can be used by Agile teams and Team management, Agile metrics that matter.
- The Opportunity
  - Reduced costs, Increase Product Quality, Increased team satisfaction
- The Potential
  - Auto Generate using exposed APIs provided by various PM tools.

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



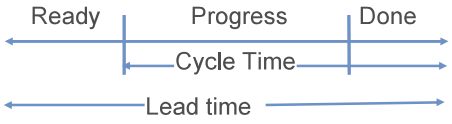
## Quantitative & Qualitative Metric

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

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

# An example Quantitative Metric

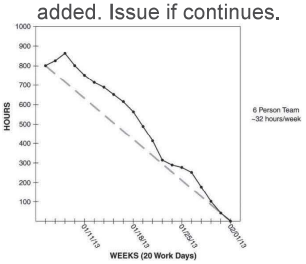
- Example: **Lead time** is a useful quantitative statistic for evaluating team performance.
- Determinant metrics:
  - A set of measurements related to a specific measurement
  - **Associated metrics:**
  - Flow efficiency (wait time)
  - Speeding tickets(%), ( Tickets moves through multiple statuses)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

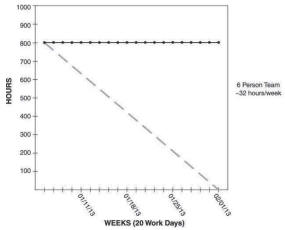





# Metrics: WHAT ARE SOME TRENDS OF BURNDOWN CHARTS AND WHAT DO THE PATTERNS INDICATE?

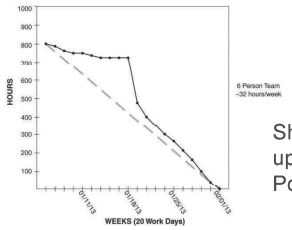
Uptick: New tasks/Stories added. Issue if continues.



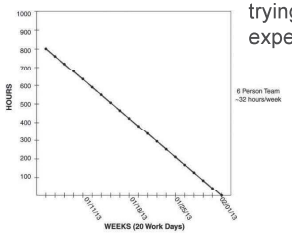




Flatline: Multiple reasons. Impediments, Task/Stories added at the same rate as work complete.

Perfect line: Team trying to align with expectations




Sharpdrop: Team not updating the chart/ Pointed removed






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

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

*Green*: It worked for the team.

*Orange*: Room for improvement.

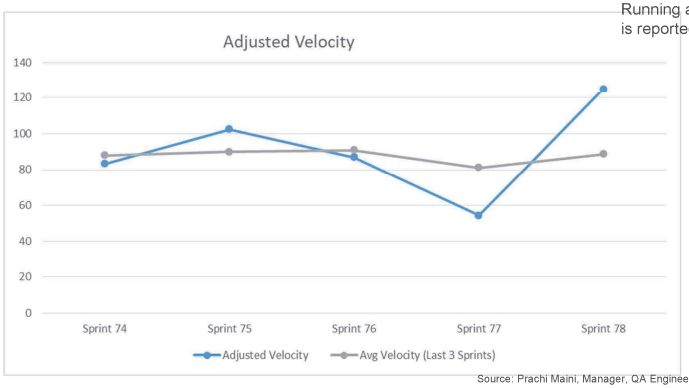
*Red*: Didn't apply or the practice is failing

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



# Velocity

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




**Velocity:** Points of work completed by an agile team within a given sprint

**Adjusted Velocity:**
Points of work completed by an agile team accounting for holidays, team absence, improvements etc..

Running average of last three sprints is reported.

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

# What to Watch for?

1. An erratic average velocity over a period of time requires revisiting the team's estimation practices.

2. Are there unforeseen challenges not accounted for when estimating the work

**DONOT**

- Use velocity to compare two different teams since the level of work estimation is different from team to team
- Use velocity to identify lower performing teams.

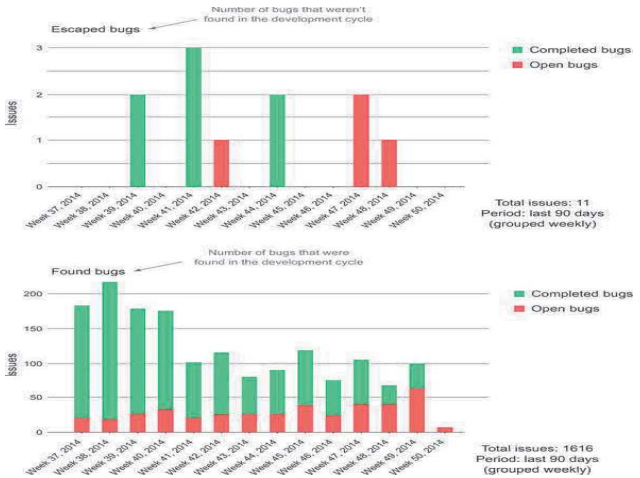


# Summary

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



# Bug counts

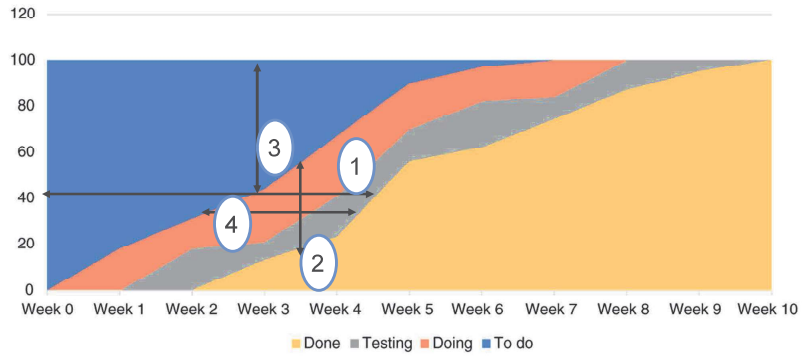



Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis , Published by Manning Publications, 2015



