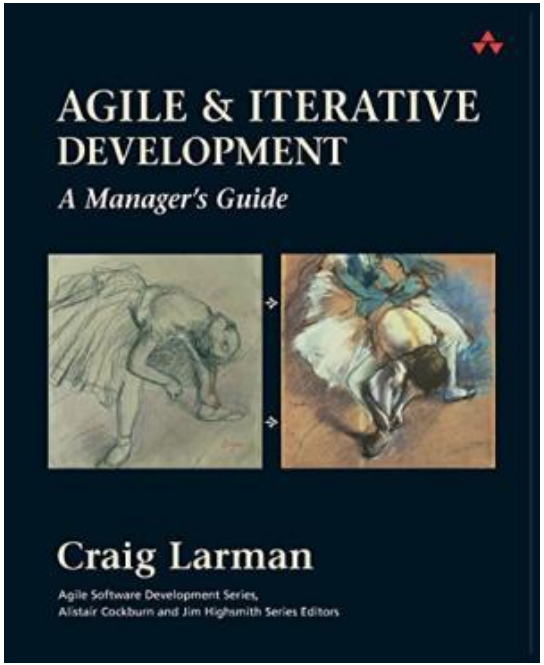
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Methods – An Introduction

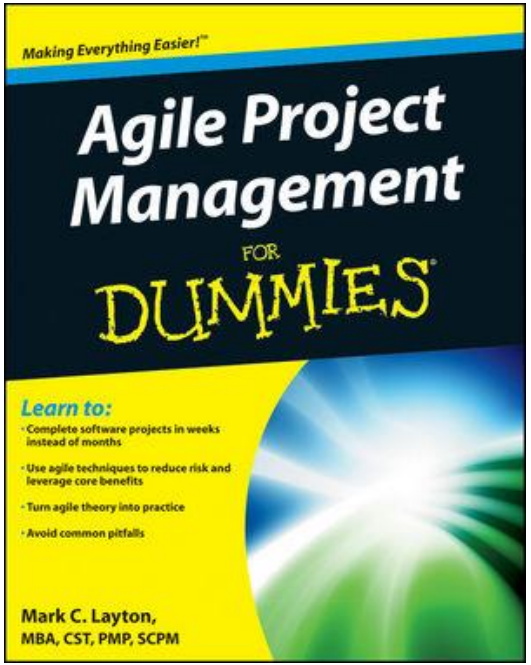
- Prof K G Krishna

Text/Reference Books

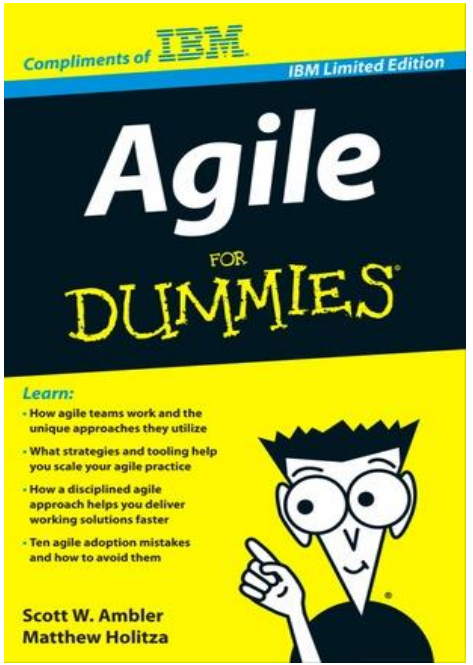
T1



T2



Compliments of IBM



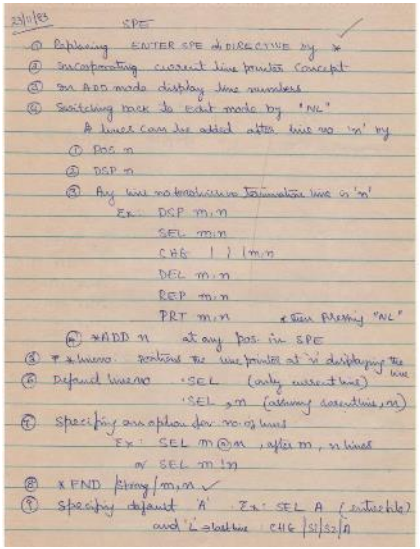
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

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

Traditional Software Development – The Backstory

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

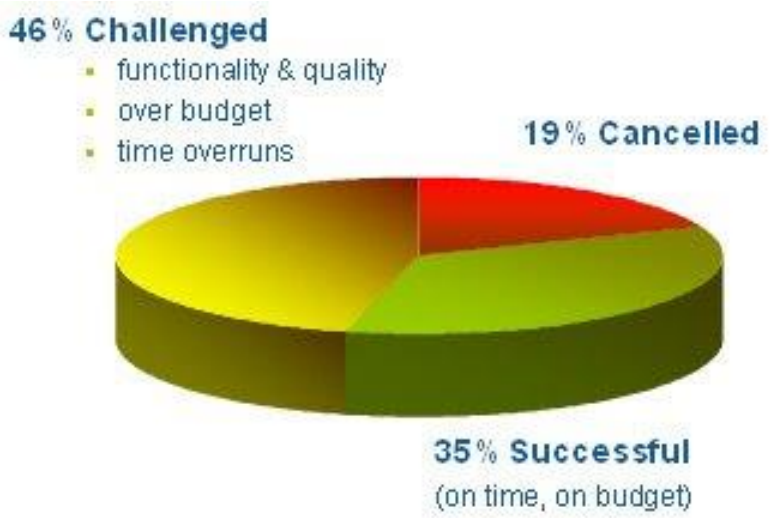


K G Krishna

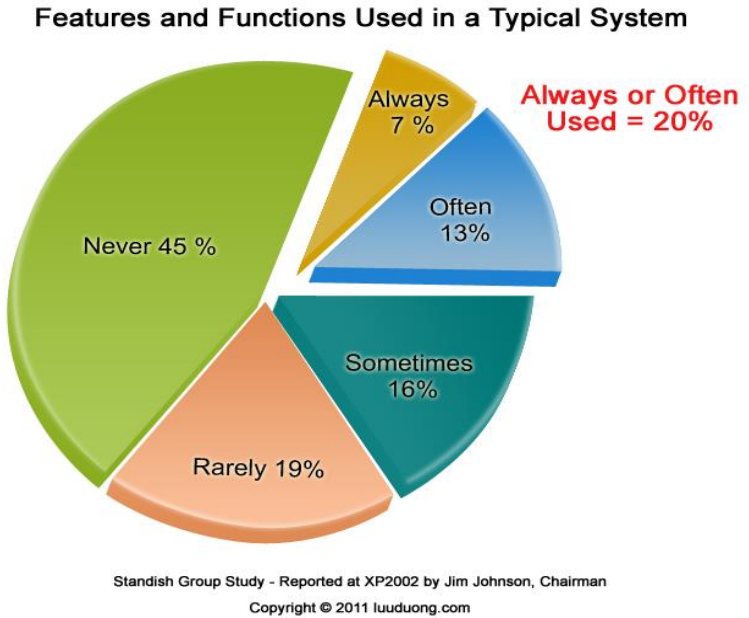
K G Krishna

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

Traditional Software Projects (circa 1990 ~ 2000)



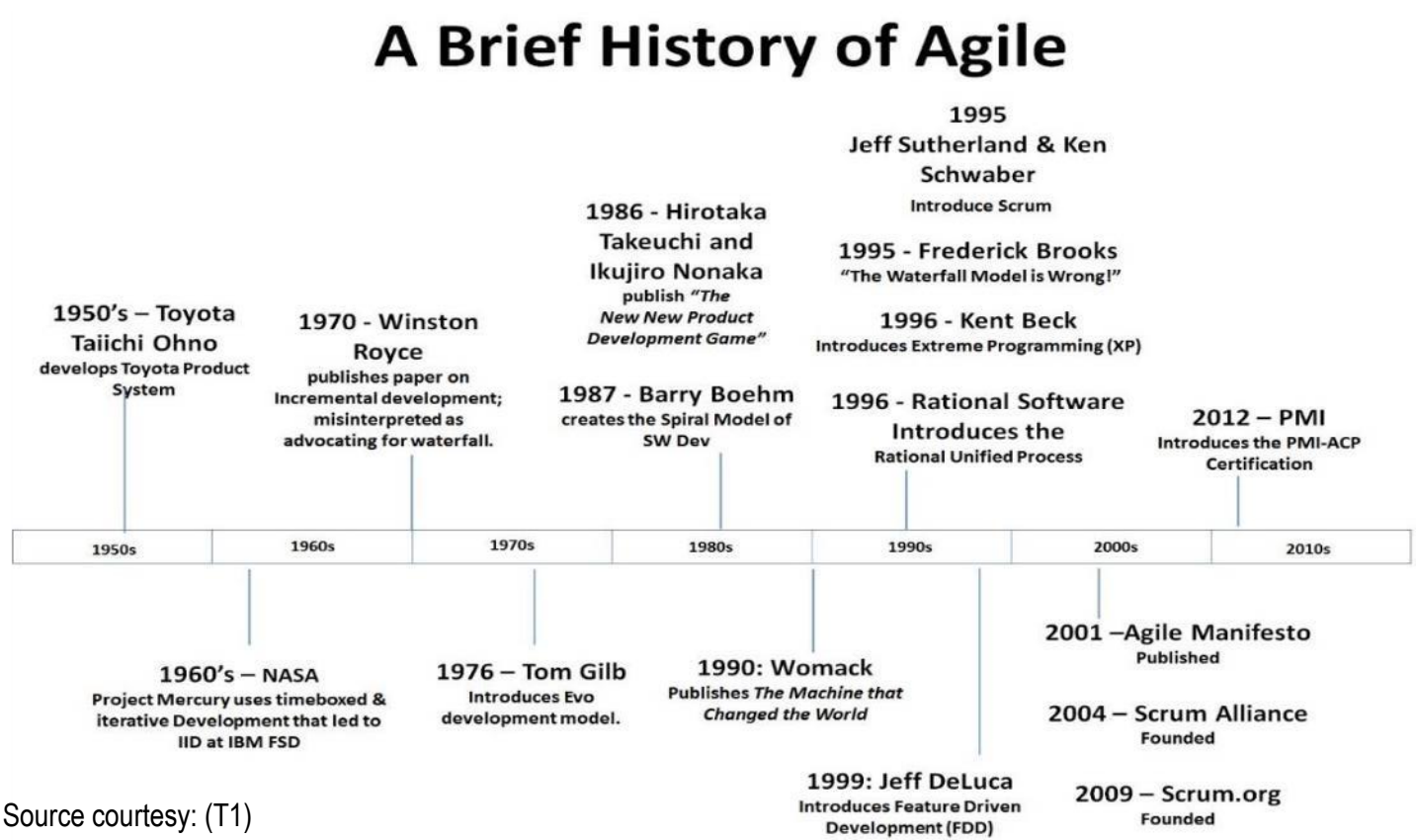
Source: Standish Group Chaos Report [2006]



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

Agile Turns Upside-down Project Constraints

Production-Line ➔ Agile System – An Evolution



Source courtesy: (T1)

Being Agile Means…

agile
/ˈadʒʌɪl/

adjective

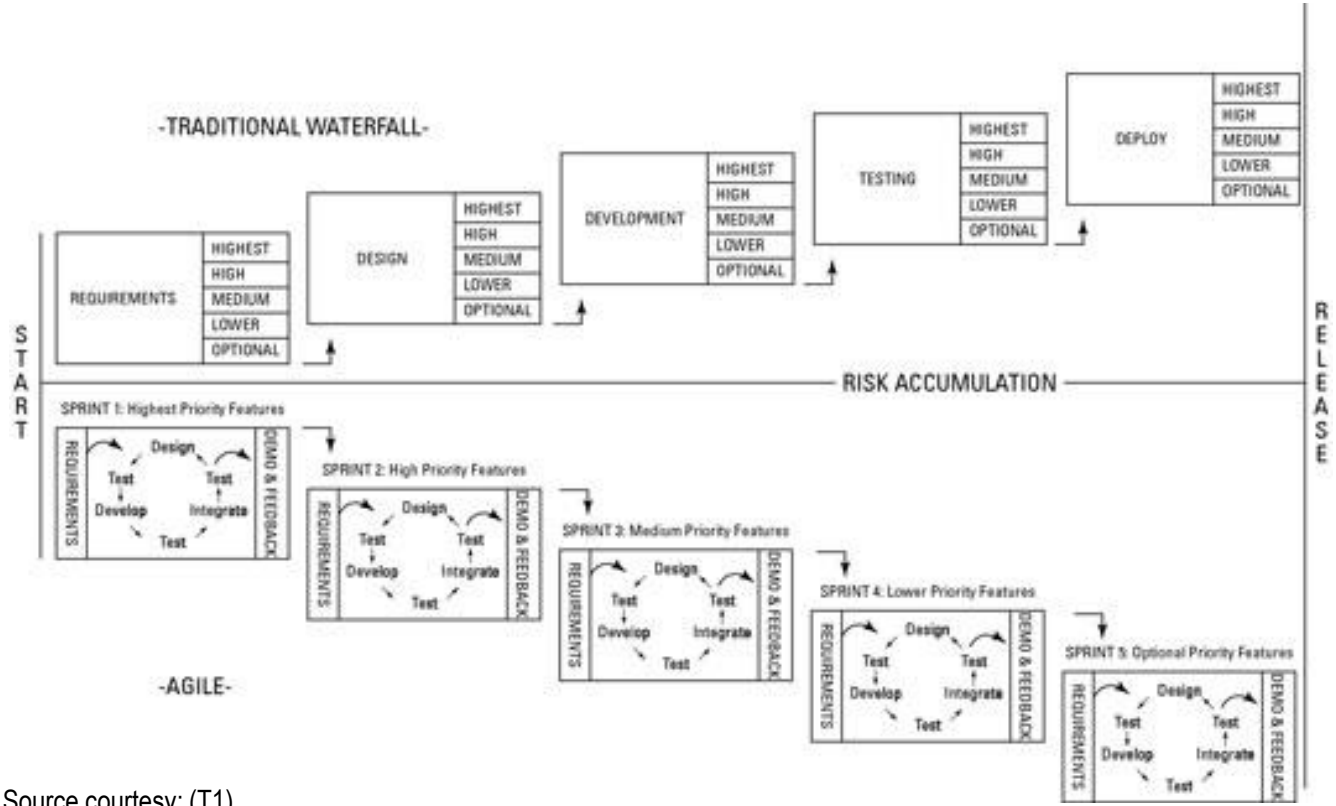
1. able to move quickly and easily.
   "Ruth was as agile as a monkey"
   synonyms: nimble, lithe, spry, supple, limber, sprightly, acrobatic, dexterous, deft, willowy, graceful, light-footed, nimble-footed, light on one's feet, fleet-footed; More

- Agility

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

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

Agile Development – A Series of Short Sprints



Source courtesy: (T1)

Agile Development = Iterative Releases!

- **Agile software development** is a conceptual framework for software engineering that promotes development **iterations** throughout the life-cycle of the project.

- Software developed during one unit of time is referred to as an iteration (**sprint**), which may last from one to four weeks.

- Agile methods also emphasize **working software** as the primary measure of progress

Need for Agile Methods ➔

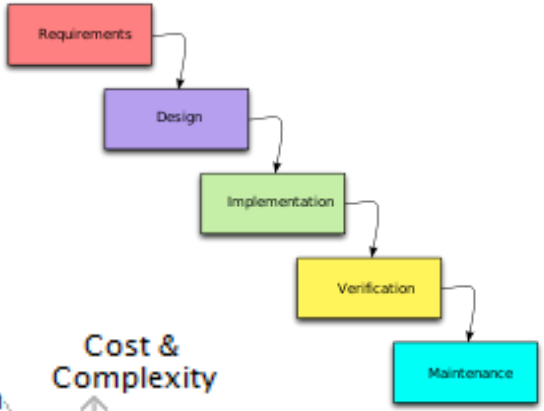
Software Systems Today…

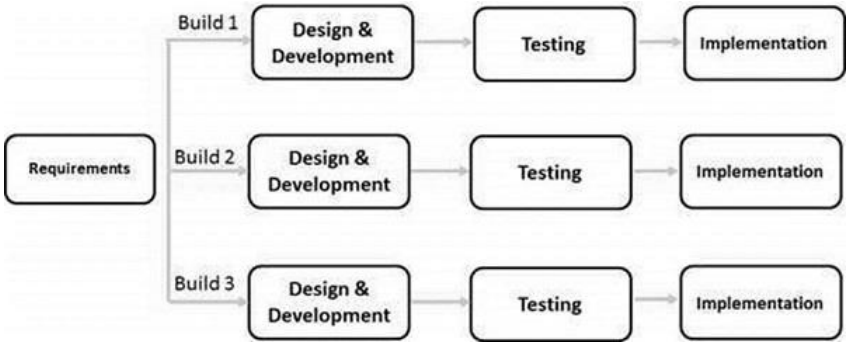
- Changing (‘Unclear’) Requirements / Customer Demands

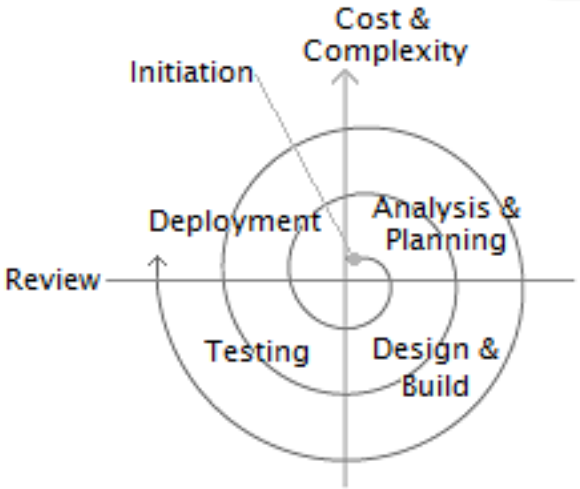
  “Requirements will be clear only at the end of the Project”
- Shrinking Development Cycles (Time-to-Market Releases)
- Entrepreneurial, Innovative Apps
- Shorter Shelf-life

Addressing Challenges of *Waterfall* Model

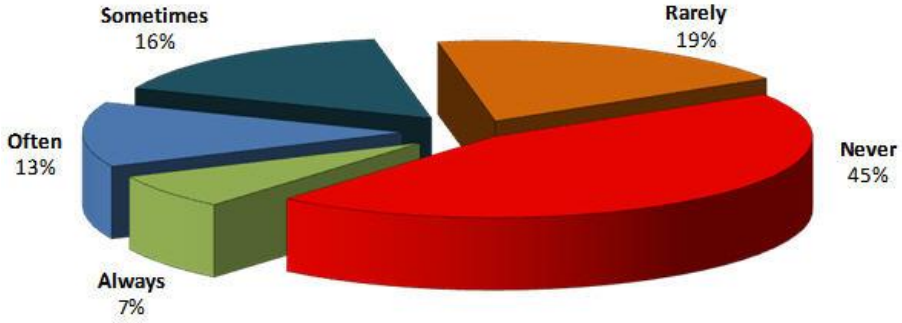
- Traditional Waterfall ➔ Adaptations of Waterfall (Incremental, Iterative Prototyping,…)
  - Focus on Requirements Management (Breaking down into manageable ‘Increments’)
  - De-risking Large Development Effort
  - Capturing Requirements (‘show-and-tell’, ‘prototype’ as basis of discussion around Requirements)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







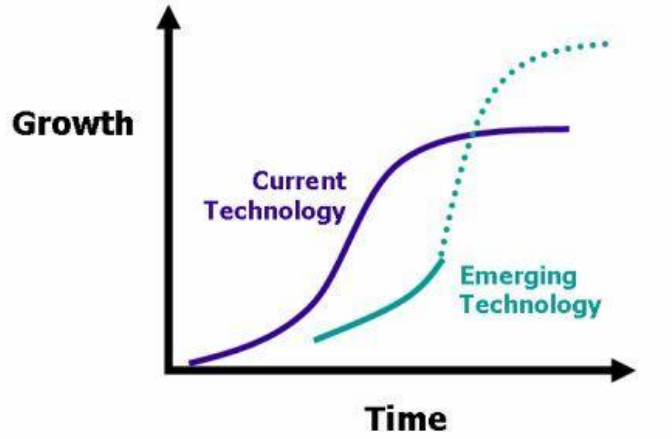
Software Feature Overload



Source: Standish Group Study, 2002

Pace of Technology *vs* Project Duration

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



De-risking Future Investments Upfront…

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

Benefits of Agile Methods ➔

Controlled Development

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

Agile Development ➔ Agile Project Management

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

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

Benefits of Agile Project Management

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

Agile Methods - Summary

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

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

Thank You

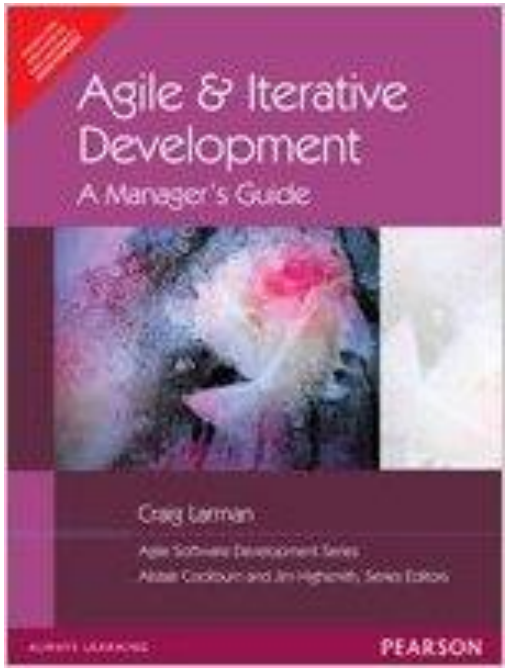
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Software Development

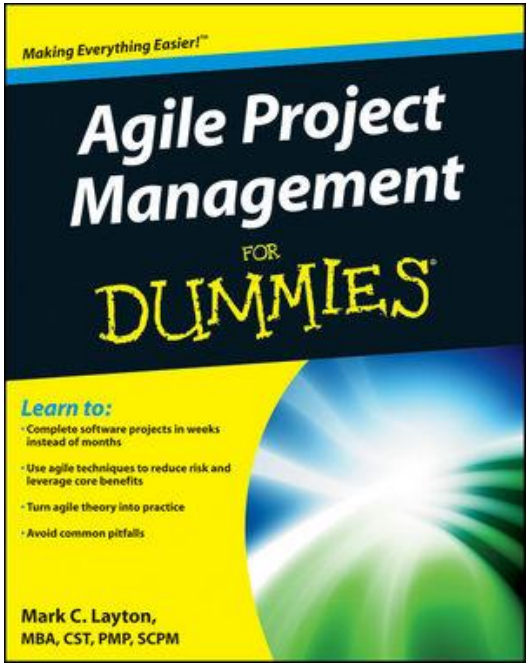
- Prof K G Krishna

Text/Reference Books

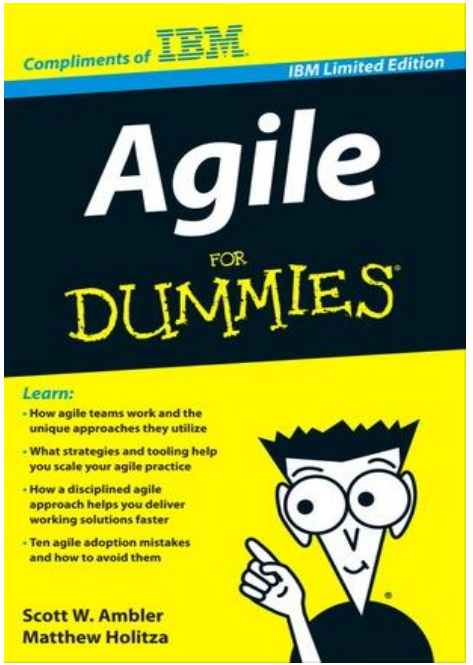
T1



T2



Compliments of IBM



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

Topics

Basics of Agile Software Development

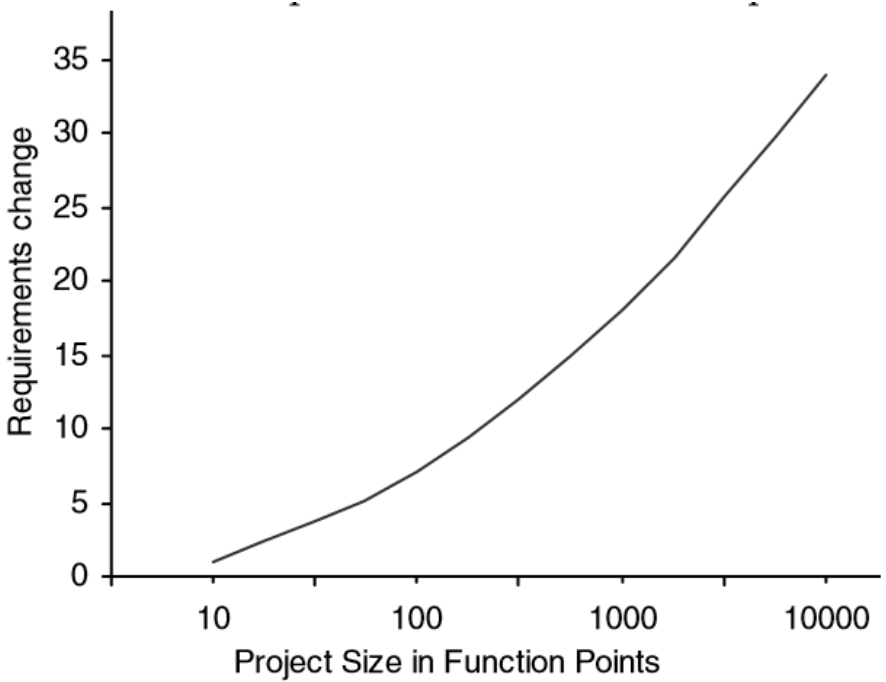
- Iterative and Incremental Approaches
- Risk driven and client driven development
- Time-boxed development
- Adaptive and Evolutionary development

Customer Requirements? Hard To Get!

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

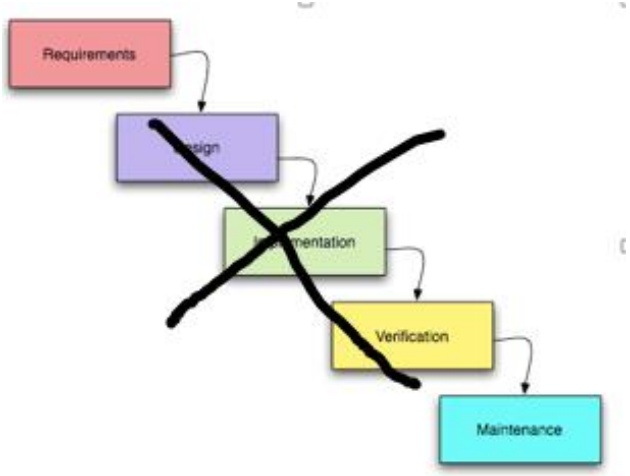
- “Requirements are capabilities and conditions to which the system—and more broadly, the project—must conform”
- “A prime challenge of requirements analysis is to find, communicate, and remember (that usually means write down) what is really needed, in a form that clearly speaks to the client and development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

Waterfall is nightmare…Don’t Go For It!

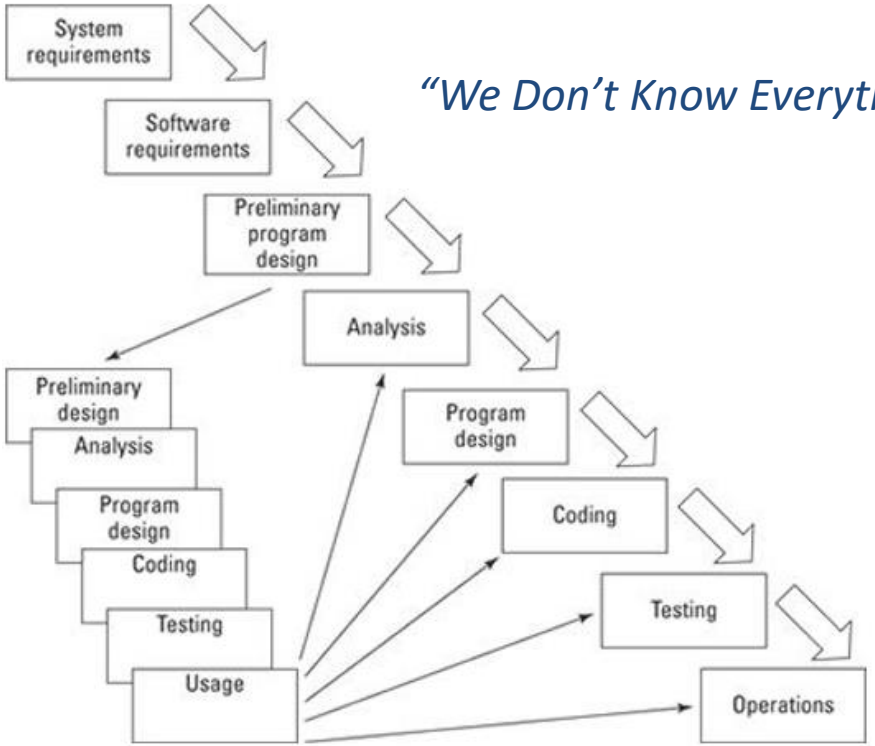


The ills of Waterfall…

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



Iterations in Waterfall? No Good Either…

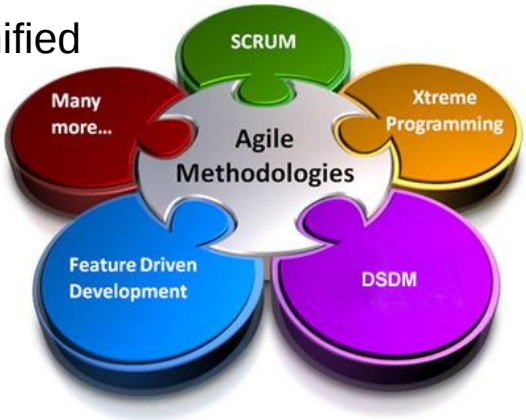


"We Don't Know Everything When We Start The Project…"

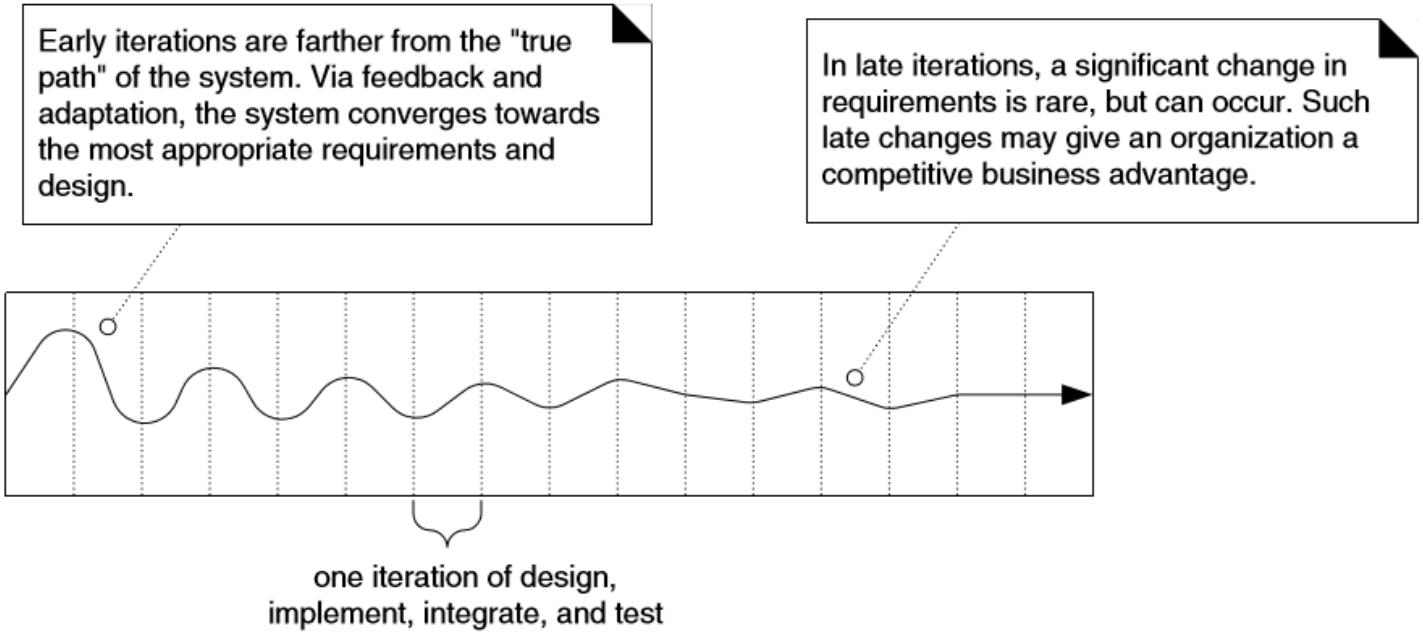
Iterative & Incremental...is the Way To Go!

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

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



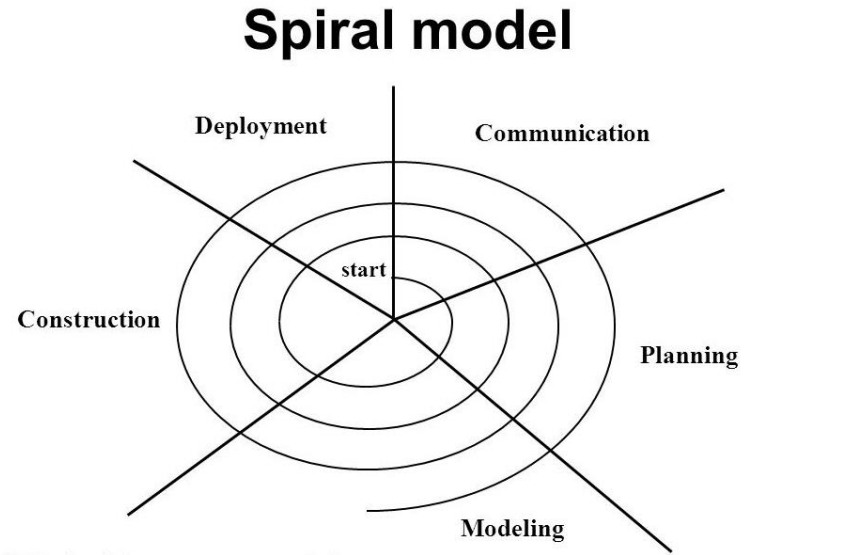
Iterations *Converge* to True Requirements!



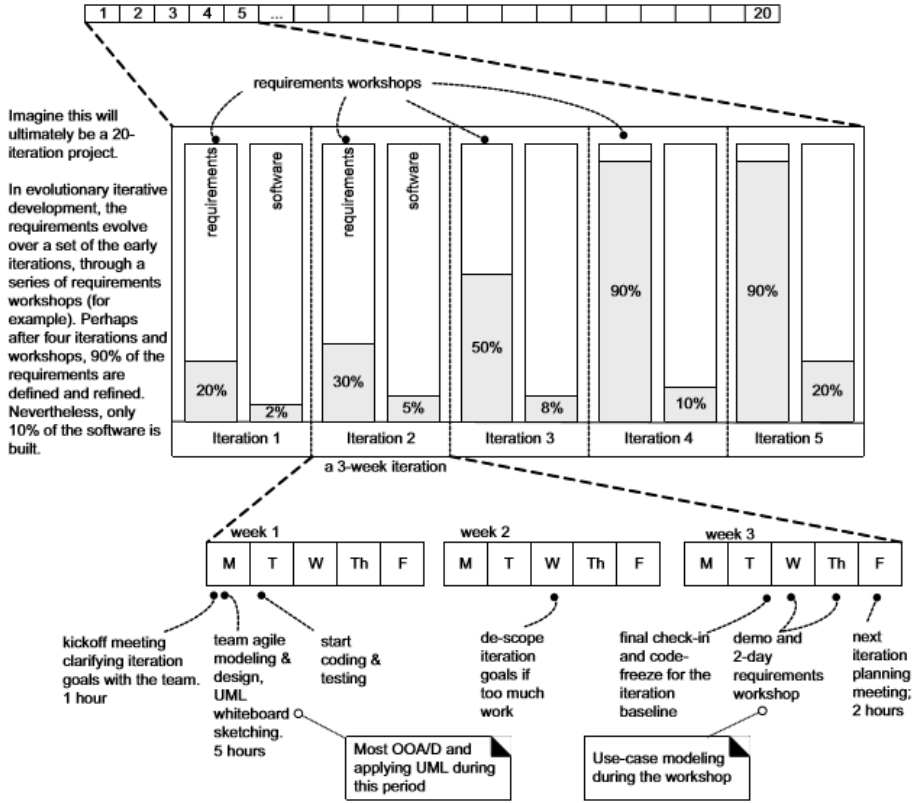
NO 'Waterfall Thinking' in Iterative Development! "…on average 45% of the features in waterfall requirements are never used, and early waterfall schedules and estimates vary up to 400% from the final actuals…"

Risk-Driven, Customer-Driven Iterative Planning

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

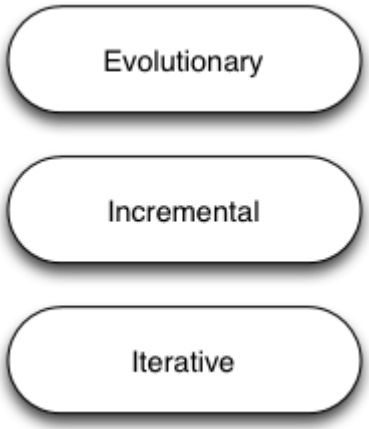


Evolutionary Analysis & Design in Early Iterations

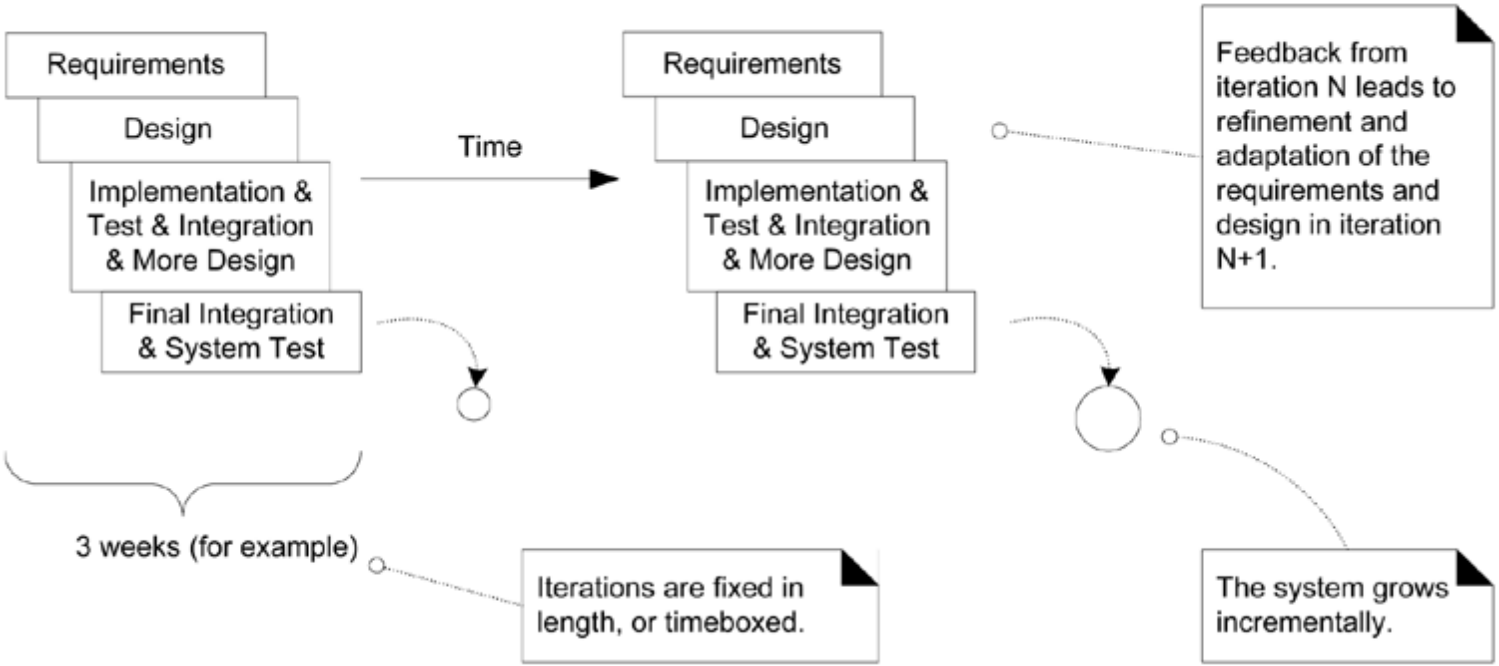


Typical *Iteration* includes…

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



Time-Boxed Iteration with *Feedback*



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

Agile Frameworks

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

Projects with Agile Frameworks:
Defining Structure vs. Being Prescriptive

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

Lean Methods – Inspiration for Agile

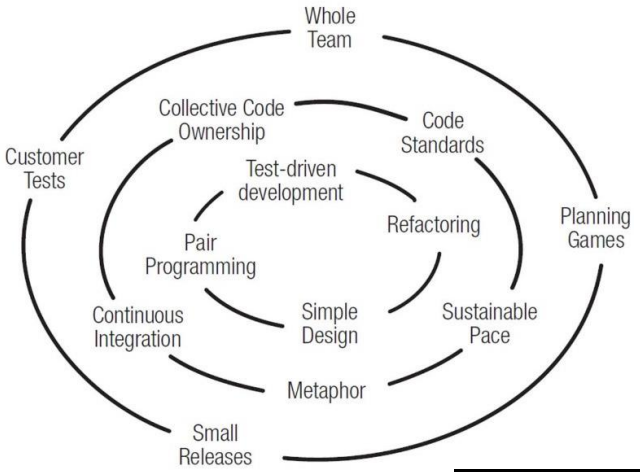
- Originated in Japanese Manufacturing Organizations (Toyota's *Kanban* system), Lean Methods **Focus** on:
  - Eliminating Wastage in mass-manufacturing processes (Just-In-Time Production System)
  - Focus on Humans in the Decision-making in the Production Process (vs. Expensive Machines)
  - Ownership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

Extreme Programming (XP)