# Quiz

# Cumulative flow diagram
# Q1: What to Watch for?

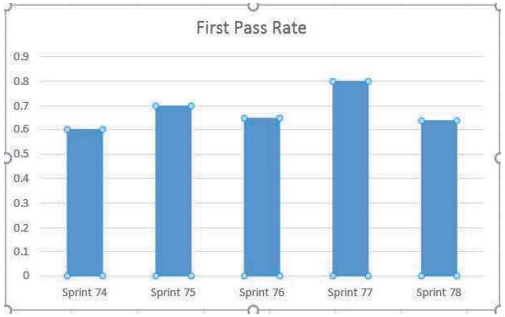



1- Lead time , 2.Work in Progress, 3. Backlog (week3), 4. Cycle time

## Q2

### Quality of Code

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

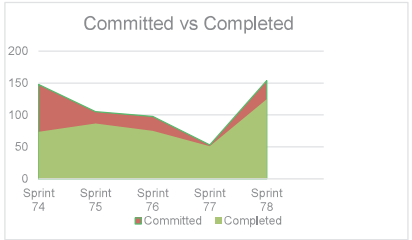


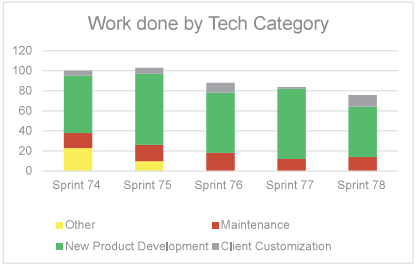

How do you connect this measure to production defects?



# Project Progress:
# Q1: What to Watch for?

Burndown chart



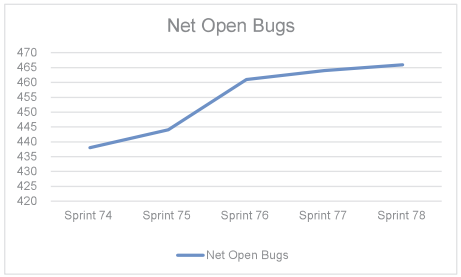

# What To Watch For

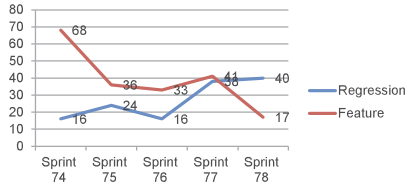
- Lower first pass rates indicate that Agile tools like desks checks , unit testing are not used sufficiently.
- Lower first pass rate could indicate lack of understanding of requirements.
- Higher first pass rate combined with high defect rate in production could indicate lack of proper QA.

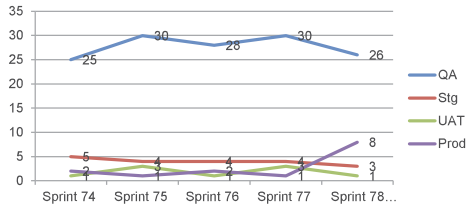
## Q3

### Bug Dashboard

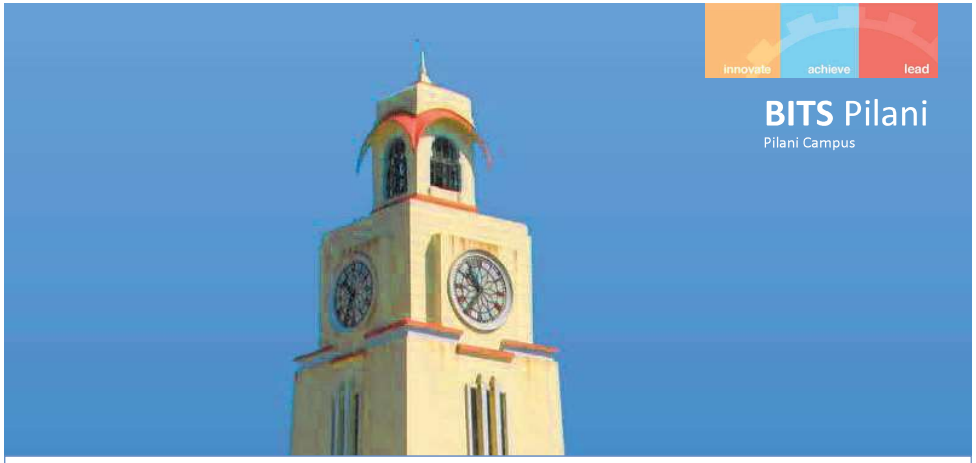
- **Net Open Bugs / Created vs Resolved**
  - This gives a view of the team flow rate. Are we creating more technical debt and defects than what the team can resolve.
- **Functional Vs Regression Bugs Trend**
  - This helps identify the defects found in new development vs regression.
- **Defects Detected in**
  - This helps identify the environment in which the defect is detected. (QA, Staging, UAT, Production)
  - For defects detected in environment higher than QA, an RCA is needed.






# Additional Notes - Measuring Agile Performance

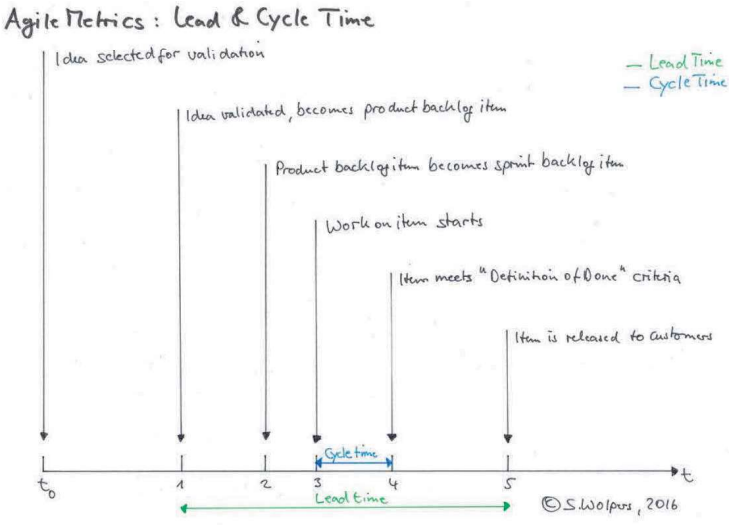


## What To Watch For

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



## Good Quantitative Agile Metrics: Lead Time and Cycle Time




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

## Most common and meaningful metrics for Team System Improvement(SI)

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

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



## Most common and meaningful metrics for Team SI …

| Simple SI metrics | Metric | Comment |
|---|---|---|
| **Team Wellness** | Team Happiness<br>Team Sprint Effectiveness Rating | Self Assessment tests: Individual engineers polled each Sprint/cycle. |