- Guiding Spirit: *Extreme Focus on Customer and Projects are like War-rooms*
  - Features to be developed when Customer needs them
  - New Requests (or Change-requests) accepted as part of daily routine
  - Dynamic Self-organization of Teams around Customer Problems or Issues as and when they surface
- Principles of XP:
  - Coding is the Language of the Product and Communication
  - Extensive Testing: Coding doesn't start unless Success-criteria is defined;

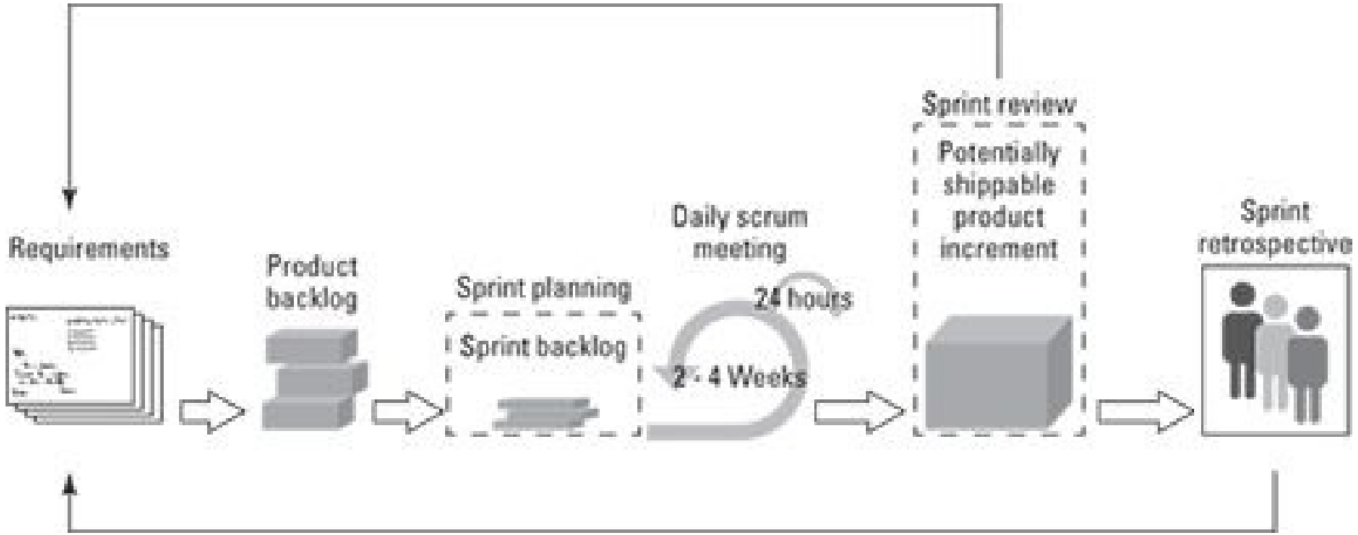
    "A bug is not a failure of code, it's a failure to define the right test"
  - Direct Communication between Programmer and Customer
  - Design during *Refactoring* to reduce Complexity and Maintainability

XP – Key Practices



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

SCRUM – The Common Agile Project Management Framework



Product Owner: Responsible for End-to-end Product Development
SCRUM Master: Manages the SCRUM Process (Not a *Manager* of Teams)
Cross-functional Teams: Involving Developers, Designers, Testers, and Operations Teams

Agile Software Development - Summary

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

Thank You

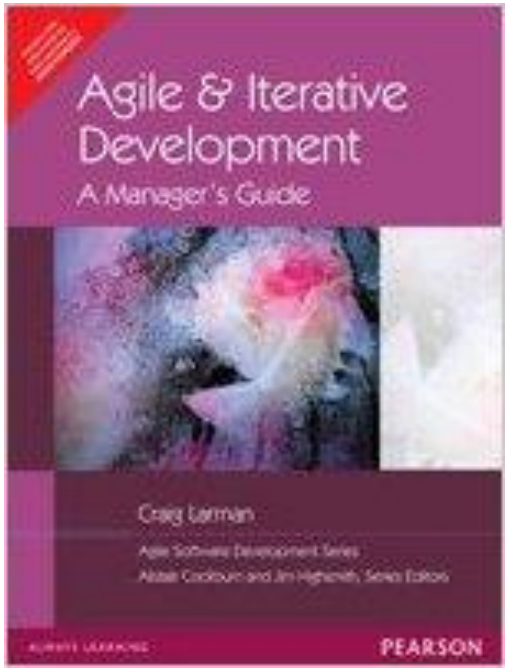
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Principles & Manifesto

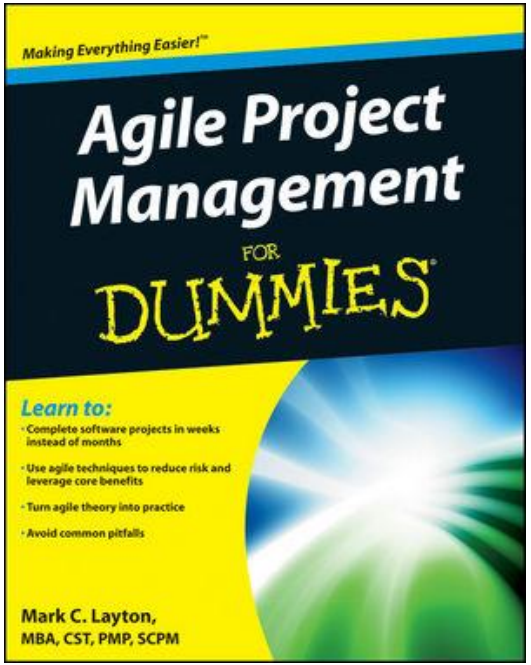
- Prof K G Krishna

Text/Reference Books

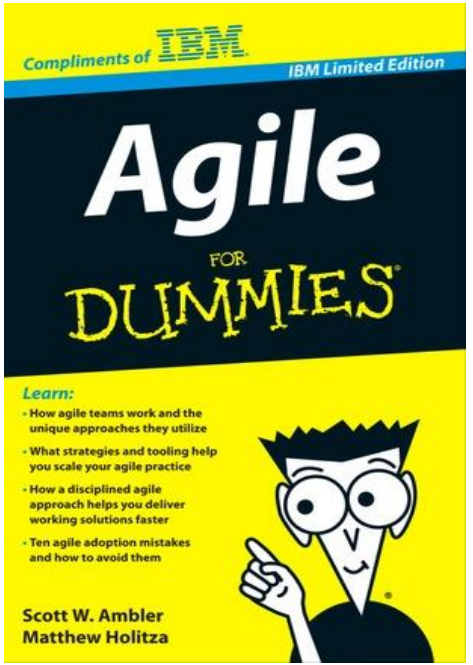
T1



T2



Compliments of IBM

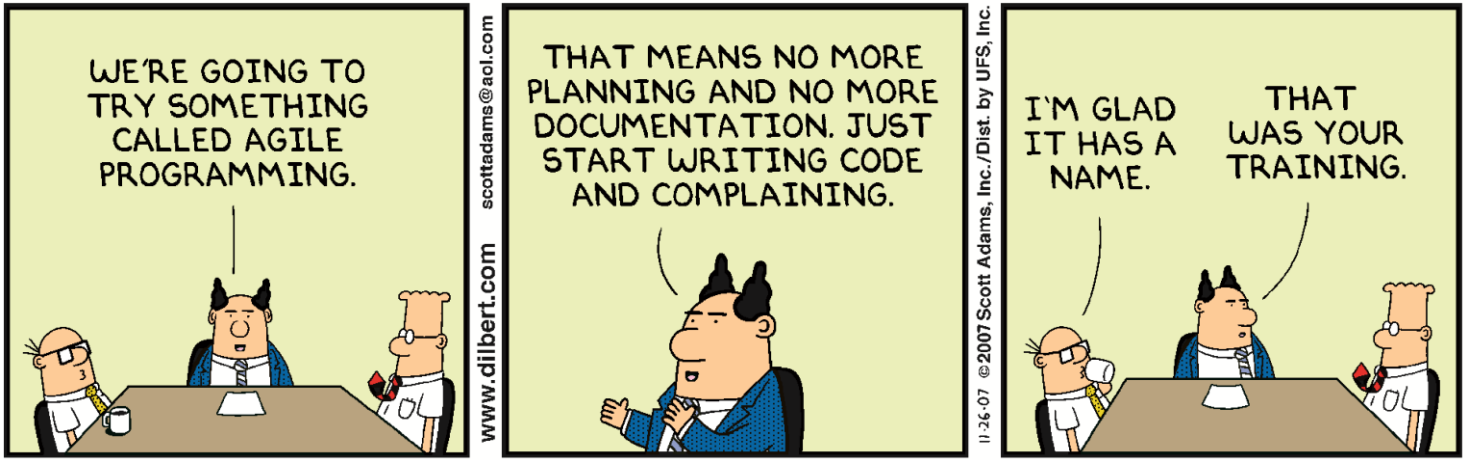


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

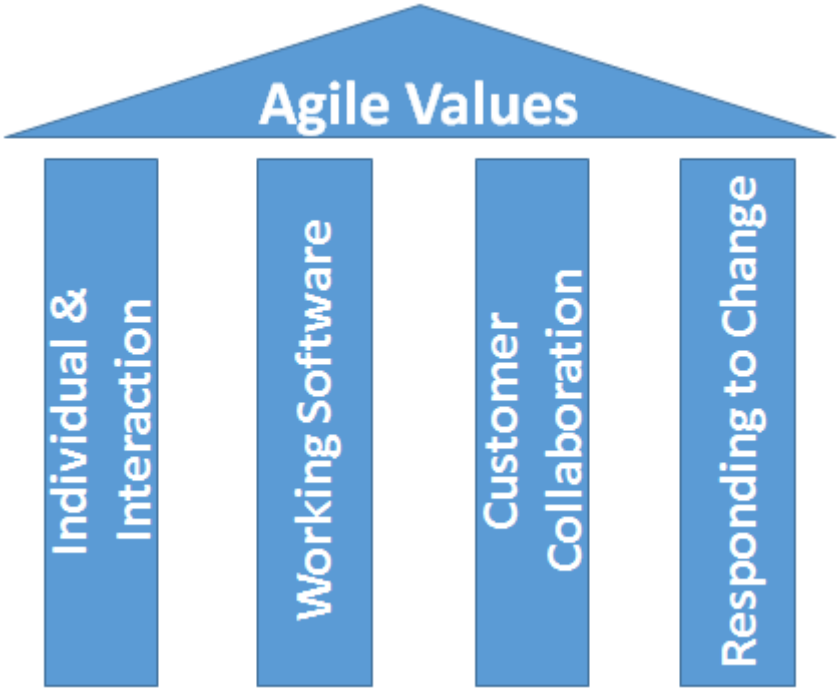
Topics

Principles of Agile

- Agile Values
- Agile Manifesto



Agile Core Values...



Agile Principles…

Principles behind the Agile Manifesto

We follow these principles:

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

Source courtesy: agilemanifesto.org

Agile Manifesto (2001)

Source courtesy: agilemanifesto.org

Agile Manifesto (adapted to times)

Source: https://www.ncsc.gov.uk/blog-post/securing-agile-delivery-collaboration-crucial

9 Principles *Agile Project Manager*

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

The Agile Project Manager

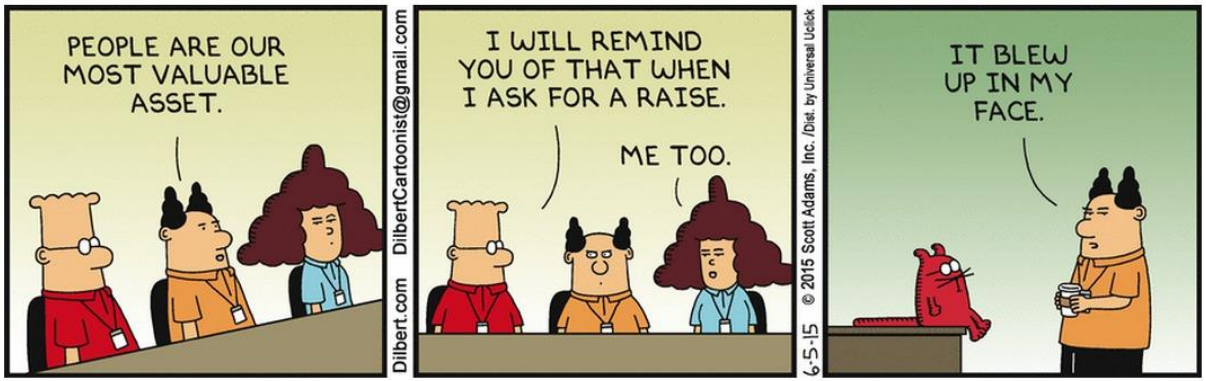
- Devolve of both control and planning to the entire team, not the manager
- The manager does not create WBS, schedule, estimates or tell people what to do
- The manager does not define and assign detailed team roles and responsibilities

Source: (T1-Chap3) Jim Highsmith, Agile Alliance Founder

The *Human Touch* in Agile

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

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



Agile Principles & Manifesto - Summary

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

Thank You

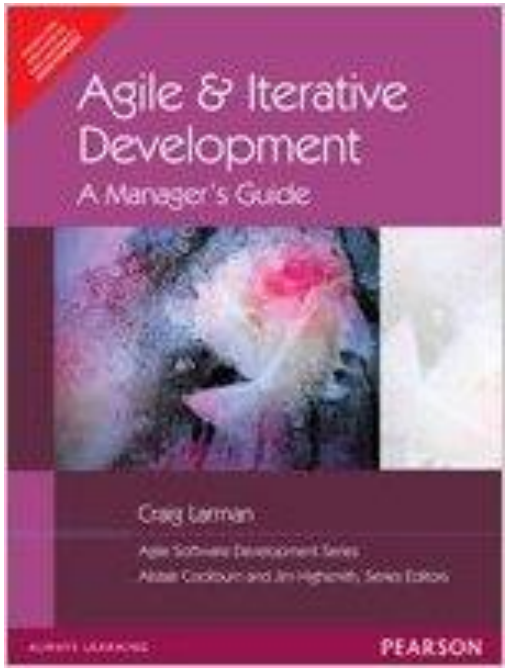
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Methodologies

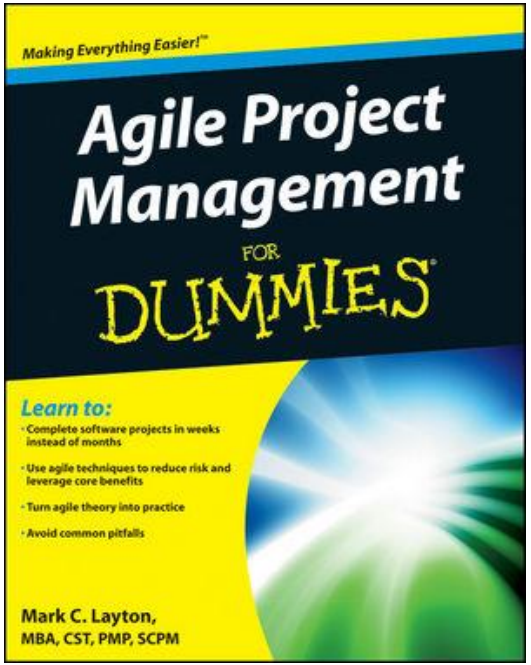
- Prof K G Krishna

Text/Reference Books

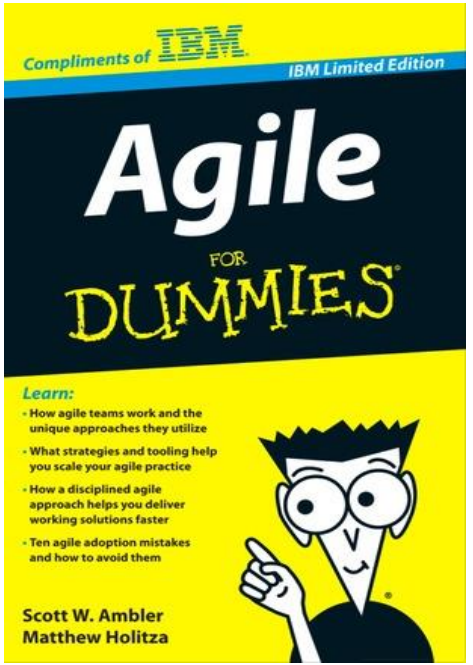
T1



T2



Compliments of IBM



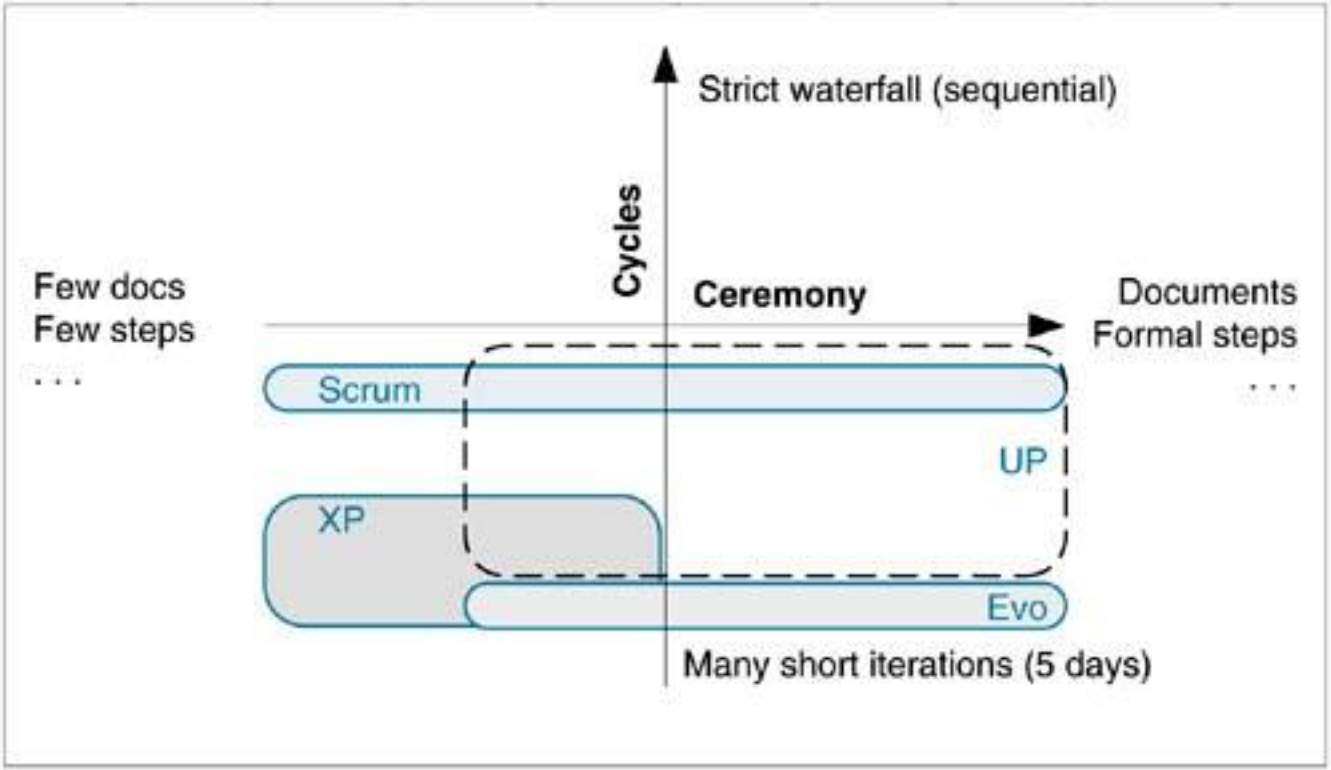
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Overview of Agile Methodologies

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

SCRUM on '*Ceremony – Cycles*' Scale



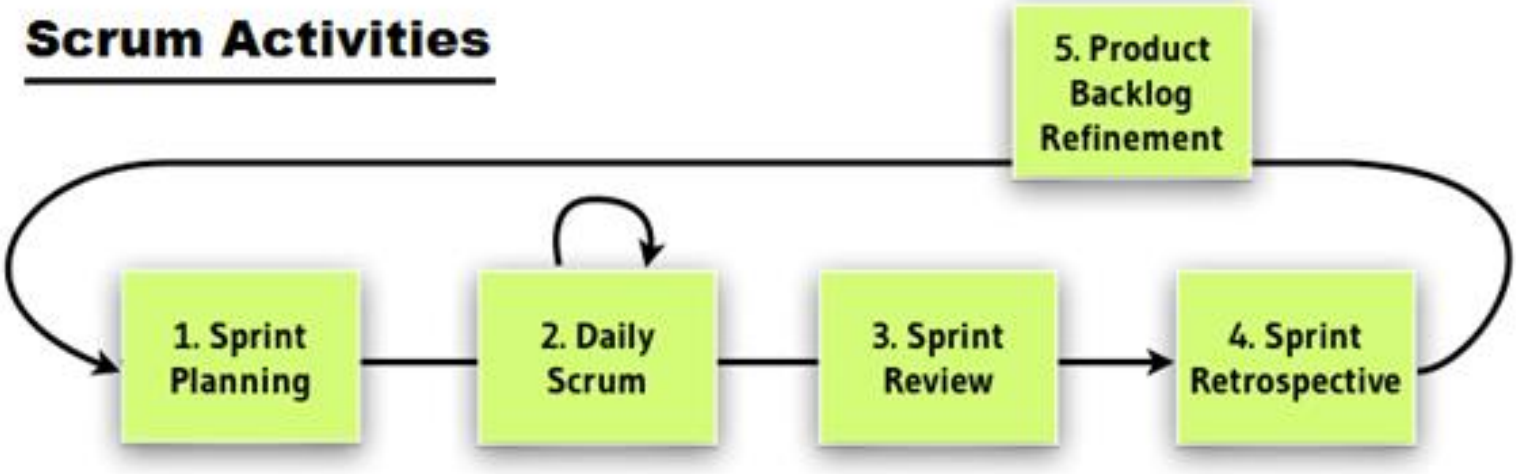
Source: T1-Chap7

SCRUM Lifecycle

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

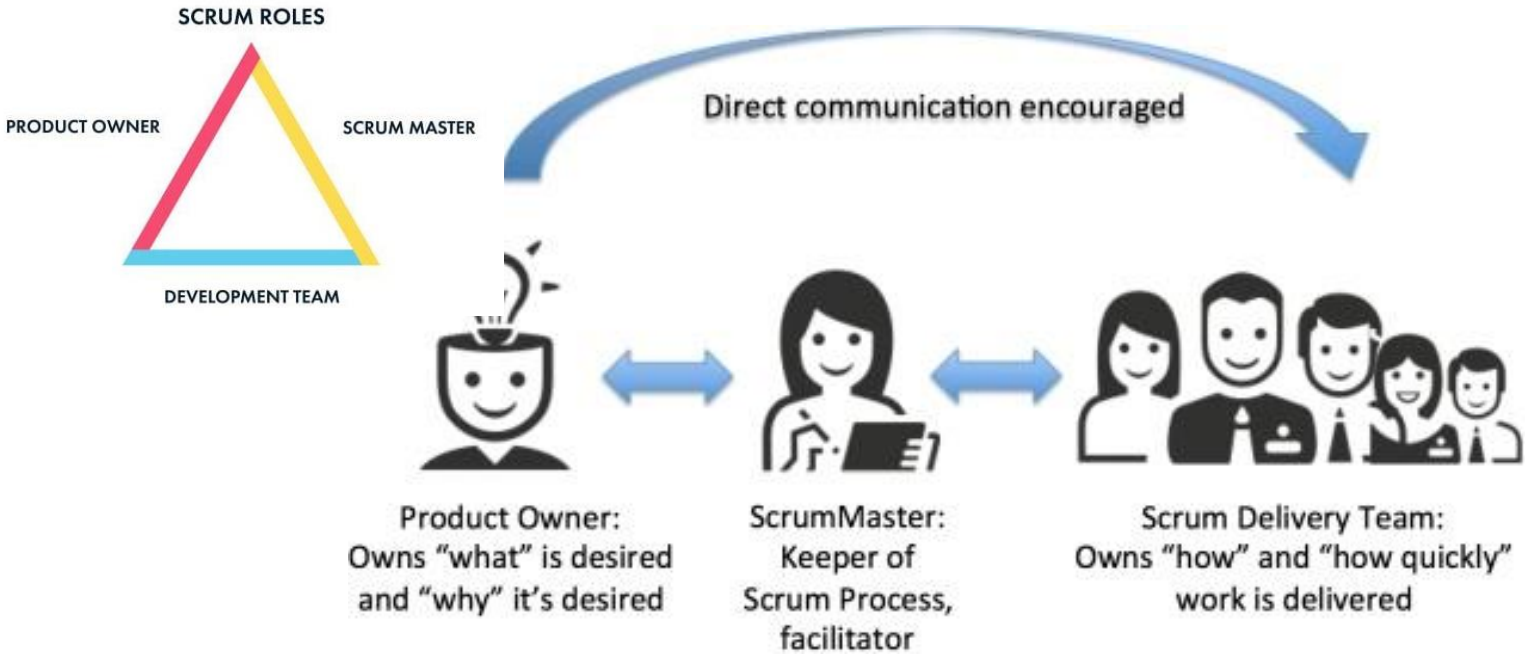
Source: T1-Chap7

SCRUM Activities

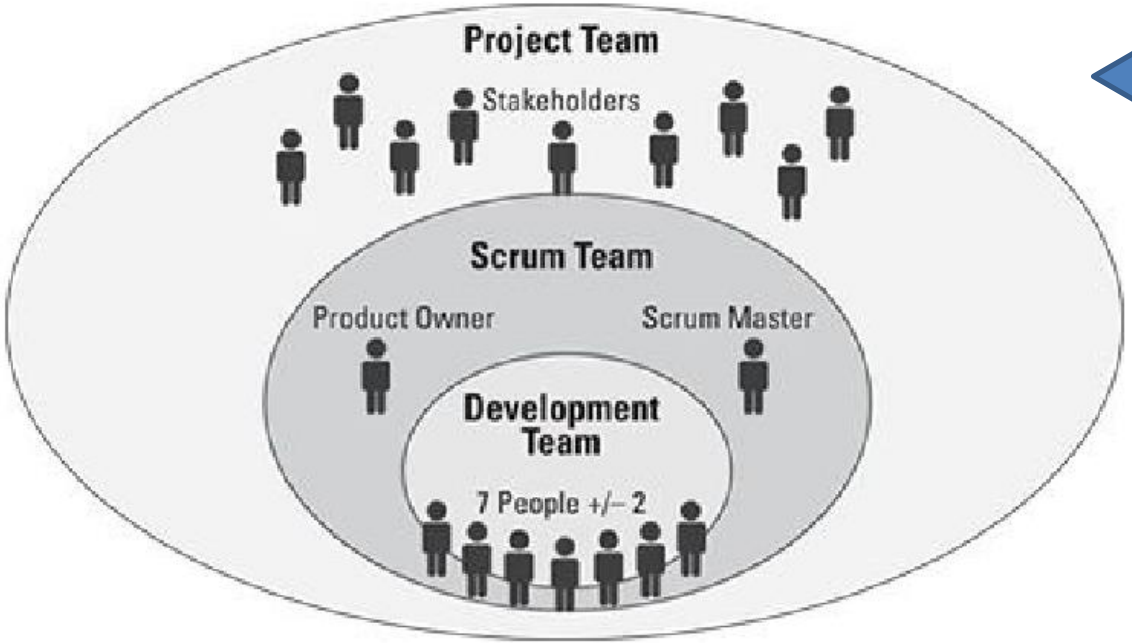


Source courtesy: www.snyxius.com/how-to-run-scrum-planning-meeting-like-pro

SCRUM – The *Key Players*



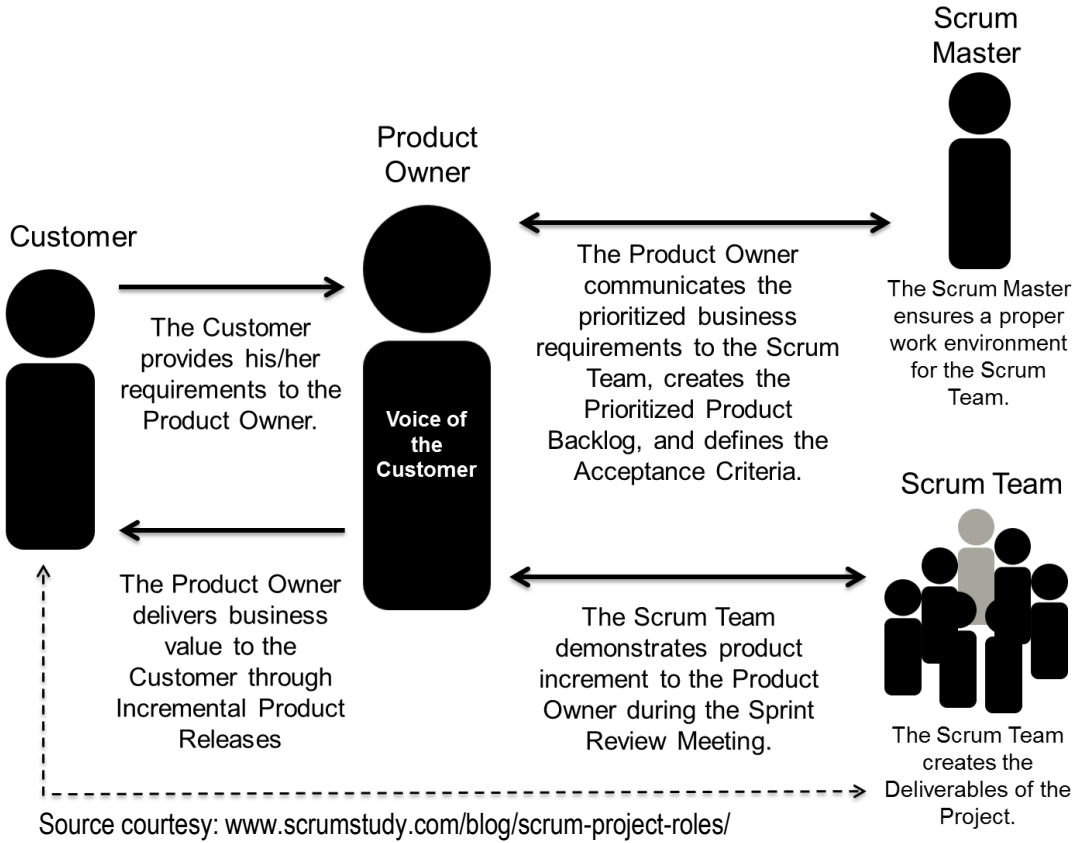
The ‘Extended’ SCRUM Team



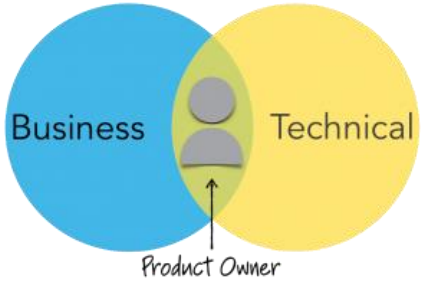
+ Agile Coach

Source: (T2)

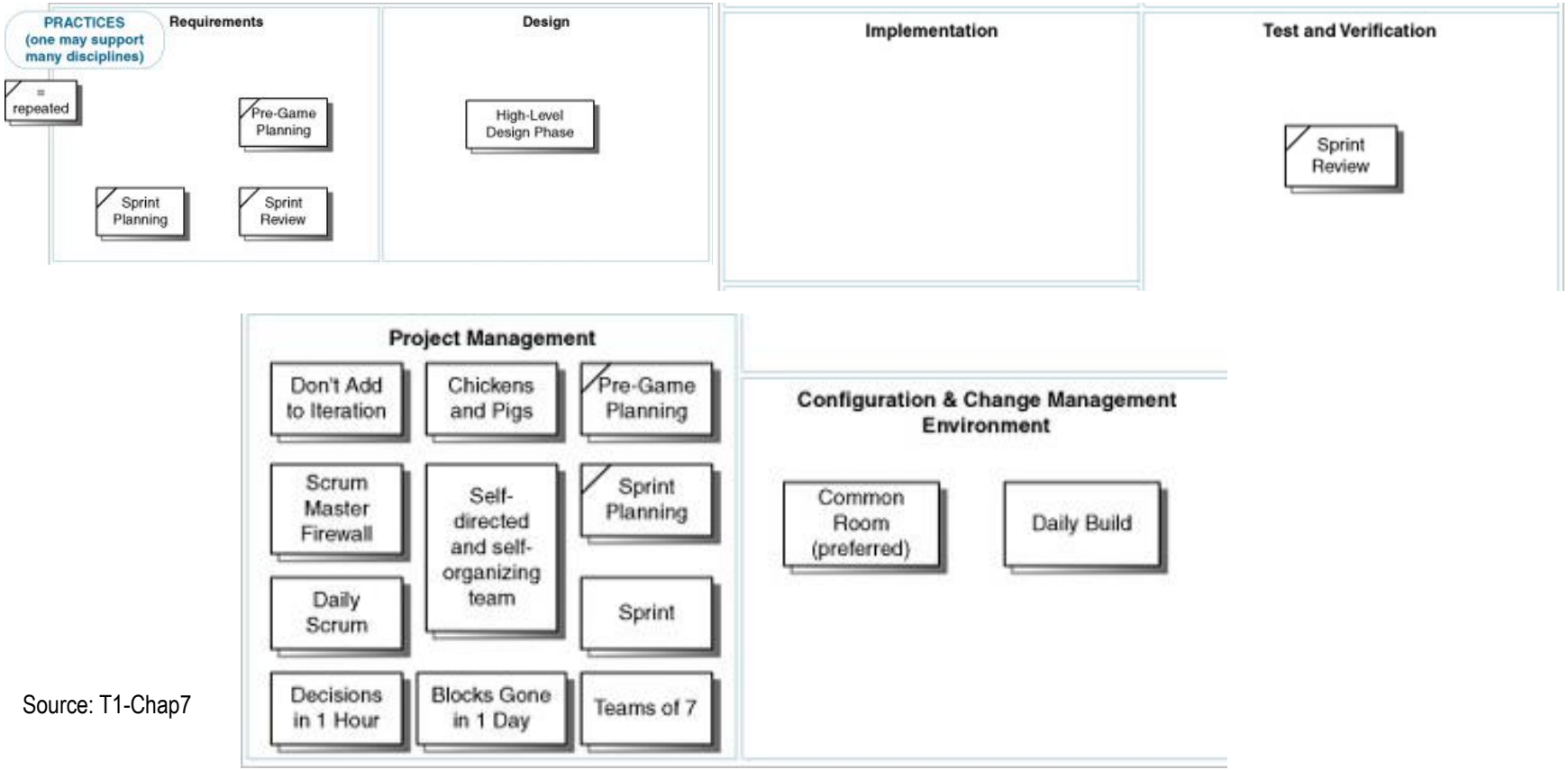
SCRUM Project Roles



Source courtesy: www.scrumstudy.com/blog/scrum-project-roles/



SCRUM Practices



Source: T1-Chap7

The Daily SCRUM Meeting –
The Heartbeat of the Project

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

SCRUM Artifacts

Product Backlog

Requirement	Num	Category	Status	Pri	Estimate
log credit payments to AR	17	feature	underway	5	2
process sale-simple cash scenario	232	use case	underway	5	60
slow credit payment approval	12	issue	not started	4	10
sales commission calculation	43	defect			
lay-away plan payments	321	enhance			
PDA sale capture	53	technology			
process sale-credit pmt scenario	235	use case			

Sprint Backlog

Task Description	Originator	Responsible	Status	Hours of work remaining					
				6	7	8	9	10	1
				362	322	317	317	306	2
Meet to discuss the goals and	JM	JM/SR	Completed	20	10	0	0	0	
Move Calculations out of	TL	AW	Not Started	8	8	8	8	8	
Get GEK Data		TN	Completed	12	0	0	0	0	
Analyse GEK Data - Title		GP	In Progress	24	20	30	25	20	
Analyse GEK Data - Parcel		TK	Completed	12	12	12	12	12	
Define & build Database		BR/DS	In Progress	80	80	75	60	52	

Source: T1-Chap7

SCRUM Values

("At the end, It's the People that Matters the Most")

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

SCRUM Drawbacks (discountable however!)

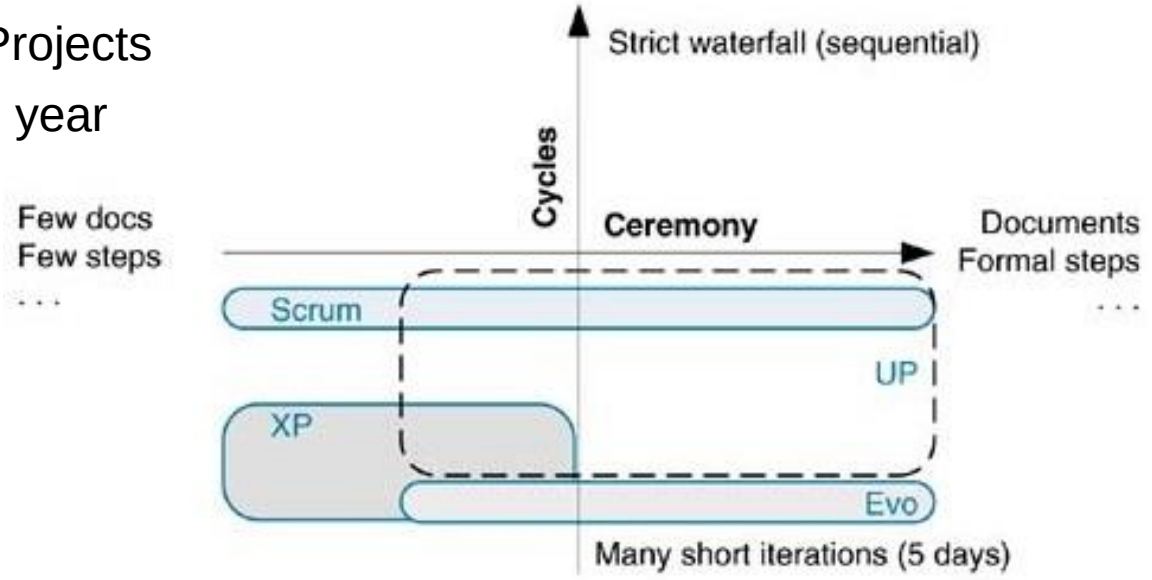
- **Active Engagement of the Customer**: the customer has to be continually involved in the project; however, while this can be seen as an advantage, it requires a lot of time and effort to manage the process
- **High Visibility may lead to Scope Creep**: as the Scrum methodology makes problems more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

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

➔➔

Extreme Programming (XP) – Core Values

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



Source: T1-Chap8

XP Lifecycle

EXPLORATION	PLANNING	ITERATIONS TO FIRST RELEASE	PRODUCTIONIZING	MAINTENANCE
Purpose: - Enough well-estimated story cards for first release. - Feasibility ensured.	**Purpose**: - Agree on date and stories for first release.	**Purpose**: - Implement a tested system ready for release.	**Purpose**: - Operational deployment	**Purpose**: - Enhance, fix. - Build major releases
Activities: - prototypes - exploratory proof of technology programming - story card writing and estimating	**Activities**: - Release Planning Game - story card writing and estimating	**Activities**: - testing and programming - Iteration Planning Game - task writing and estimating	**Activities**: - documentation - training - marketing - . . .	**Activities**: - May include these phases again, for incremental releases.