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



## Most common and meaningful metrics for Team SI …

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

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

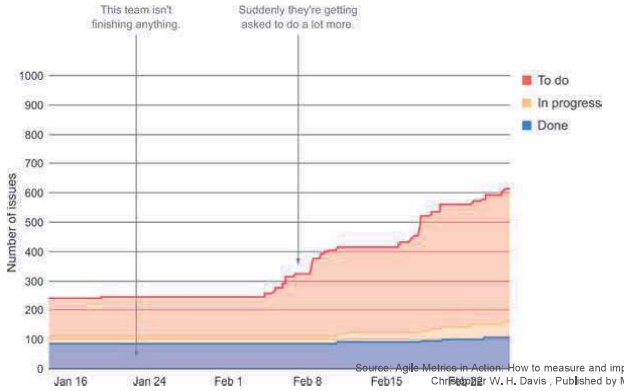


## The Importance of Metrics to Agile Teams

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

# Cumulative Flow

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

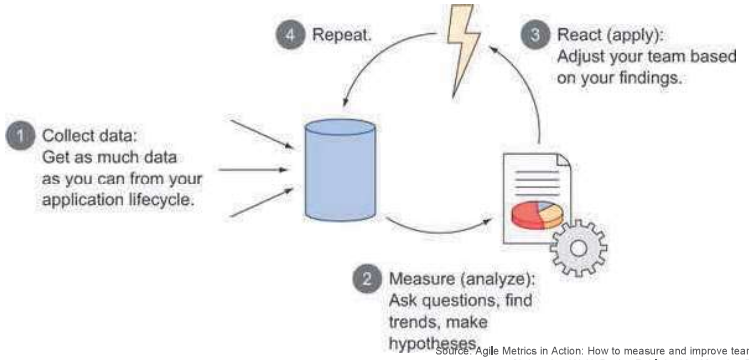



Source: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis , Published by Manning Publications, 2015



# COLLECT, MEASURE, REACT, REPEAT—THE FEEDBACK LOOP

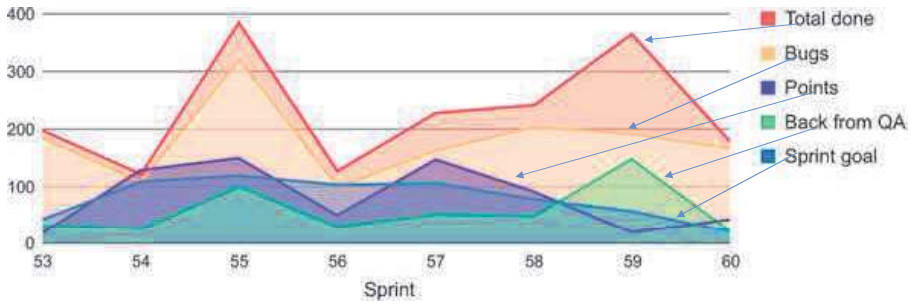
- There isn't a silver-bullet metric that will tell you if your agile teams are performing as well as they can.
- Collecting and analyzing data in the form of metrics is an objective way to learn more about your team and a way to measure any adjustments you decide to make to your team's behavior.




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



# Example - Combination of data



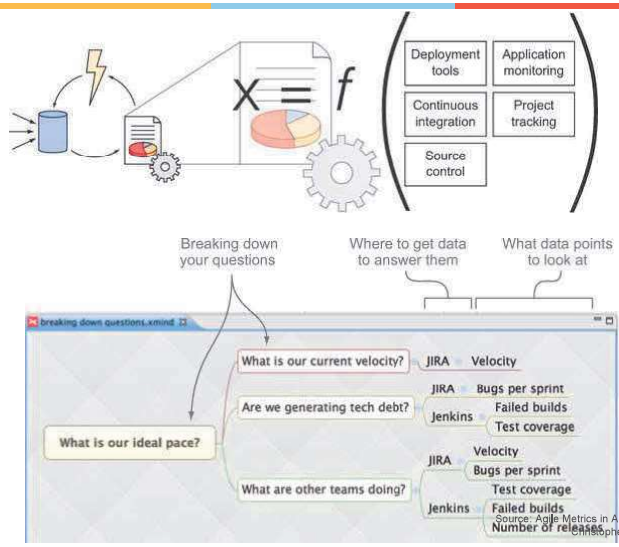

- **Spikes in this data point can indicate potential problems:**
- There's a communication gap somewhere on the team.
- Completion criteria (a.k.a. done) are not defined clearly to everyone on the team.
- Tasks are being rushed, usually due to pressure to hit a release date.

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



# Figuring out what matters

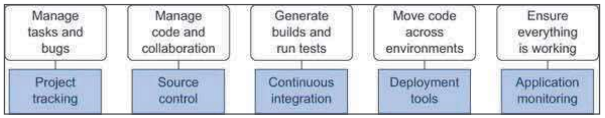
**(X is what you want to answer; some combination of your data can get you there.)**



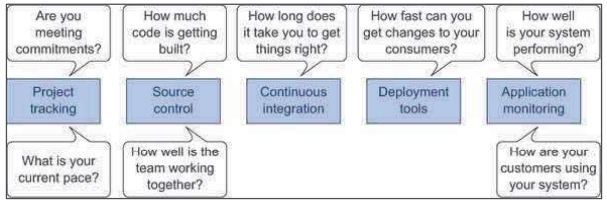

- Mind mapping is a brainstorming technique where you start with an idea and then keep deconstructing it until it's broken down into small elements. XMind (www.xmind.net/),