Source: T1-Chap8

XP Core Practices (12)

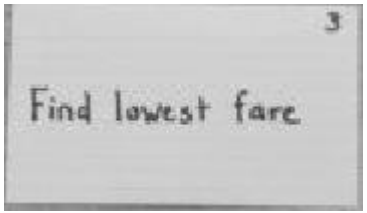
1. Planning Game
2. Small, Frequent Releases
3. System Metaphors
4. Simple Design
5. Testing
6. Frequent Refactoring
7. Pair Programming
8. Team Code Ownership
9. Continuous Integration
10. Sustainable Pace
11. Whole Team Work together
12. Coding Standards

Pitfalls of XP

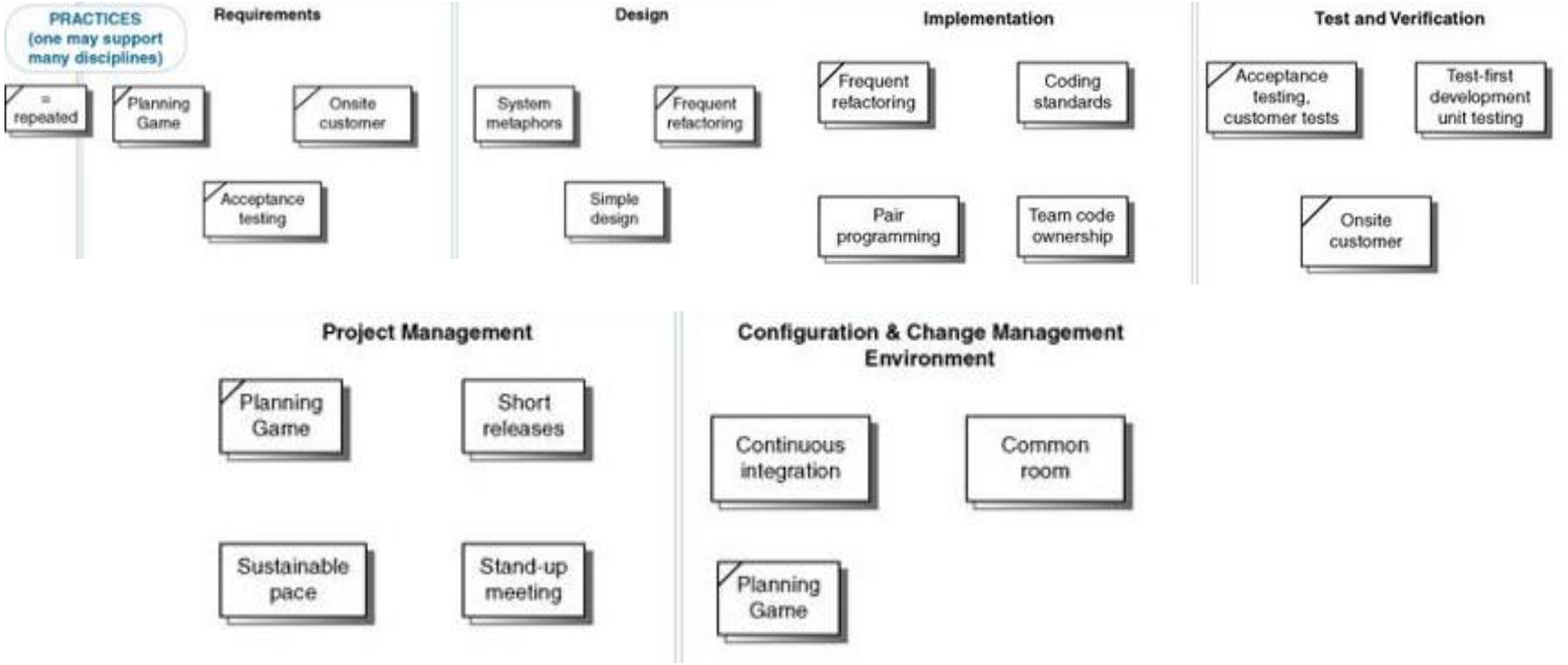
- Requires presence of Onsite Customer or his/her Proxy
- Relies on Informal oral understanding – difficult to ramp-up to speed new members or in large projects
- XP practices are highly interdependent and tightly coupled—we can’t be selective in any
- No ‘Standard’ means of documenting design
- Pair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

Extreme Programming (XP) Work Products

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



XP Practices



Source: T1-Chap8

XP – Other Common Practices/Values

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

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

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

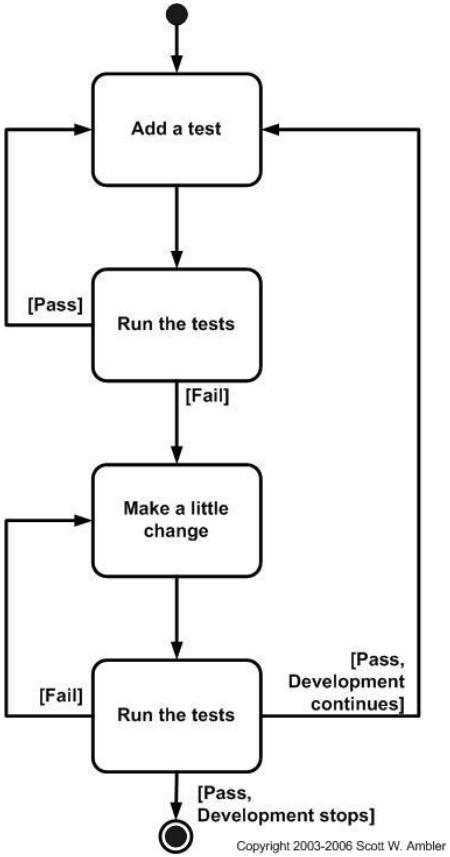
➔➔

Test-Driven Development

- Writing Test-cases before Coding (Unit Tests are written before writing the Code to be tested)
- Adoption of Short Iterative Development Cycle & Automated-Test Suites
- Eliminates “coder bias”

- “test with a purpose" - know why you are testing something and to what level it needs to be tested
- “achieve 100% coverage test” – every single line of code is tested
- “well-written unit-tests provide working specification of functional code” (code ~ documentation, tests ~ specifications)
- “proxies”: (unit-tests ~ design specifications, acceptance-tests ~ requirements)

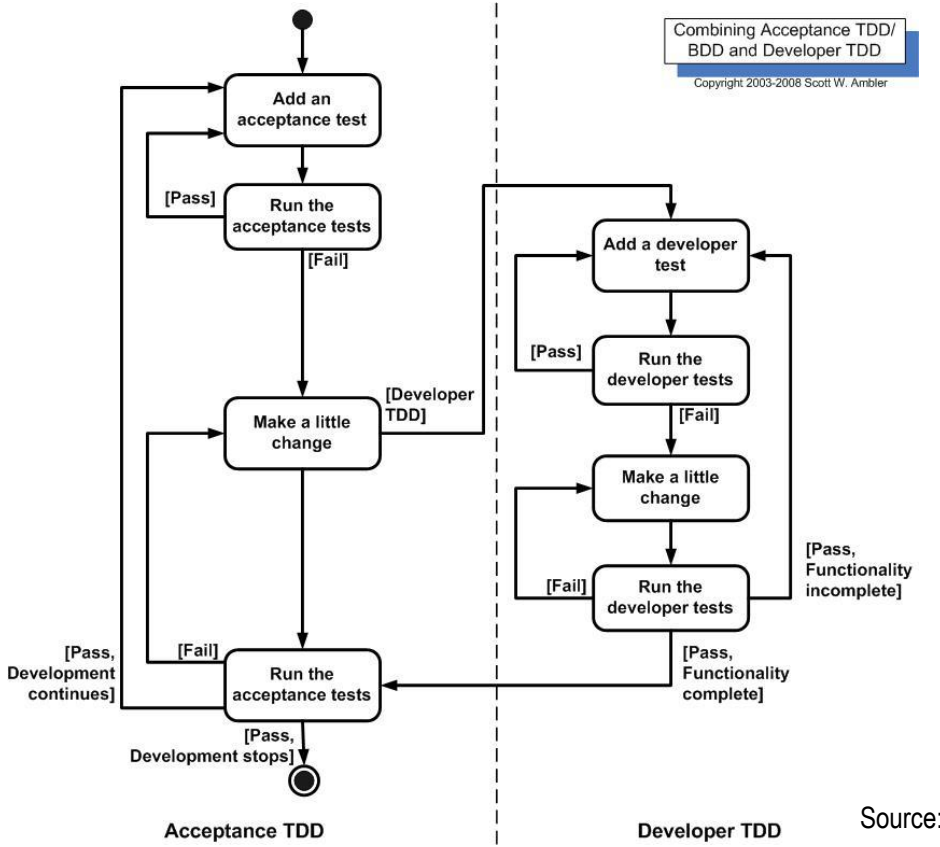
Test-Driven Development (TDD) =
Test-First Development (TFD) + Refactoring



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

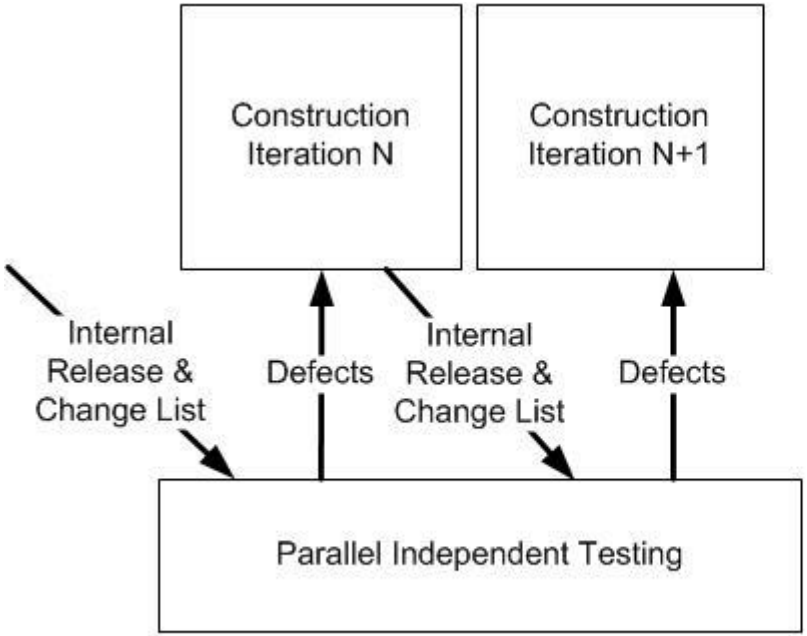
CUnit
DUnit (Delphi)
DBFit
DBUnit
DocTest (Python)
Googletest
HTMLUnit
HTTPUnit
JMock
JUnit
Moq
NDbUnit
NUnit
OUnit
PHPUnit
PyUnit (Python)
SimpleTest
TestNG
TestOoB (Python)
Test::Unit (Ruby)
VBUnit
XTUnit
xUnit.net

TDD: Acceptance TDD, Developer TDD



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

TDD is NOT an End to Itself...



Confirmatory Testing
- Acceptance TDD
- Developer TDD
- Developer regression testing

Independent Testing:
- Pre-production integration testing
- Other expensive/skillful testing

Agile Methodologies - Summary

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

Thank You

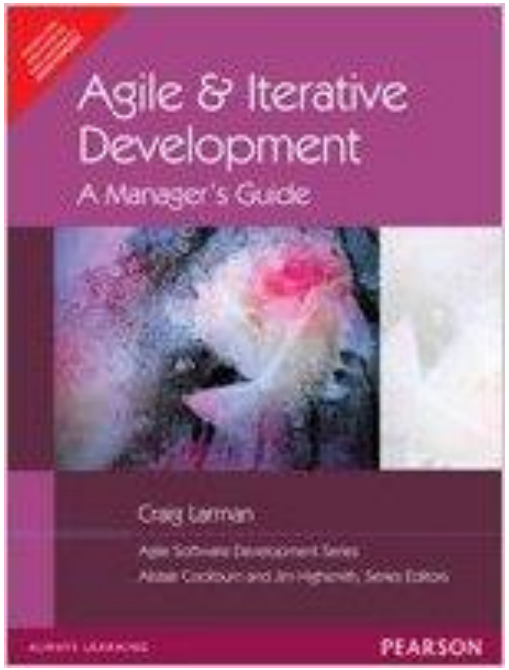
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Requirements Management in Agile

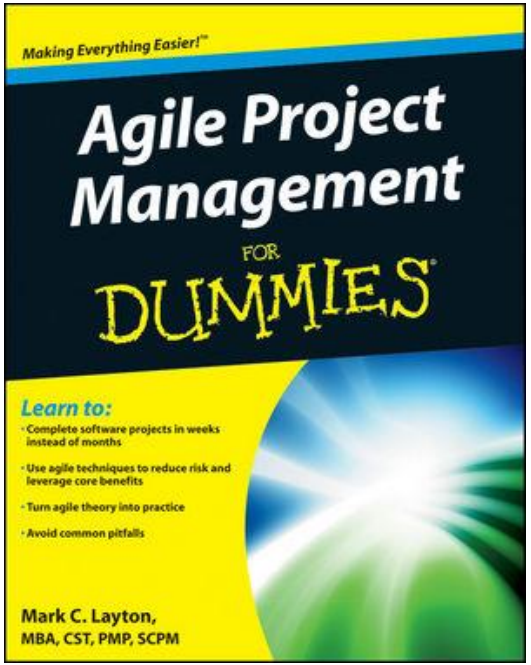
- Prof K G Krishna

Text/Reference Books

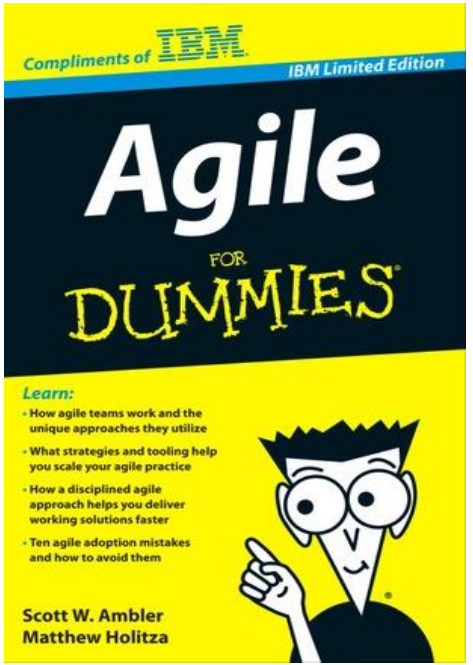
T1



T2



Compliments of IBM



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

Topics

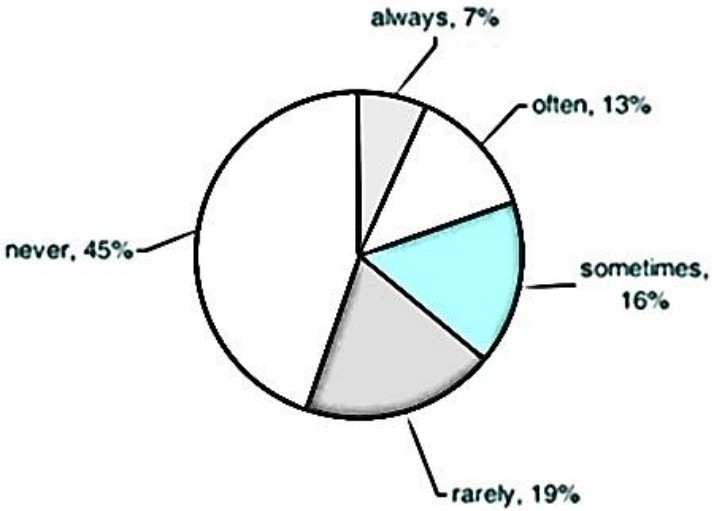
Agile Requirements Management

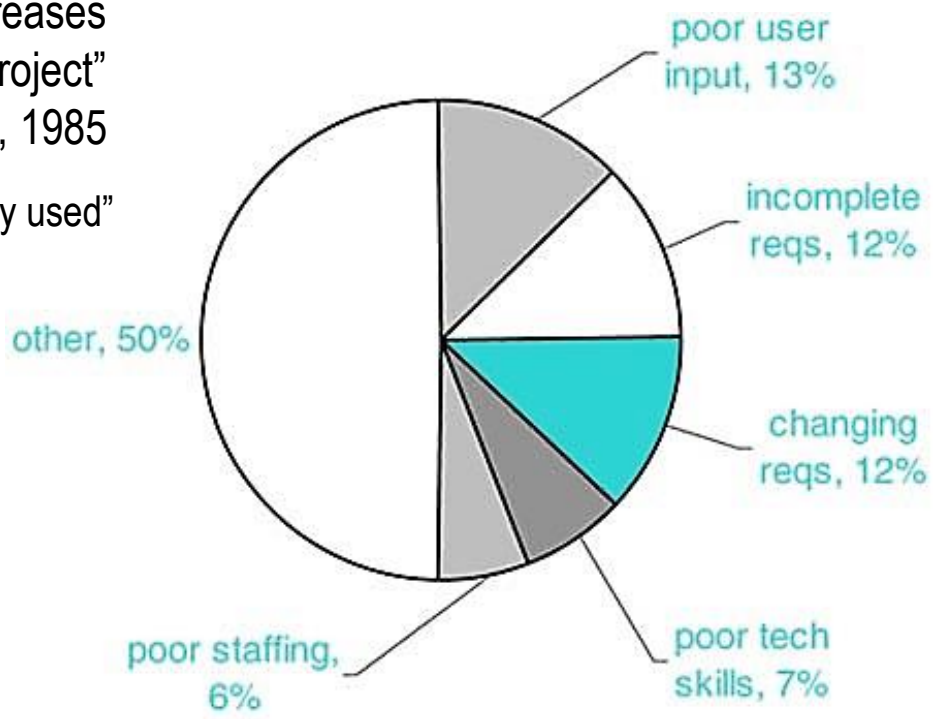
- Managing Requirements Iteratively
- User Stories as Requirements
- Size/Effort Estimation
- Prioritization Techniques
- Preparing the Product Roadmap

37% of Project Challenges are in *Requirements Management*

"Cost of fixing a Requirement defect increases non-linearly from early to late in the Project"
- B. Boehm, 1985

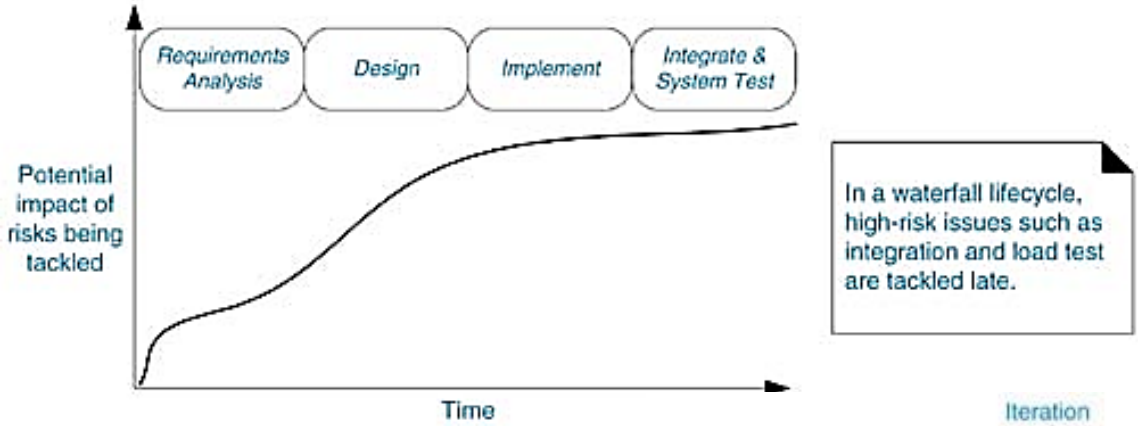
"45% of Features never used and ~20% rarely used"





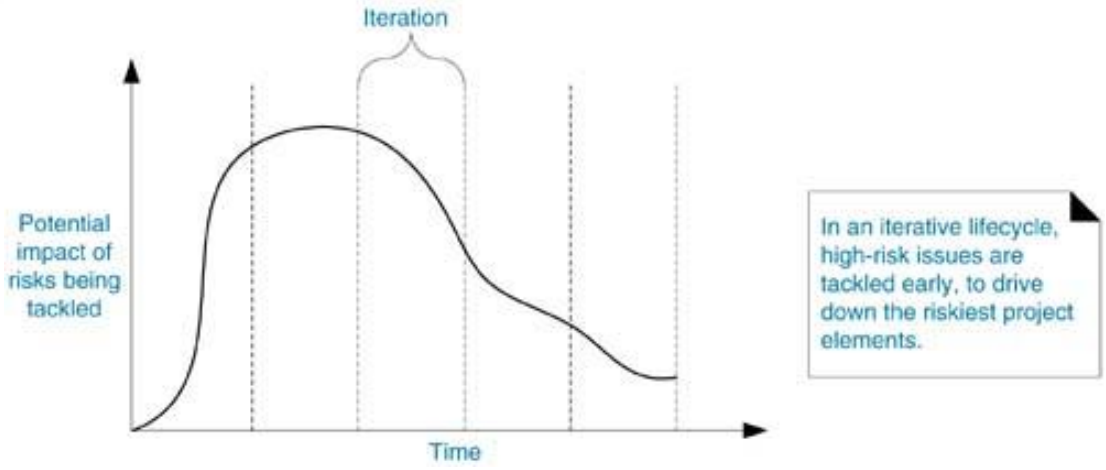
Source: (T1) / Standish Group Report, 2004

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



Risk Profile in Waterfall Model

Risk Profile in IID



Source: (T1-Chap 5)

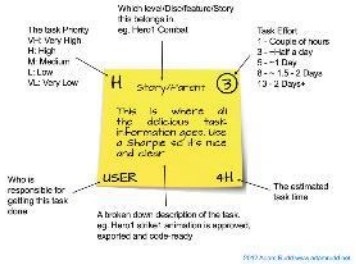
Identifying Product Requirements in Agile

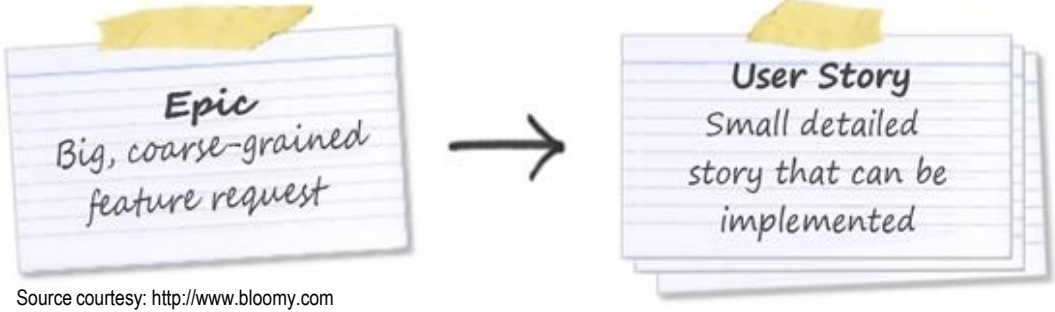
- From the Product Roadmap, arrive at Initial Set of High-priority Requirements
  - Product Roadmap ➔ Requirements ➔ Releases ➔ Sprints
- Requirements ➔ Logical Groups
- Decompose Requirements into: (level of detailing depends on early product definition)
  - **Themes**: logical grouping of features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

Source: http://eleventwenty.com

Building *Product-backlog*

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





Source courtesy: http://www.bloomy.com

Size/Effort Estimation in Agile ➔ ➔

Relative Effort Estimation

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

Relative Scoring of Requirements

- Relative Scoring of Requirements using “Fibonacci Sizing Sequence”

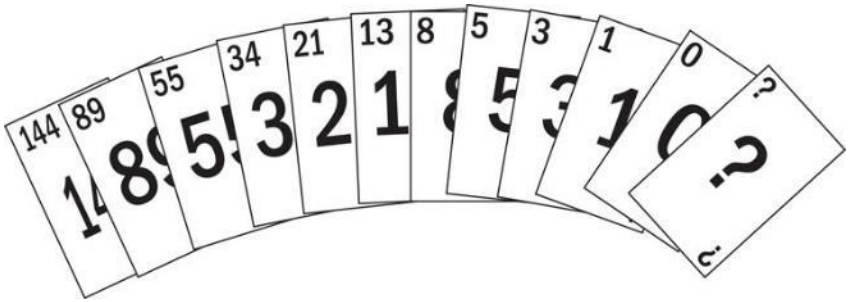
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

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

Effort Estimation by “Poker Estimation *Game*”

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

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



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

Source: www.platinumedge.com/estimationpoker

Poker Estimation (by Consensus) contd.,

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

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

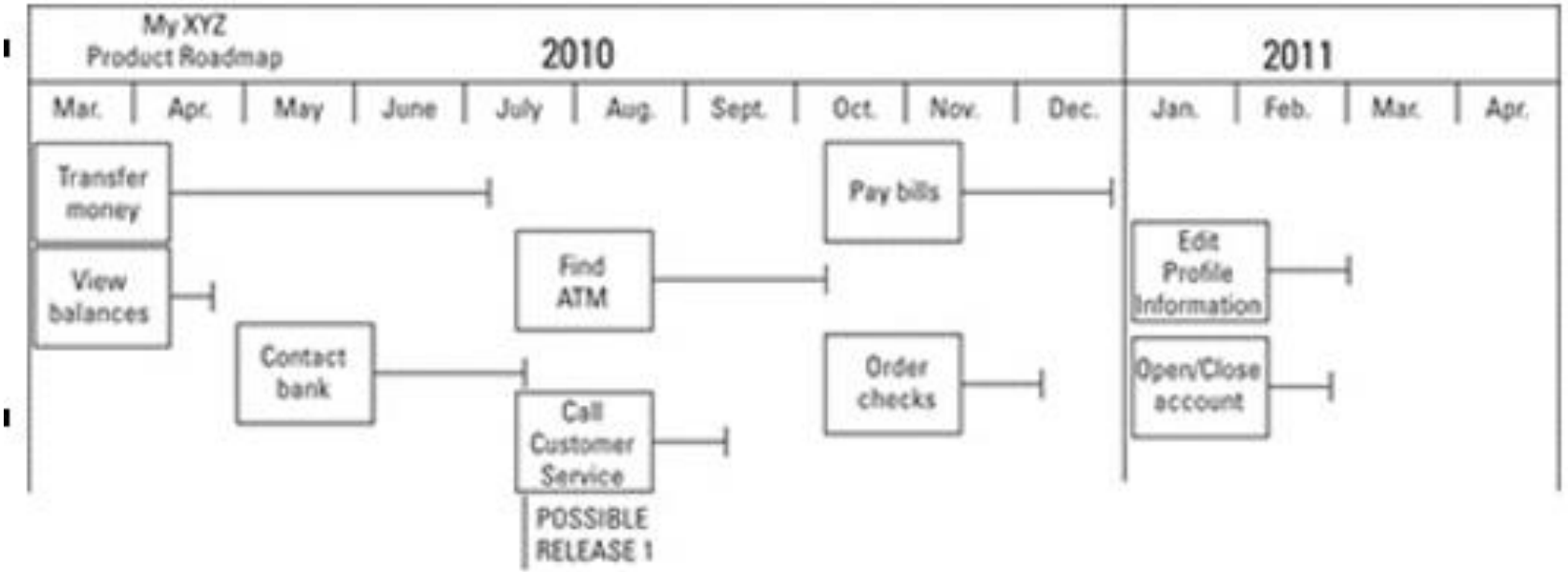
Estimates should be for the total ***Done*** – Developed, Integrated, Tested and Documented

Relative Prioritizing of Requirements

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

Build Product Roadmap with *Prioritized* Requirements

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



Source: (T2-Chap 7)

Summary: Agile Requirements

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

Thank You

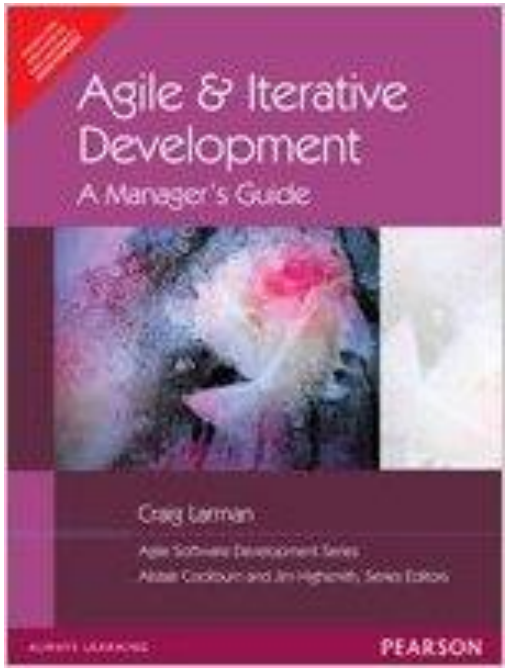
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Release Planning in Agile

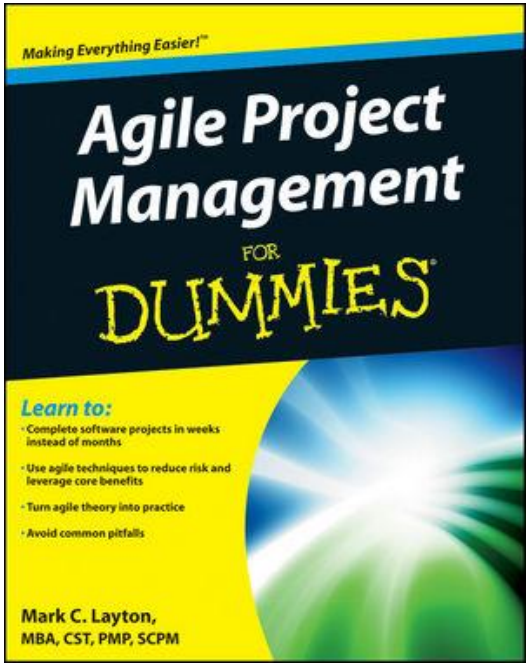
- Prof K G Krishna

Text/Reference Books

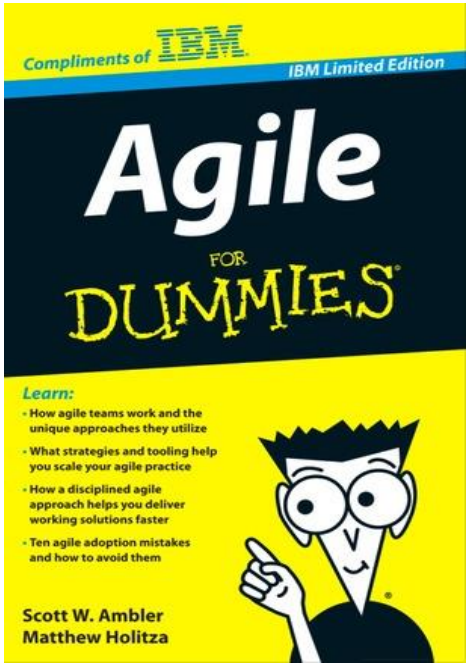
T1



T2



Compliments of IBM



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

Topics

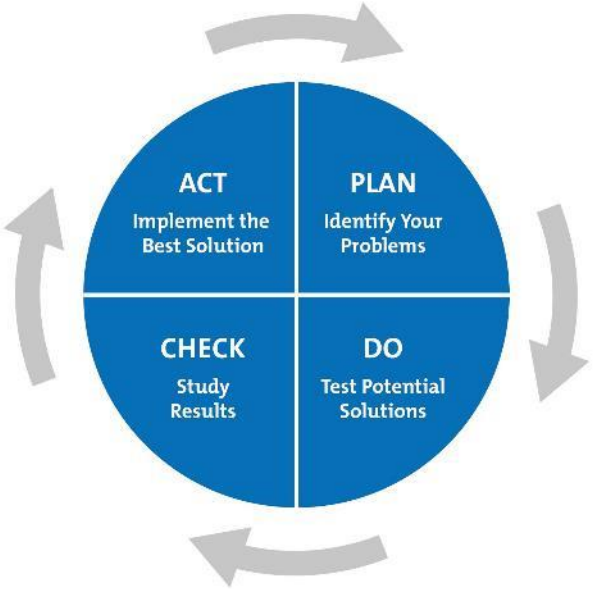
Release Planning in Agile Methods

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

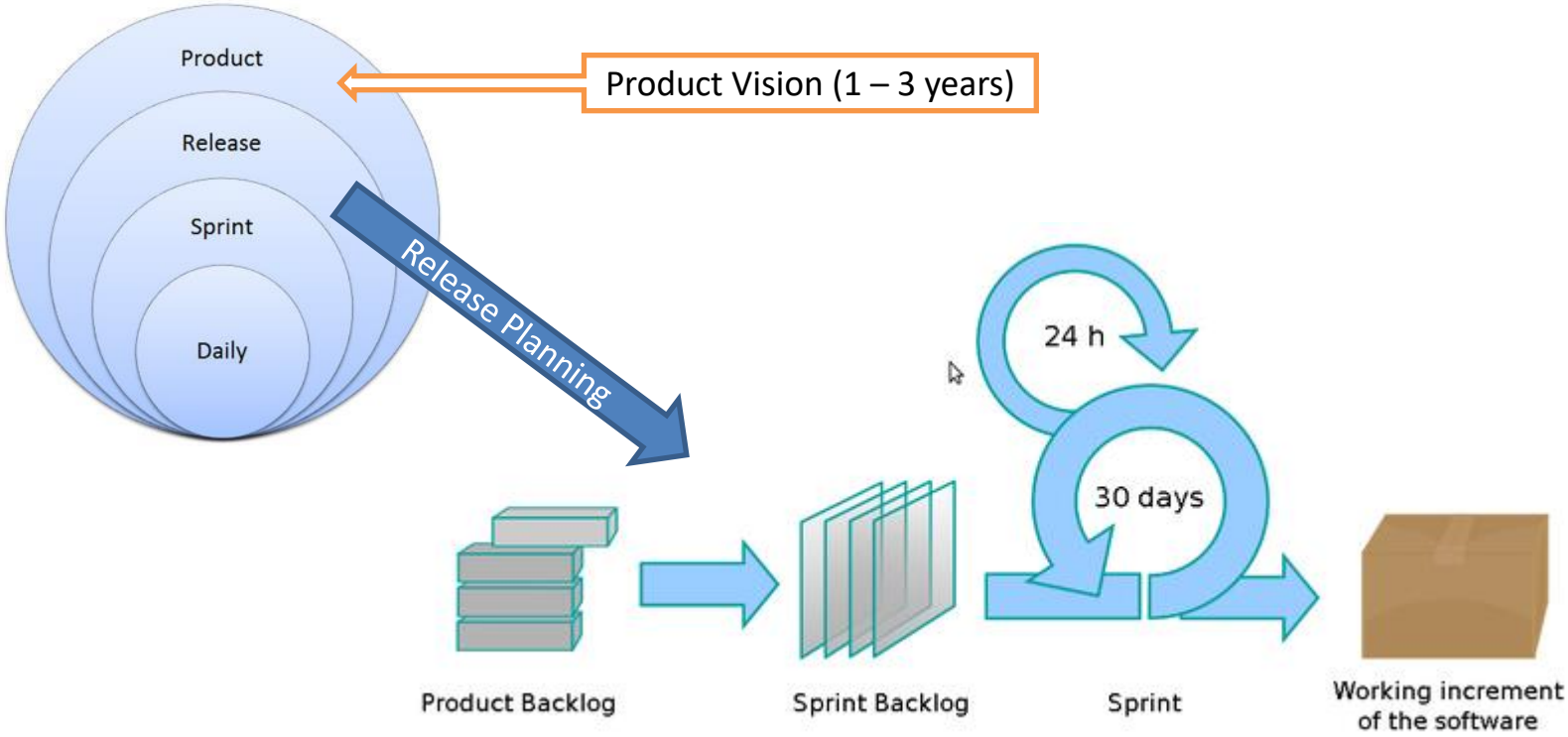
Planning is *Continuous*!
(Deming's Continuous Improvement Cycle)

- Planning is NOT a One-time Activity at the Beginning of a Project
- Part of PDCA (Deming's Cycle of Continuous Improvement)
- PDCA vs. "Plan-the-Work, Work-the-Plan" (Waterfall Model)

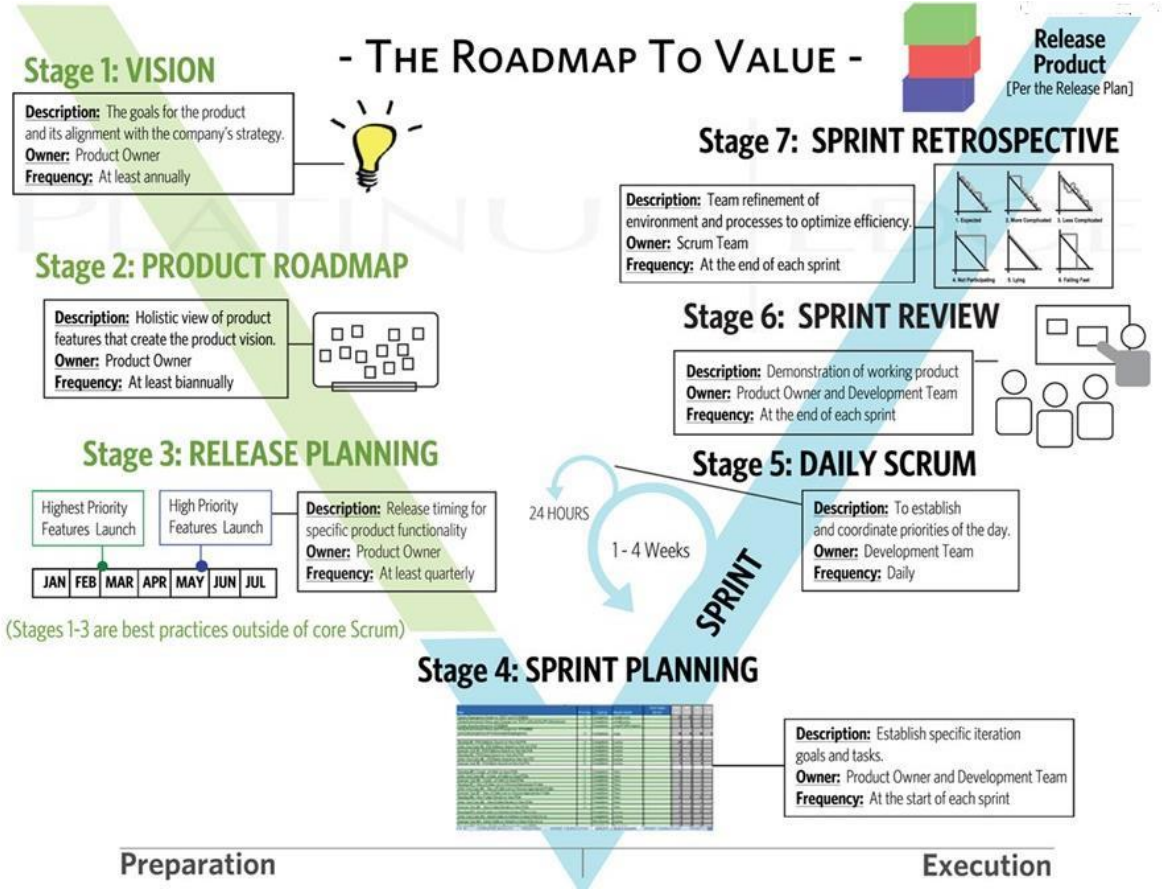
- Agile Planning
  - Just-In-Time Planning / "Situational Planning"
  - Agile Planning is Continuous throughout the Project



Levels of Planning in Agile



Planning at *Every* Stage in Agile (SCRUM)



Source: (T2)

Key Characteristics of Agile Planning

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

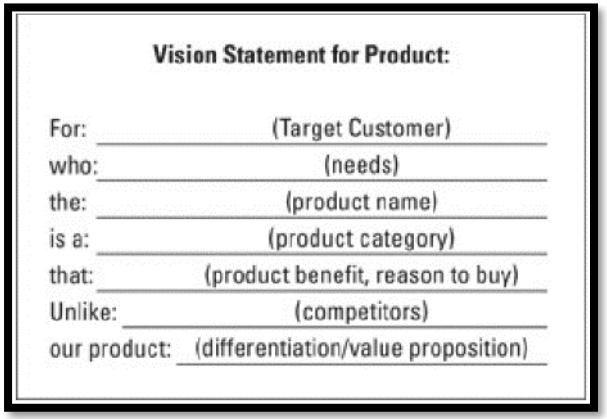
Plan-**D**o-**I**nspect-**A**dapt

Agile Release Planning Stages in Detail

➔➔

Stage-1: Defining Product Vision

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



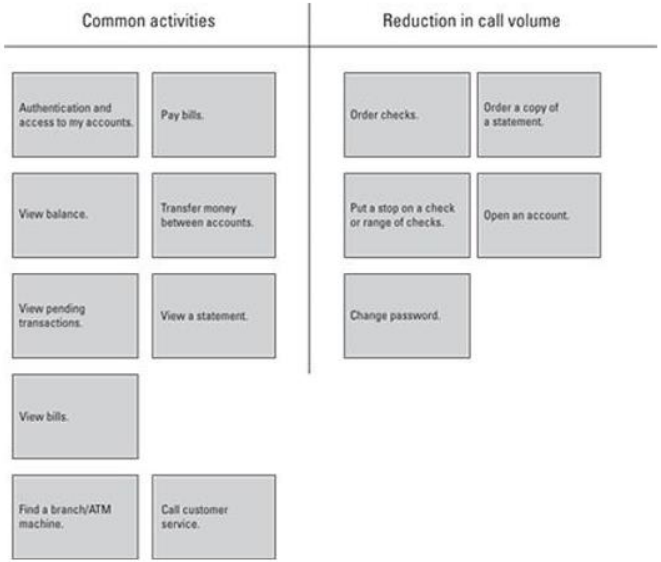
Vision Statement for Product:

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

Stage-2: Holistic Product Roadmap

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

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



Stage-3: Release Planning

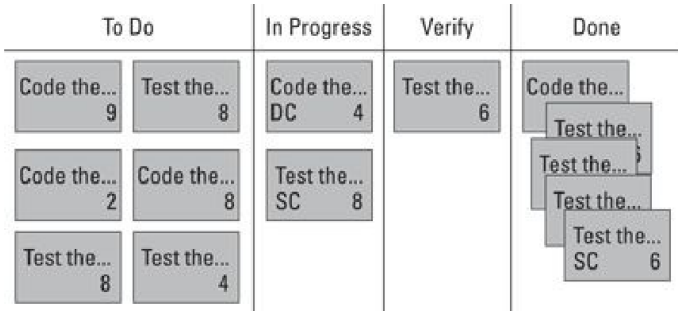
- Elaboration of Project Roadmap into Details
- A Release is a *Minimum set* of Marketable Requirements
- Requirements Breakdown Structure (WBS) by granular Decomposition
- “Epic-story ..→ User-stories”
- User-story: Simple Description of Requirements (user-walkthrough or benefit statement)
- Create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

Stage-4: Sprint Planning

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

Stage-5: Daily Scrum

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



Stage-6: Sprint Review

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

Stage-7: Sprint Retrospective

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

Preparing for Release

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

Summary: Release Planning

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

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

Thank You

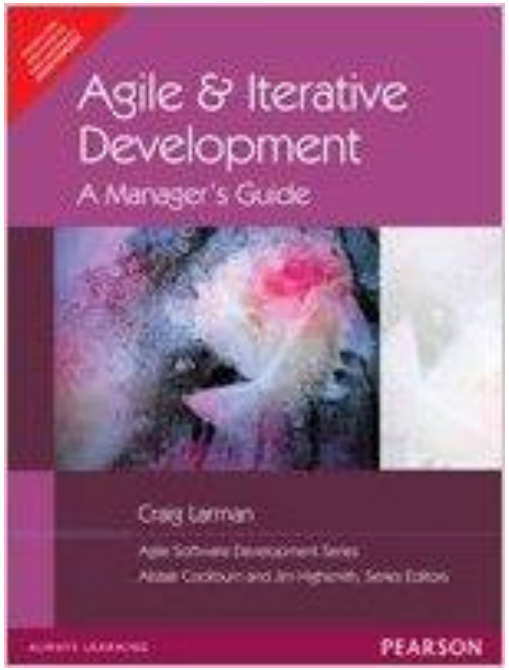
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Iteration Planning in Agile