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

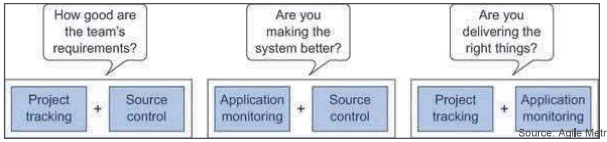
# Project performance data




Data is all over the place without a unified view




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




Adding data together to answer high-level questions

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



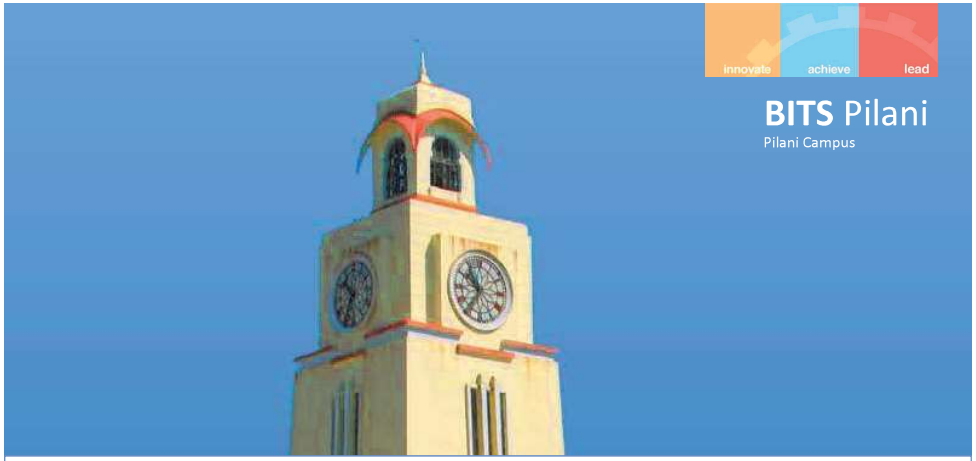
# BITS Pilani presentation

BITS Pilani
Pilani Campus
K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



# Thank you





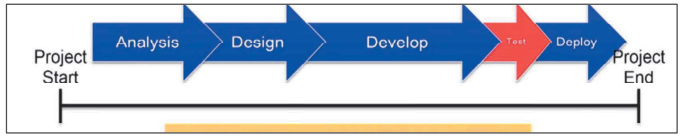
# Module-10 – Managing Quality and Risks in Agile Project

## Key Differences between Agile and Traditional Quality Management

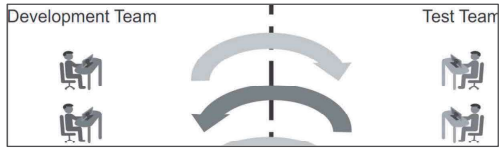
- Integration of Testing with Development
  - Concurrent vs Sequential
- Testing Approach
  - More reactive vs More Proactive
- Responsibility of Quality
  - Overall Team <> QA Team
- Regression testing
  - Frequent (Code Changes), At end after Code stabilizes



## Agile Approach to Quality




Water fall Project




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

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

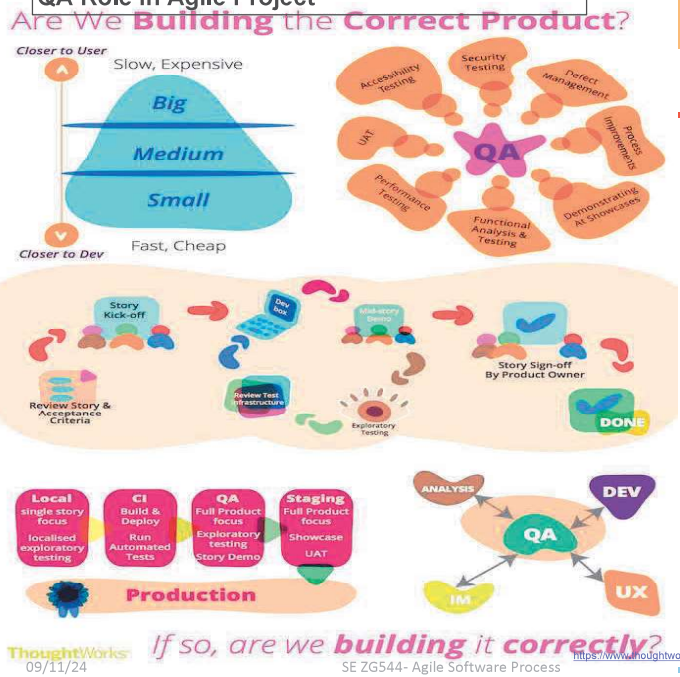


## Agile Development and Testing Practices

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



### QA Role in Agile Project

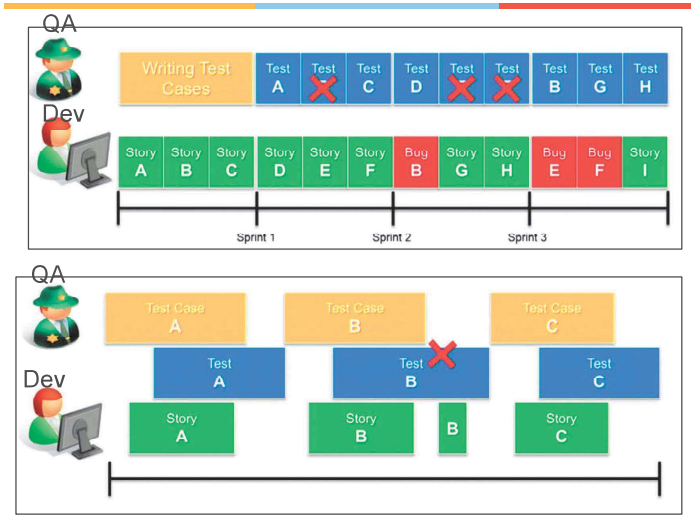



- Agile/Scrum: QA is involved in every aspects of Project/Product development cycle