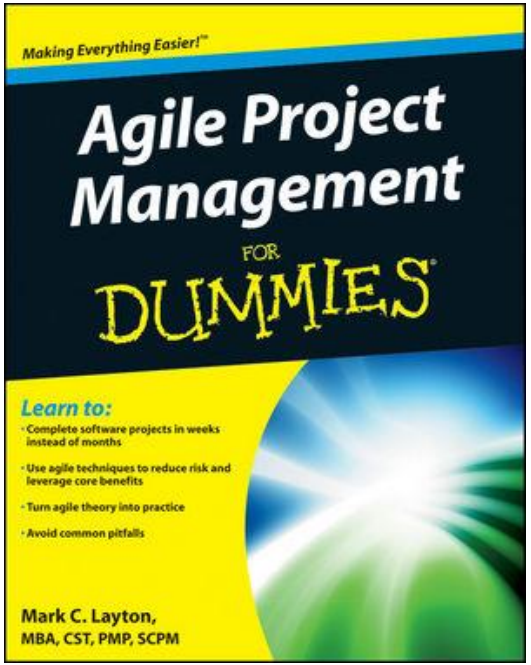
- Prof K G Krishna

Text/Reference Books

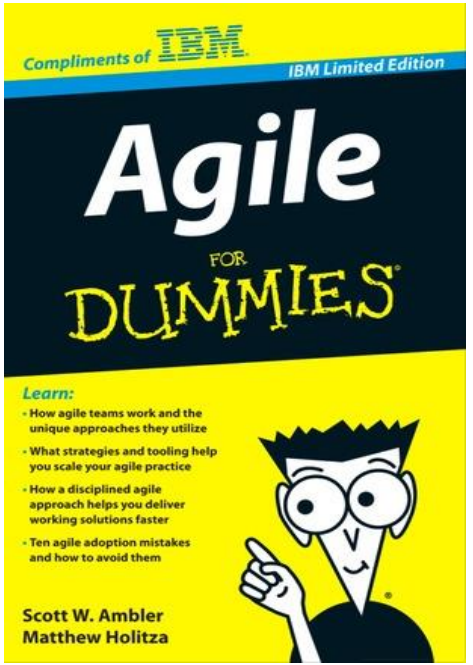
T1



T2



Compliments of IBM



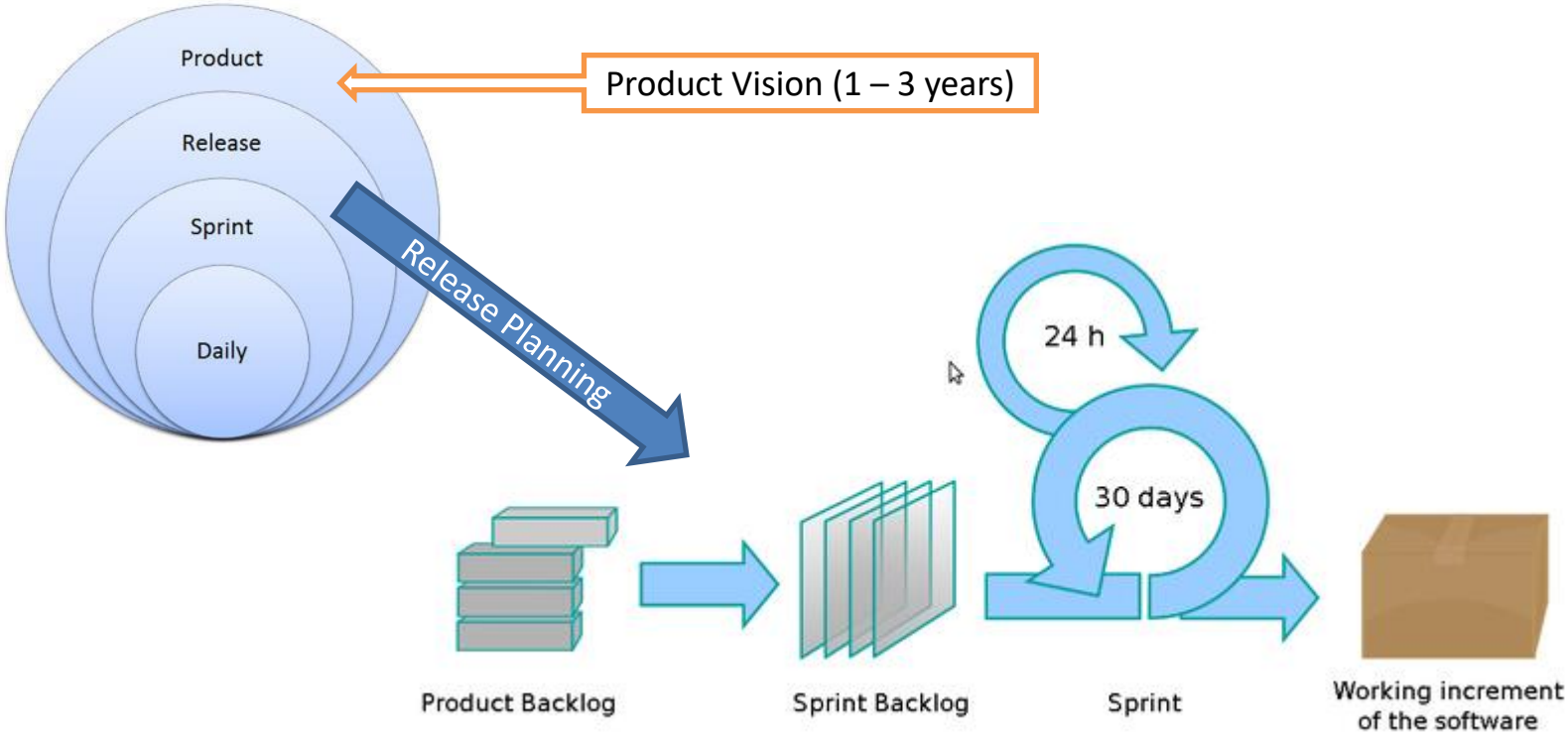
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

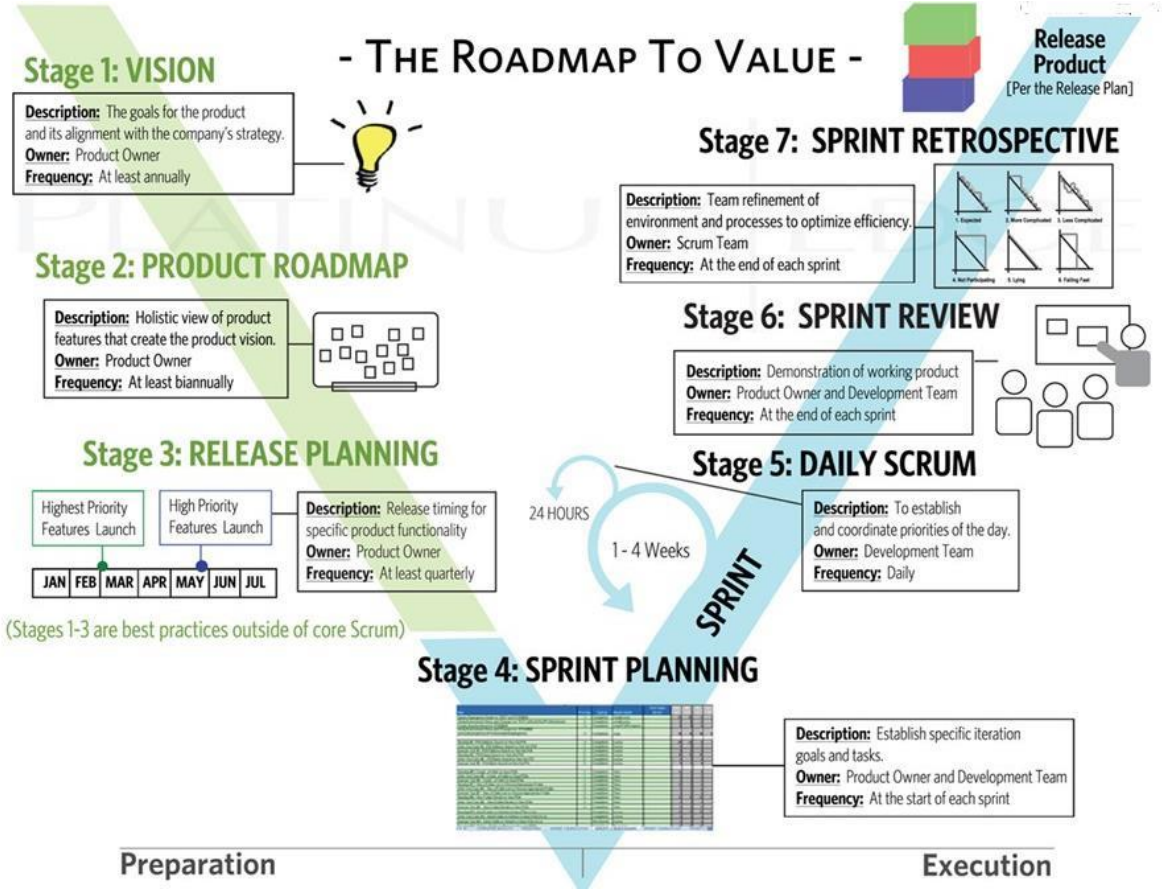
Iteration Planning in Agile

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

Levels of Planning in Agile

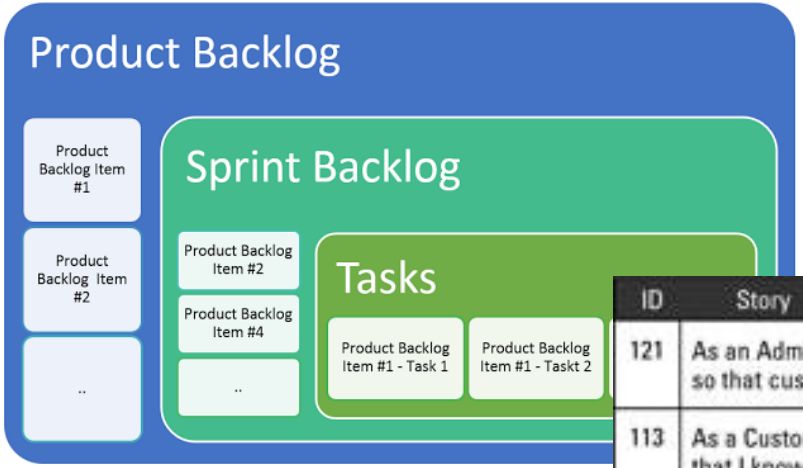


Planning at *Every* Stage in Agile (SCRUM)



Source: (T2)

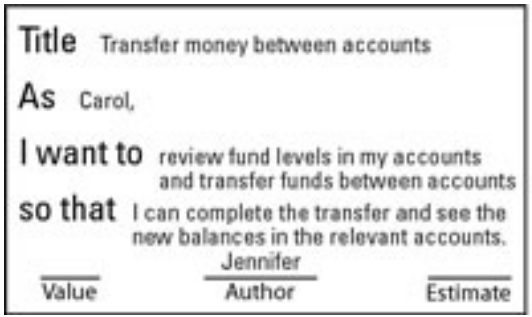
Product-backlog vs. Sprint-backlog vs. Tasks/User Stories

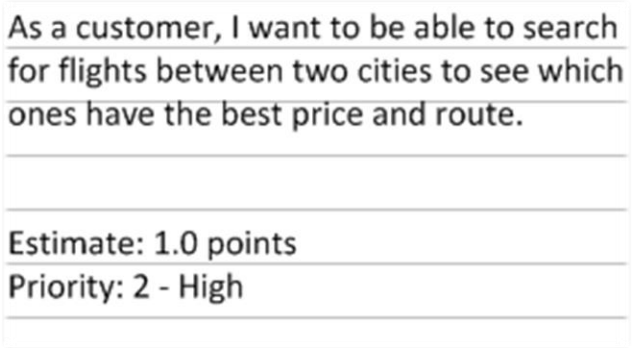


ID	Story	Type	Status	Value
121	As an Administrator, I want to link accounts to profiles, so that customers can access new accounts.	Feature	Not started	5
113	As a Customer, I want to view my account balances, so that I know how much money is currently in each account.	Feature	Not started	3
403	As a Customer,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

User Stories = Requirements for the Developer/Tester

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





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

Source: (T2-Chap8)

Sprint Planning (work backwards from Goal)

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

Example: Sprint Goal

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

Sprint Stories (for the above Goal)

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

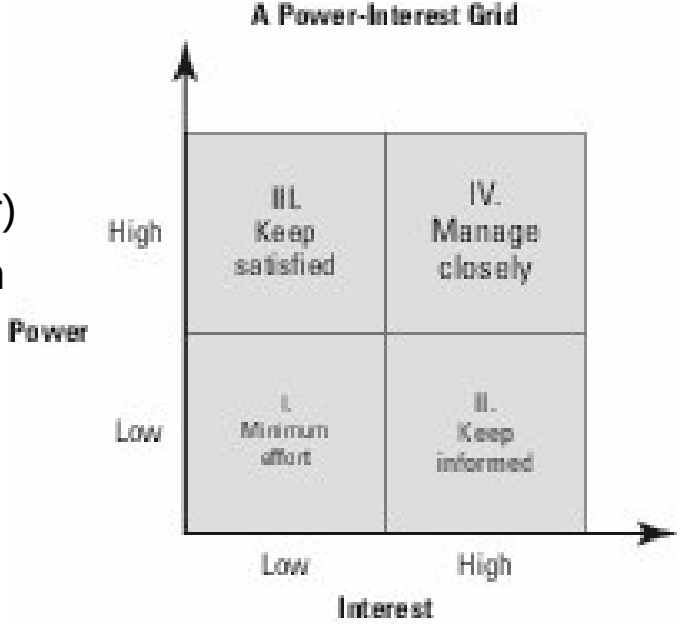
Tasks (for the above Stories)

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

Source: (T2-Chap8)

Who are the *Stakeholders* to Contribute User Stories?

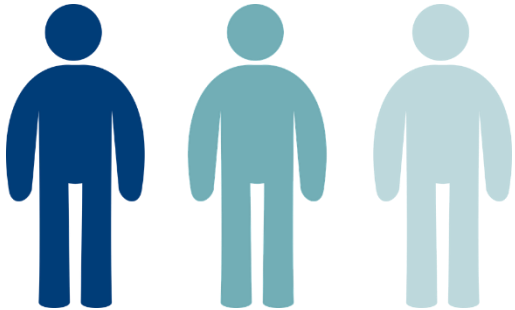
- Who Interact with End-users / Customers on regular basis (Customer Service, Marketing, Sales,…)
- Business Domain Experts
- Actual End-users
- Technical Experts (who developed such Products earlier)
- Technical Experts (when the Product need interface with other systems)
- …



Source: (T2-Chap8)

Define *Personas* (Classes of End-users)

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



Source: (T2-Chap8)

Velocity / Capacity Planning ➔➔

Creating Tasks for Sprint-backlog

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

Source: (T2-Chap8)

Velocity Planning (of Sprint)

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

Source: (T2-Chap8)

Capacity Planning in Scrum

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

Ref: https://www.cprime.com

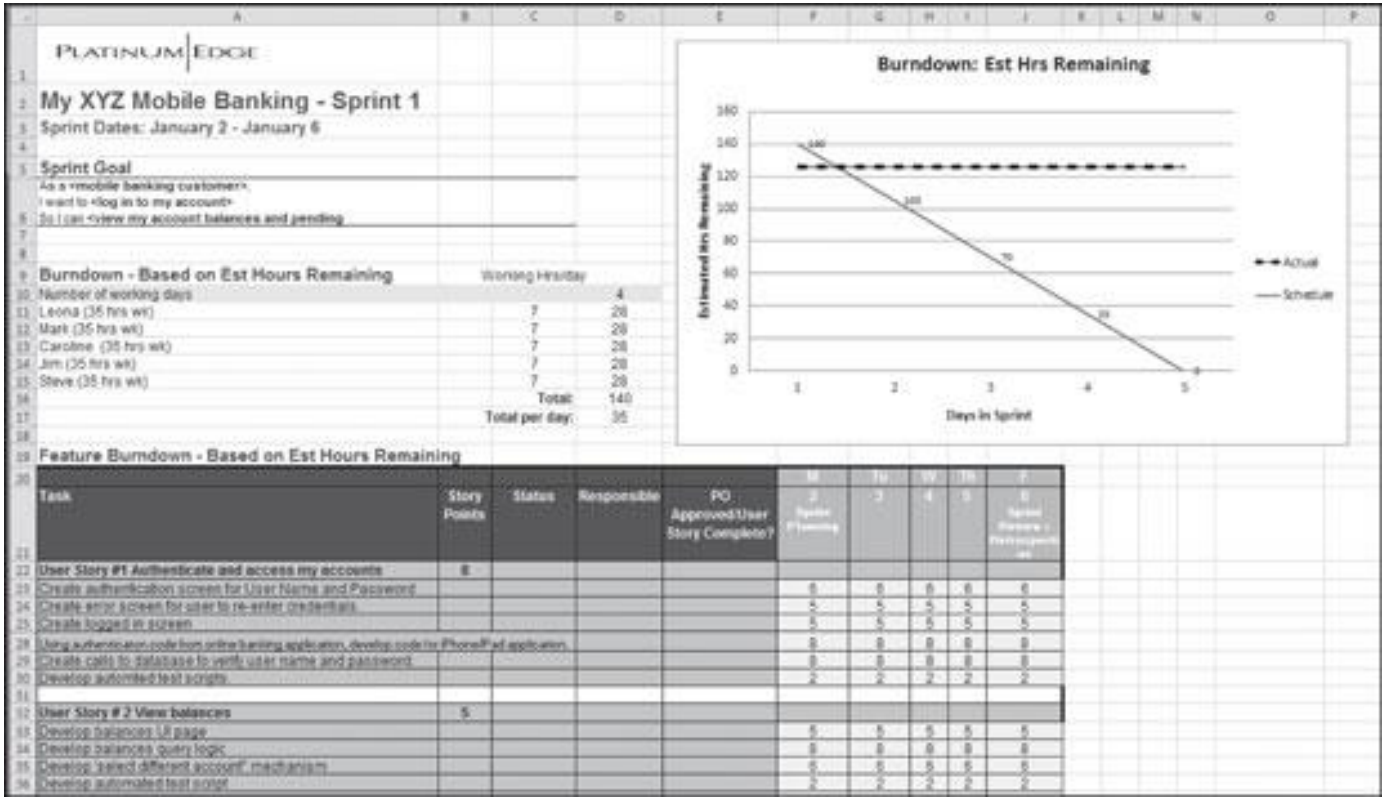
Capacity Planning (Be *Realistic*!)

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

Where:

Total Sprint Hours =	Sprint duration (weeks) * Work hours per week
Known Spent Hours =	Hours needed fpr the ceremonies, vacation, statutory holidays, planned training days, recurrent meetings or other things
Effective hours on task =	Compensation factor for actual effective hours in a day (factors in e-mail, coffee or bathroom breaks, impromptu hallway banter)

Sprint-backlog can be an Excel Sheet



PLATINUM EDGE

My XYZ Mobile Banking - Sprint 1

Sprint Dates: January 2 - January 6

Sprint Goal

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

Burndown - Based on Est Hours Remaining

	Working Hrs/day	
Number of working days		4
Leona (35 hrs wk)	7	28
Mark (35 hrs wk)	7	28
Caroline (35 hrs wk)	7	28
Jim (35 hrs wk)	7	28
Steve (35 hrs wk)	7	28
	Total:	140
	Total per day:	35

Burndown: Est Hrs Remaining

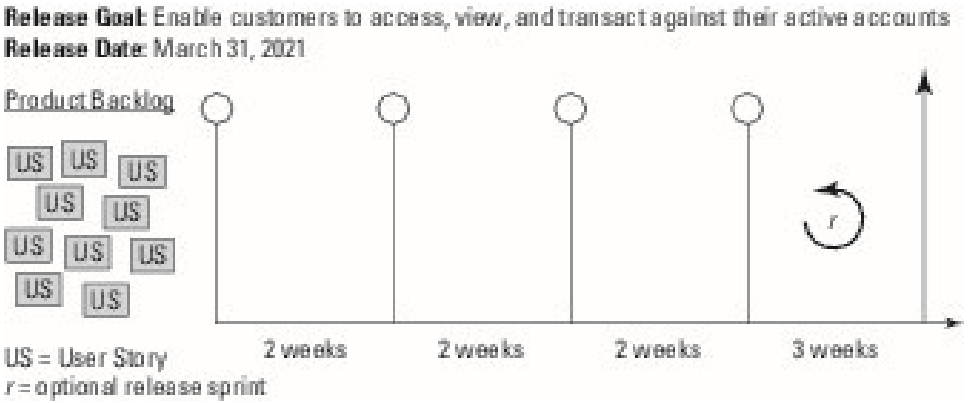
Feature Burndown - Based on Est Hours Remaining

Task	Story Points	Status	Responsible	PO Approved/User Story Complete?	1	2	3	4	5
User Story #1 Authenticate and access my accounts	8								
Create authentication screen for User Name and Password					6	6	6	6	6
Create error screen for user to re-enter credentials					5	5	5	5	5
Create logged in screen					5	5	5	5	5
Using authentication code from online banking application, develop code for iPhone/iPad application					8	8	8	8	8
Create calls to database to verify user name and password					8	8	8	8	8
Develop automated test scripts					2	2	2	2	2
User Story # 2 View balances	5								
Develop balances UI page					5	5	5	5	5
Develop balances query logic					8	8	8	8	8
Develop 'select different account' mechanism					5	5	5	5	5
Develop automated test script					2	2	2	2	2

Source: (T2-Chap8)

Release Sprint Planning ➔➔

Release Planning



Source: (T2-Chap8)

No Separate ‘Release-backlog’ Recommended

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

➔ The product backlog and release plan are some of the most important communication channels between the product owner and the development team.