QA has a unique mix of all these capabilities.
QA brings the mindset of **"Are we building the correct product and, if so, are we building it correctly?"**

https://www.thoughtworks.com/insights/blog/qa-role-what-it-really

# Testing within Sprint




- QA lagging behind in Testing
- Bugs Snowballing effect

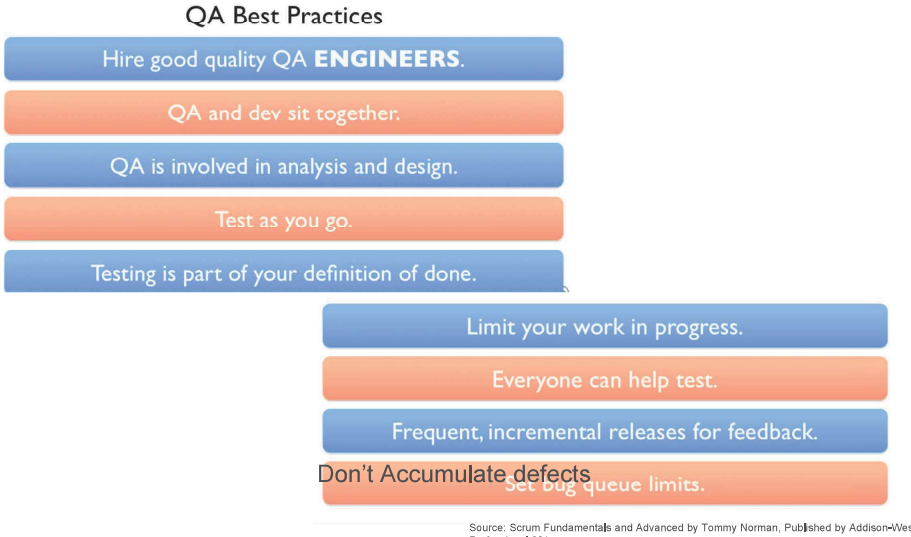
- Collaboratively testing with Dev.
- Fully tested Software
- Minimal Hands-off.

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

Source: Scrum Fundamentals and Advanced by Tommy Norman, Published by Addison-Wesley Professional 201



# QA good Practices




Source: Scrum Fundamentals and Advanced by Tommy Norman, Published by Addison-Wesley Professional 201



# Process Quality and Product Quality

**Process quality:** Metrics primarily focus on delivery speed or the effectiveness of feedback loops to make sure a team is responding appropriately to change.
- Examples: Velocity, Lead time, Cycle time, Team Happiness index

**Product Quality:** Metrics measure the excellence of a product and its features
- Examples: Bugs, Time to resolution, Uptime, Code coverage

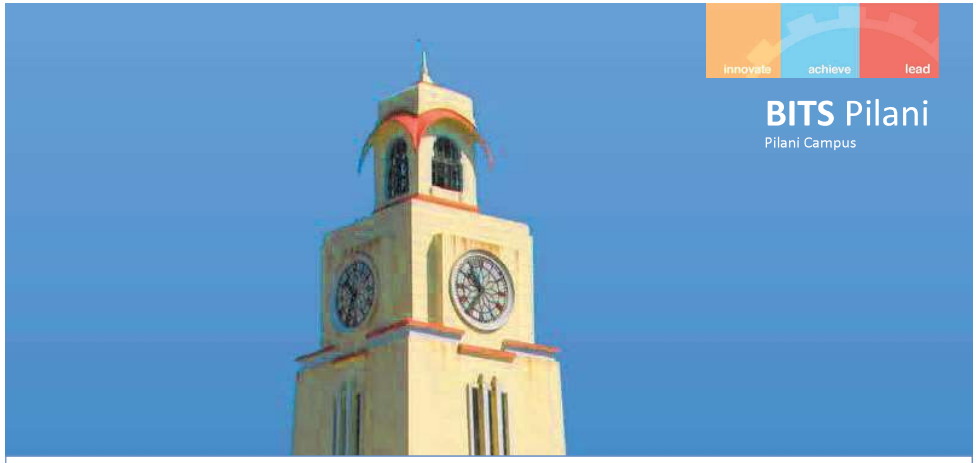
**Agile Practices:**
- Sprint planning, Sprint Review, Sprint retrospective, Whole team

- Testing, TDD, Pair programing, Refactoring, 10-minutes build



# Quizes

- Q1,Q2,Q3




# Risk Management in Agile



## Mitigation Strategies for positive risks or opportunities

**Exploit:**

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

**Share:**

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

**Enhance:**

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



## Risk management in Agile

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



## Mitigation Strategies for Negative Risks (Threats)

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

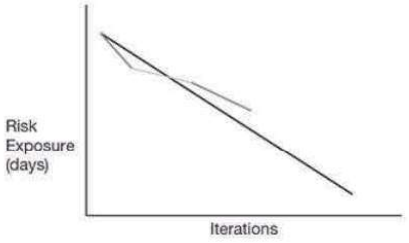
# Risk response and Risk Assesement matrix

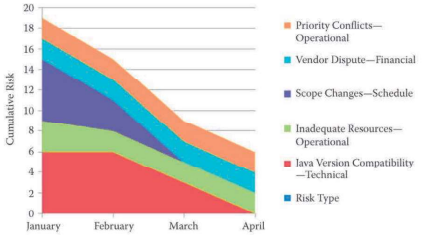
- **Risk response**
  - Adding resources
  - Cross Training
  - Skill Development
- Risk Assesement Matrix
  - Row : Impact (Low (1-2),Medium(3-5), High(6-10))
  - Qualitative assesement
  - Column: Risks (Schedule , Cost …)
  - For example: Schedule Variance(<2%,5% to 9%, =>10%)





# Risk burndown chart




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



# Communicating - Risk Register – An Example

Impact

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

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

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



# Use Risk Management to Make Solid Commitments to executives

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

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

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

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

# Using risk multiplier in your estimation

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

You would calculate the range of possibilities as:

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



# Summary

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



# An Example

Suppose, estimated number of sprints is 10 for a release
Product Backlog at end of Sprint 5 - F1,F2,F3,F4,F5,F6
Assume each feature size = 20, Velocity = 20
Total size of the remaining features = 6*20= 120 points
Sprint Remaining = 5, Risk Multiplier = 1,1,4,1.8

**6th Sprint Commitments:**

10% Chance : Sprint remaining * Velocity - 5*20/1 = 100 points
  - 10% chance of delivering F1,F2,F3,F4,F5 (100 points)

50% Chance : 5*20/1.4 = 71.4 points
  - 50% chance of delivering F1,F2,F3 (60 points), Stretch = F4

90% Chance : 5*20/1.8 = 55,5 points
  - 90% chance of delivering F1,F2 (40 points), Stretch = F3

- Repeat this process after completing Sprint6



You are faced with three investment options, each requiring an upfront cost of $20,000. The success probabilities and potential returns for each investment are as follows:

Option X: There is a 70% chance of success, which would result in a return of $50,000.
Option Y: There is an 85% chance of success, with a return of $30,000.
Option Z: There is a 60% chance of success, leading to a return of $70,000.

To assess these investment options while considering the impact of uncertainty, calculate the Expected Monetary Value (EMV) for each option. EMV is calculated as the probability of success multiplied by the potential return. Based on your EMV calculations, which investment option appears to be the most economically favorable, taking into account both the likelihood of success and potential returns?

To determine which investment option is better, you can calculate the Expected Monetary Value (EMV) for each option. EMV considers both the probability of success and the potential returns.
Using the information provided:

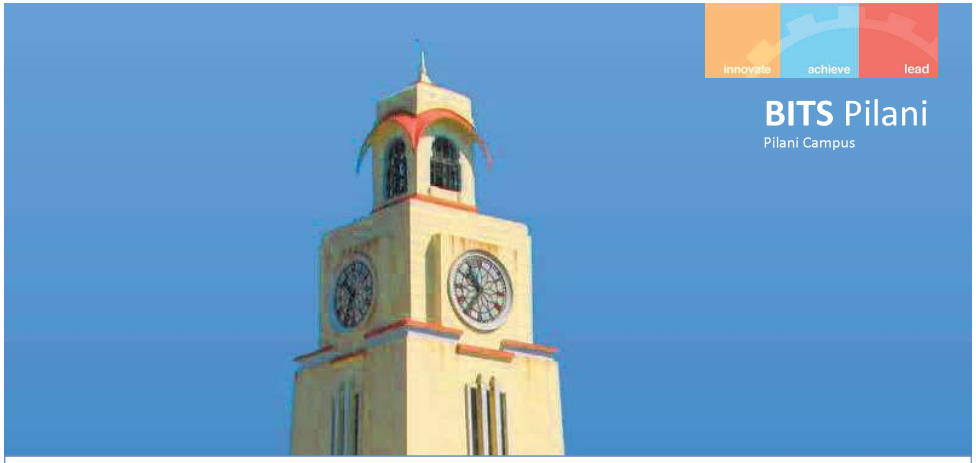
**Option X:**
•Probability of success (P) = 70% or 0.70
•Potential return (R) = $50,000
EMV for Option X = P x R = 0.70 x $50,000 = $35,000

**Option Y:**
•Probability of success (P) = 85% or 0.85
•Potential return (R) = $30,000
EMV for Option Y = P x R = 0.85 x $30,000 = $25,500

**Option Z:**
•Probability of success (P) = 60% or 0.60
•Potential return (R) = $70,000
EMV for Option Z = P x R = 0.60 x $70,000 = $42,000

Comparing the EMVs:
•Option Z has the highest EMV of $42,000.
•Option X follows with an EMV of $35,000.
•Option Y has the lowest EMV of $25,500.






# Additional notes – Quality & Risk management



## Quizes

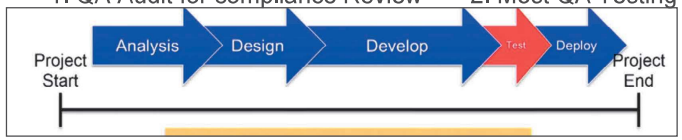
- Q4,Q5,Q6,Q7,Q8



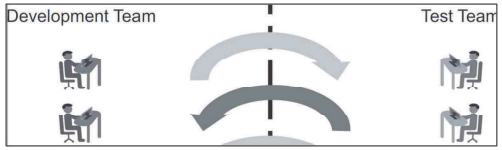
## Issues with Traditional Approaches to Quality Management

**Traditional Approach to QA**

1. QA Audit for compliance Review 2. Most QA Testing happens at the end




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




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

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

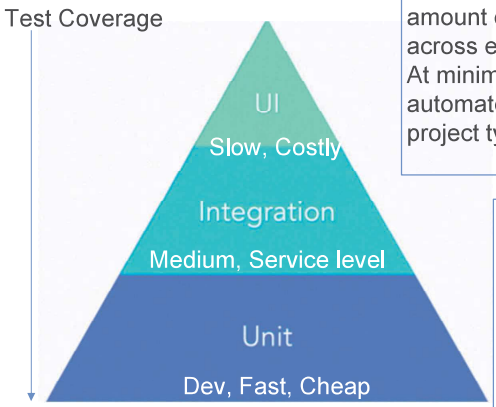
# Agile Approach to Quality Management

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



# Agile Approach to Quality Management …. Automate Tests at Different Levels

- Automation Pyramid




Visual representation of the recommended amount of test coverage that should exist across each type of test.
At minimum we should have three type of automated tests. Depending upon the project type we can have more type of tests

**Uniit Test:** Isolated tests , test functions, Fast, Need greater number of tests.
**Integration tests:** Slower, tests interfaces, databases, file system, other applications.
**UI Tests:** Tests end-end work flow. Much slower.



# Agile Approach to Quality Management ….

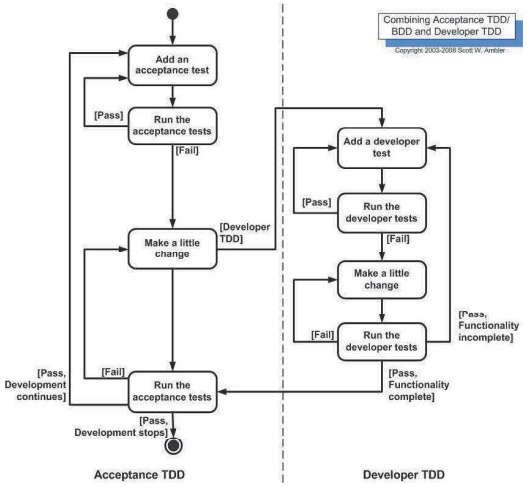
- The hand-offs between programmers and testers (if they exist at all) will be so small as not to be noticeable.
  - Team work, Doing a little of everything (designing, coding, testing, and so on) all the time helps teams work together.
  - Tester creates automated tests and the programmer programs.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



# The Role of Manual Testing

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

# Do Acceptance Test Driven Development




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



# Mitigating Risks with Agile Methods

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



# What is Risk?

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

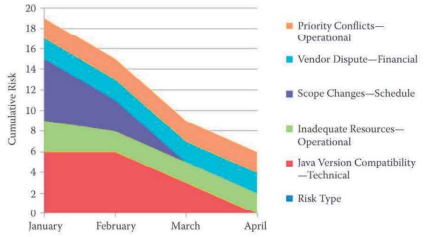


# Mitigating Risks with Agile Methods ….

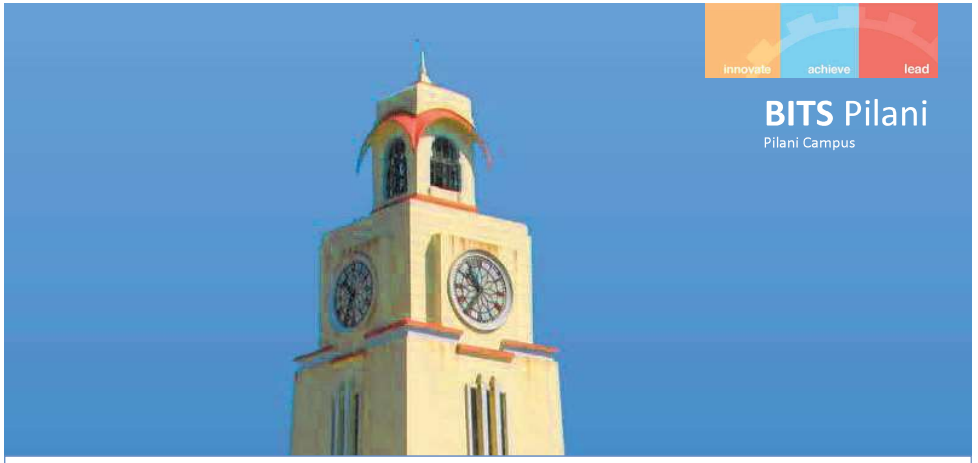
- Agile team ownership supports reduced estimation risk.
  - When the agile team takes responsibility for estimates of backlog items, this leads to increased accuracy of the estimates that they provide which in turn results in the timely delivery of the product.
- Transparency is a risk reducer of undetected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

# Risk Register- Another example

| Risk | Type | Impact (0–3) | Probability (0–3) | Severity = Impact × Probability |
|---|---|---|---|---|
| 1. Java version compatibility | Technical | 3 | 2 | 6 |
| 2. Inadequate resources | Operational | 3 | 2 | 6 |
| 3. Scope changes | Schedule | 3 | 2 | 6 |
| 4. Vendor dispute | Financial | 2 | 1 | 2 |
| 5. Priority conflicts | Operational | 2 | 1 | 2 |
| | | | | — |
| | | | | 22 |






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




# BITS Pilani presentation

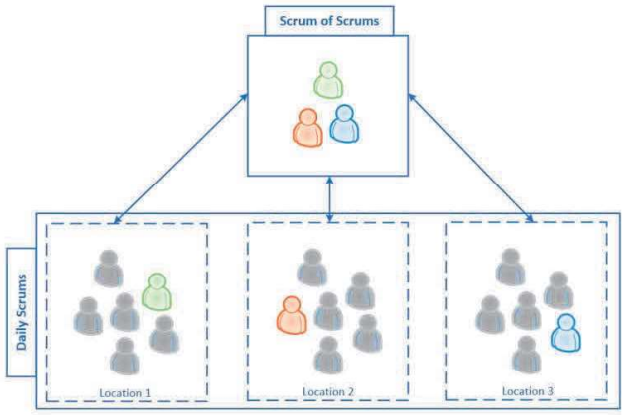
K.Anantharaman
kanantharaman@wilp.bits-pilani.ac.in



# Agile Myths/Misconceptions

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

# Scaling Scrum: Scrum of Scrums




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

Source: Product Management Journal Vol.7.



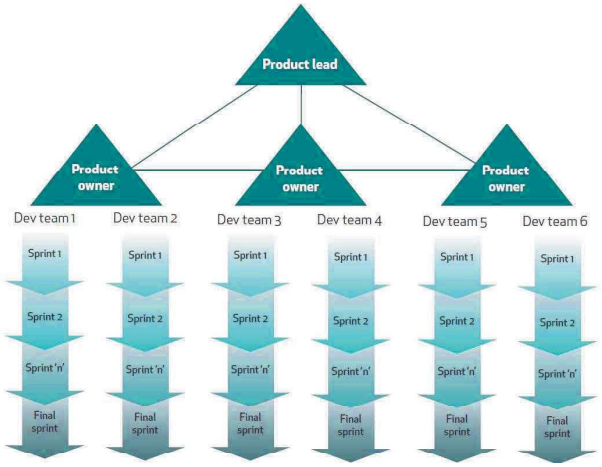
# Distributed Agile Models

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

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



# Scaling Scrum/ Scrum of Scrum/ SAFe




- Scrum is great for standalone development projects where goals are changing and the scale can be handled by a single team.
- Challenges for large projects, multiple teams, different locations.
- Drive hybrid development as shown in the diagram such Scaled Agile Frame work (SAFe).

Source: Product Management Journal Vol.7.



# Key issues with Distributed Model

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

# Distributed Agile team best practices

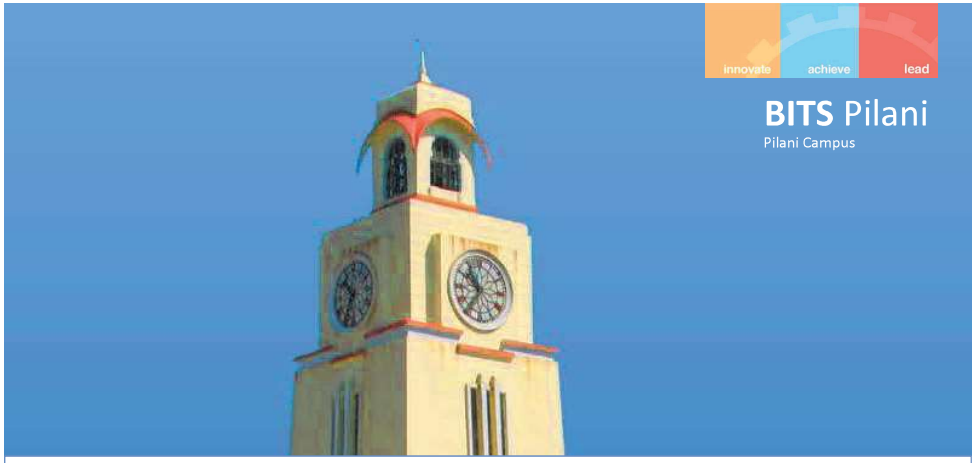
Best practices reinforce each other to mitigate risks

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

© ThoughtWorks 2008

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

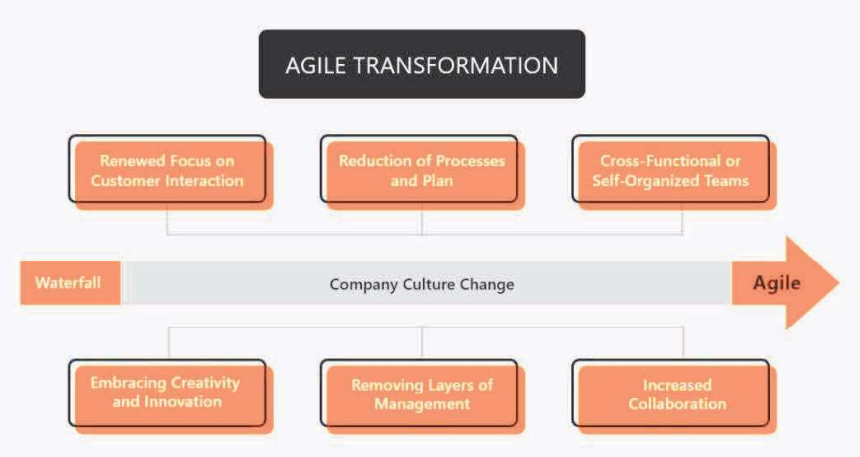





# Additional Notes



# Agile Transformation




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



# Distributed Agile/Location Independent Agile teams

- Covid-19 sudden disruption and impact
- https://www.mckinsey.com/business-functions/organization/our-insights/revisiting-agile-teams-after-an-abrupt-shift-to-remote
- TCS Perspectives:

➢ First: Assess the Organization:

- What is the level of business expertise and other skills required, and to what extent do they exist at a specific location?
  - If a location lacks business expertise, it will require more of it to be able to support agile teams there.
- How urgent and volatile is the work?
  - Location-independent agile teams should focus on work that is neither urgent nor volatile. If the work is both, if it has non-negotiable constraints (such as overnight fixes, intra-day scope changes, or regulatory requirements), or if there is a need for constant access to the project owner, it's best to work with teams in the same location, if at all possible.

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

# Distributed Agile/Location Independent Agile teams …

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

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



# Agile Transformation

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



# Distributed Agile Models

Location-Independent Agile Model

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

**Local - Model-1** ; This model is best when the teams are new to the business area, when continuous access to a product owner is paramount, or when regulatory concerns require the project to be executed in a specific geography.
**Minimally Distributed (Model 2):**The product owner and a few members of a project or program are located together as one team, while the rest of them work together as another inter-related team in a different place. This model requires the teams to understand the underlying business processes for their product.
**Significantly Distributed (Model 3):**Team has shared understanding of the business processes of a project or program are well positioned to adopt significantly distributed agile work processes.
**Fully Distributed (Model 4):**The product owner may be at any site, while the rest of the project or program team members are grouped in agile teams across distributed locations. These teams each include product specialists with sufficient business knowledge to drive day-to- day decisions within a framework defined by the product owner.

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



# Scrum Meeting time Guidelines

| | | SPRINT DURATION IN WEEKS | | | |
|---|---|---|---|---|---|
| MEETING | GUIDELINE | 1 | 2 | 3 | 4 |
| SPRINT PLANNING | 2 HOURS/WK | 2 | 4 | 6 | 8 |
| DAILY SCRUM | 15 MINUTES/DAY | 1.25 | 2.5 | 3.75 | 5 |
| SPRINT REVIEW | 1 HOUR/WK | 1 | 2 | 3 | 4 |
| SPRINT RETROSPECTIVE | 1 HOUR/WK | 1 | 2 | 3 | 4 |
| | MEETING TIME | 5.25 | 10.5 | 15.75 | 21 |
| | TOTAL TIME | 40 | 80 | 120 | 160 |
| | % | 13% | 13% | 13% | 13% |

Kanban differs from Scrum in several ways:

**Flexibility:** Kanban allows for more flexibility in workflow management, as there are no fixed iterations or predefined roles.

**Continuous Delivery**: Kanban emphasizes delivering value continuously, with work items being pulled as capacity allows, rather than waiting for the end of a sprint.

**Focus on Flow:** Kanban focuses on optimizing the flow of work through the system, whereas Scrum focuses on delivering potentially shippable increments at the end of each sprint