Source: (T2-Chap8)

Release Sprint? Consider Tasks Too *Heavy* for one Regular Sprint…

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

Source: (T2-Chap8)

Release Plan, Release Schedule

- The Product-backlog and Release Plan are some of the most important communication channels between the product owner and the development team
- The Release Plan contains a Release Schedule for a specific set of Features. The Product Owner creates a Release Plan at the start of each Release. Creating a release plan involves:

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

Source: (T2-Chap8)

Just-in-Time Planning of Releases

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

Source: (T2-Chap8)

Summary: Iteration Planning

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

Thank You

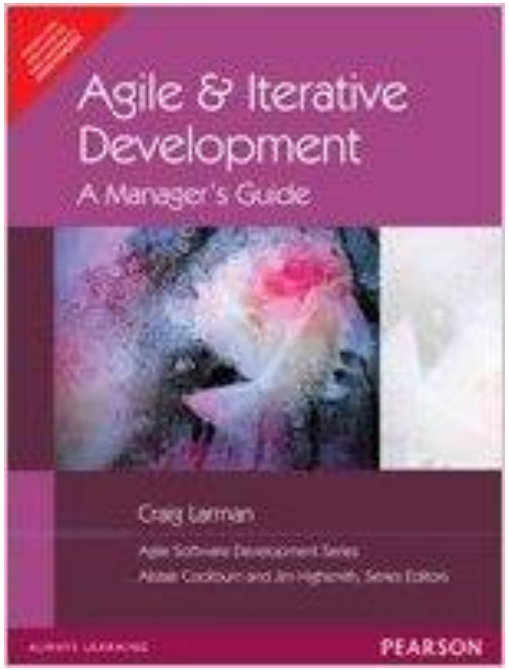
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Executing a Sprint

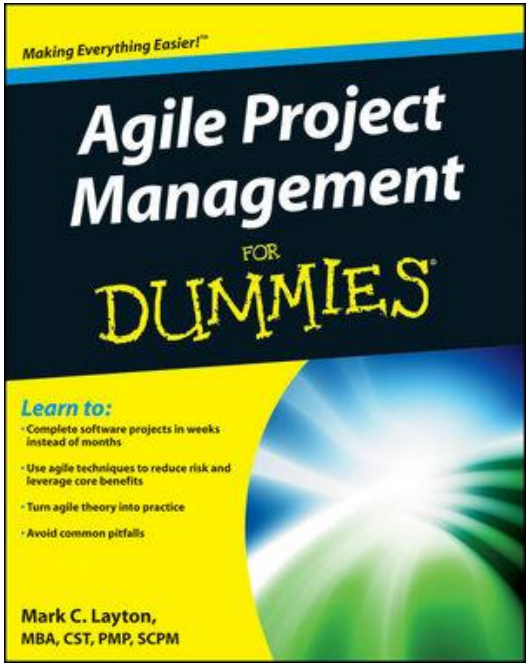
- Prof K G Krishna

Text/Reference Books

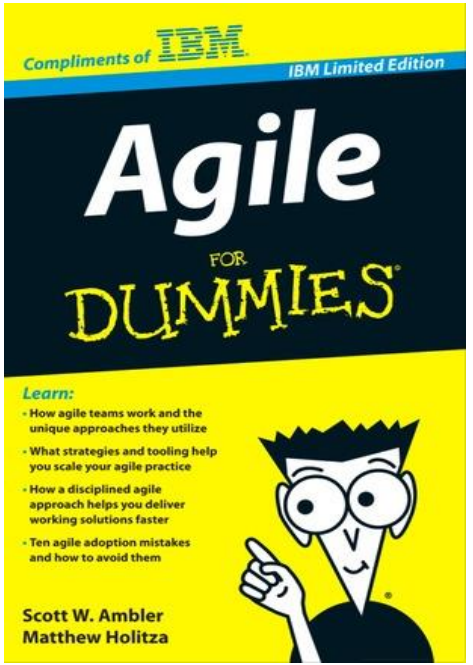
T1



T2



Compliments of IBM



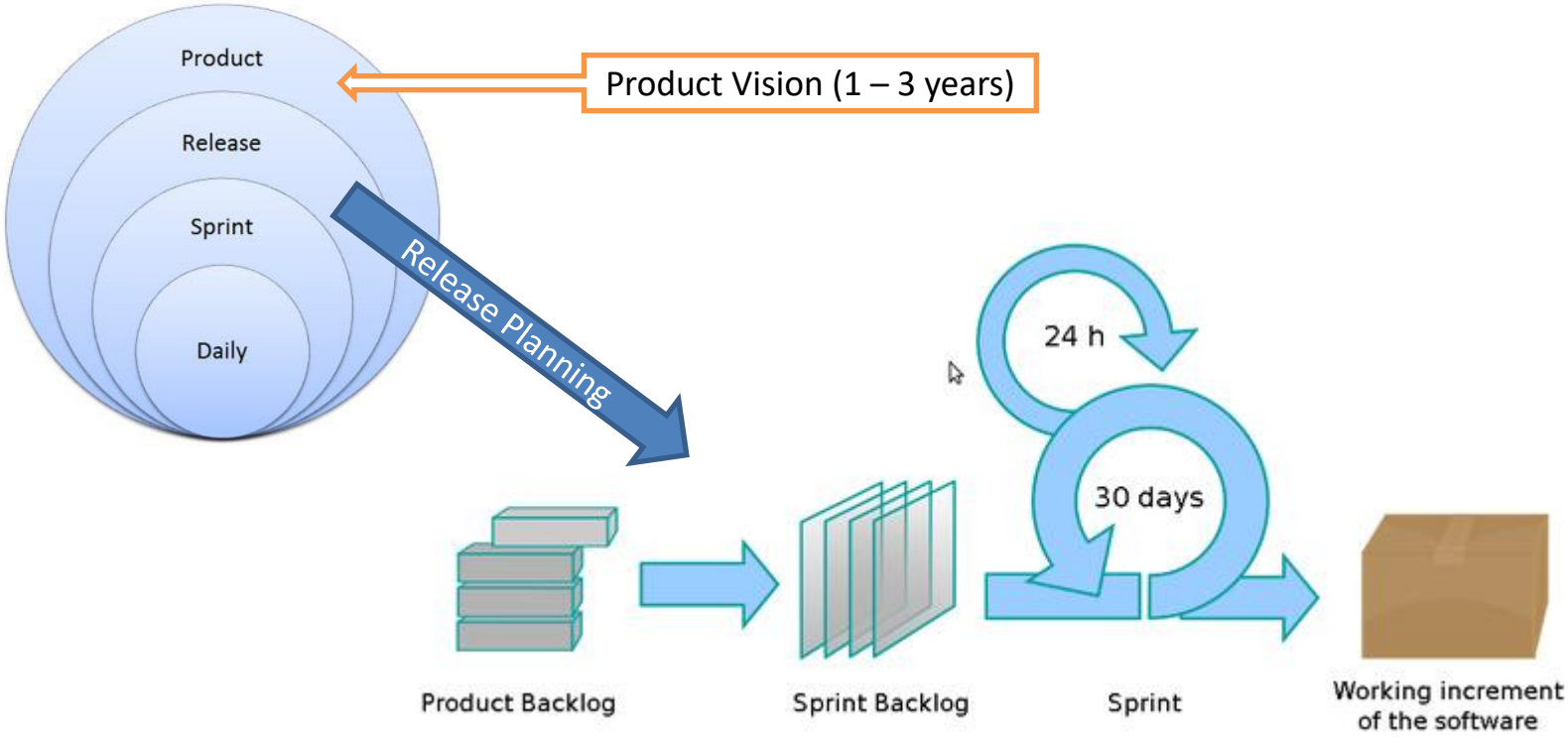
➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

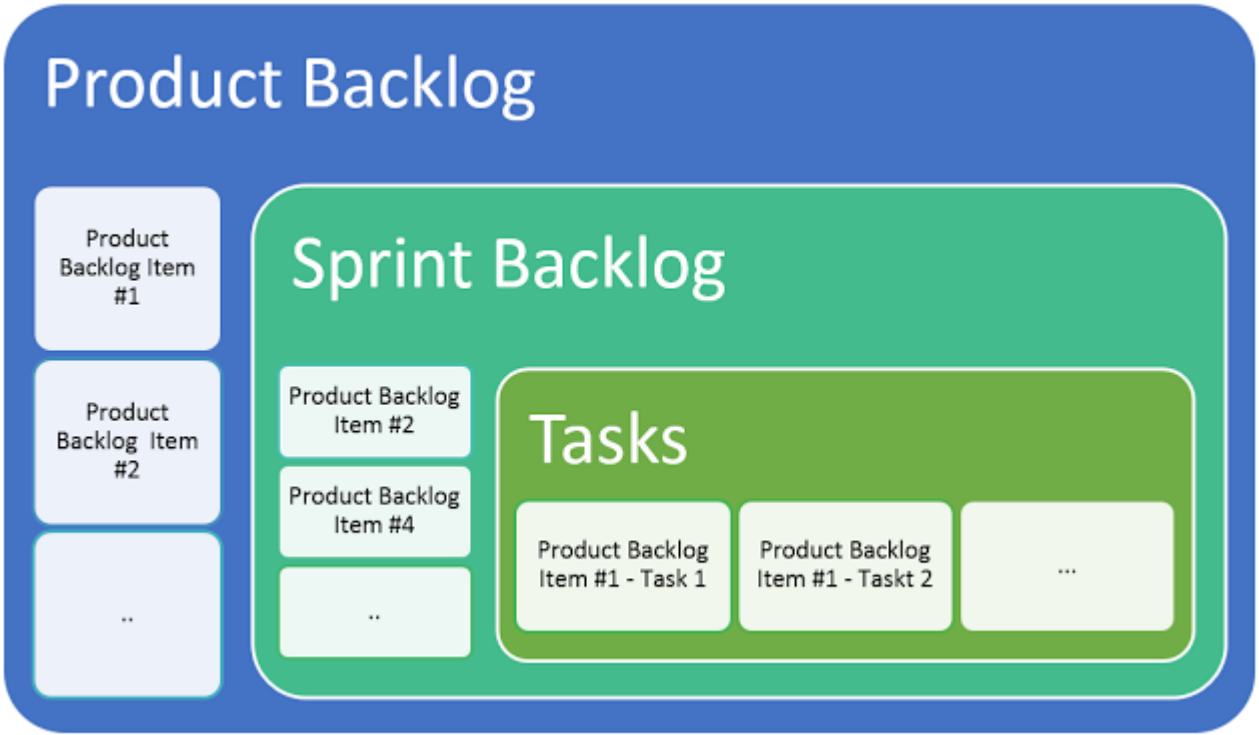
Executing a Sprint

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

Levels of Planning in Agile



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



Scrum: Roles – Artifacts - Ceremonies

Roles

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

Team Members
- Who will do it and how?

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

Artifacts

Product Backlog
- What are we doing and in what order?

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

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

Ceremonies

Sprint Planning
- What are we doing next?

Daily Scrum
- How are we doing today?

Sprint Review
- How did we do?

Sprint Retrospective
- How do we get better?

Sprint Ceremonies (Meetings) – A Snapshot

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

Sprint-backlog – an Overview

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

Ref: (T2-Chap9)

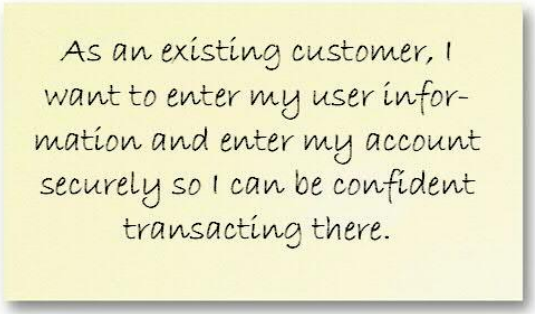
Daily Scrum Meeting

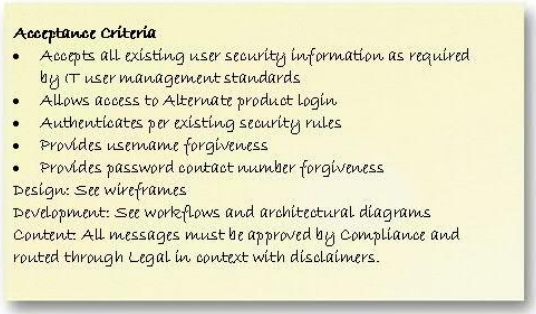
- Scrum meeting is a Stand-up meeting lasting not more than 15-20 minutes
- Anyone may attend a daily scrum, but only the development team, the scrum master, and the product owner may talk. Stakeholders can discuss questions with the scrum master or product owner afterward, but stakeholders should not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

Ref: (T2-Chap9)

Executing Sprint

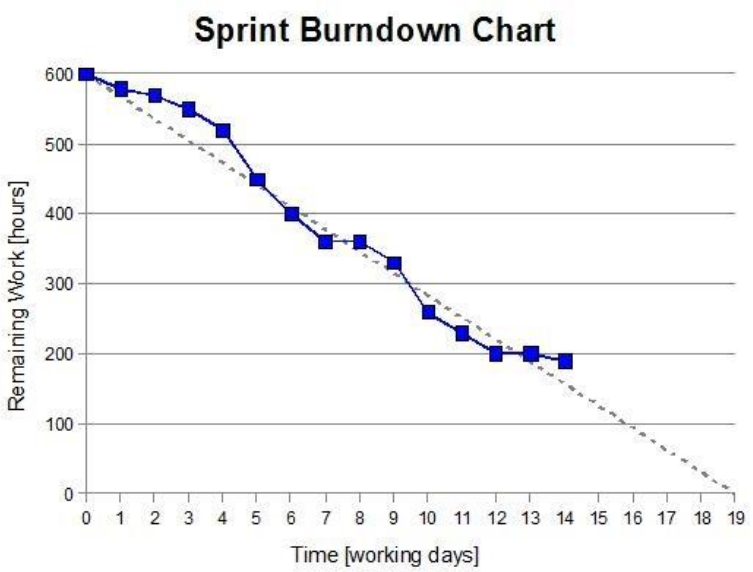
- Creating Shippable Functionality
- The objective of the day-to-day work of a sprint is to create shippable functionality for the product in a form that can be delivered to a customer or user.
- To create shippable functionality, the development team and the product owner are involved in following major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

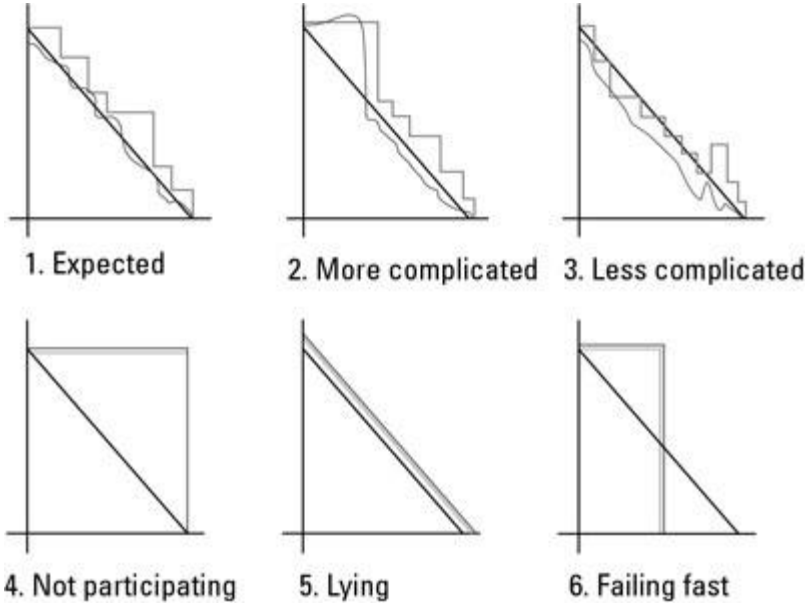




Ref: (T2-Chap9)

Tracking Progress: Sprint Burndown Chart

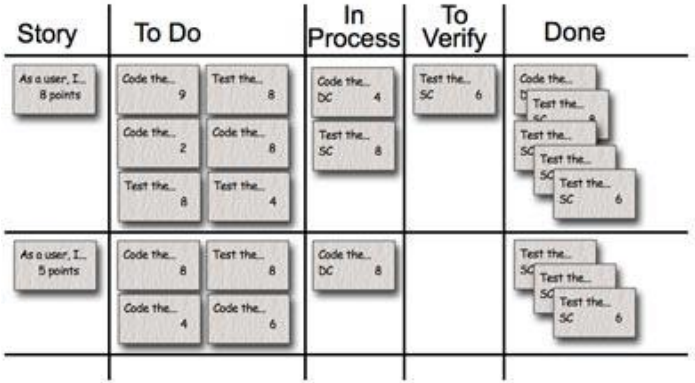




Ref: (T2-Chap9)

Tracking Progress: Sprint *Task-board*

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



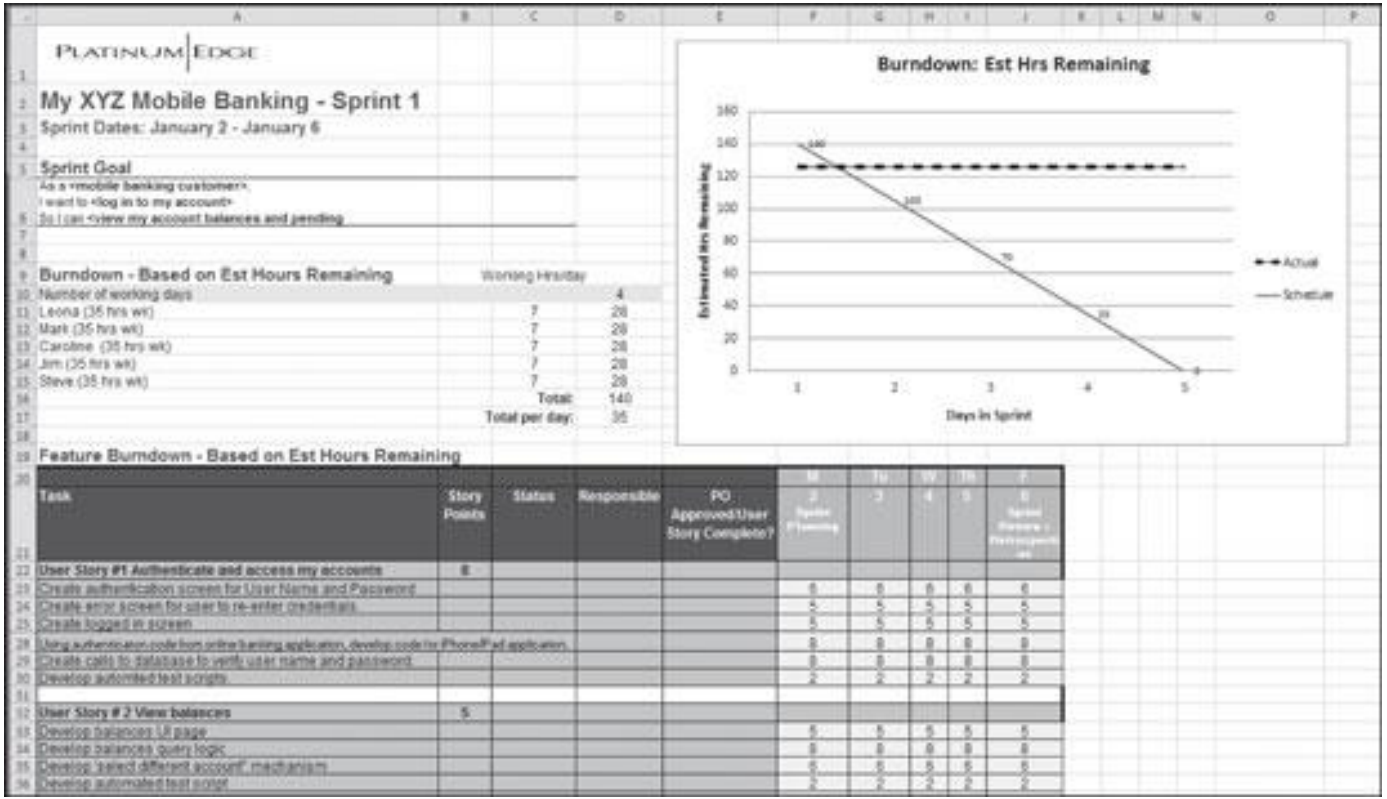
Ref: (T2-Chap9)

Progress Tracking through Burndown Charts

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

Ref: (T2-Chap9)

Sprint-backlog Example



PLATINUM EDGE

My XYZ Mobile Banking - Sprint 1

Sprint Dates: January 2 - January 6

Sprint Goal

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

Burndown - Based on Est Hours Remaining

	Working Hrs/day	
Number of working days		4
Leona (35 hrs wk)	7	28
Mark (35 hrs wk)	7	28
Caroline (35 hrs wk)	7	28
Jim (35 hrs wk)	7	28
Steve (35 hrs wk)	7	28
	Total:	140
	Total per day:	35

Burndown: Est Hrs Remaining

Feature Burndown - Based on Est Hours Remaining

Task	Story Points	Status	Responsible	PO Approved/User Story Complete?	M	Tu	W	Th	F
User Story #1 Authenticate and access my accounts	8								
Create authentication screen for User Name and Password					6	6	6	6	6
Create error screen for user to re-enter credentials					5	5	5	5	5
Create logged in screen					5	5	5	5	5
Using authentication code from online banking application, develop code for iPhone/iPad application.					8	8	8	8	8
Create calls to database to verify user name and password					8	8	8	8	8
Develop automated test scripts					2	2	2	2	2
User Story # 2 View balances	5								
Develop balances UI page					5	5	5	5	5
Develop balances query logic					8	8	8	8	8
Develop 'select different account' mechanism					6	6	5	5	5
Develop automated test script					2	2	2	2	2

Source: (T2-Chap9)

Sprint-backlog – An Example

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

Scaling Agile: Scrum of Scrums

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

Ref: (T2-Chap9)

Scrum of Scrums (contd.,)

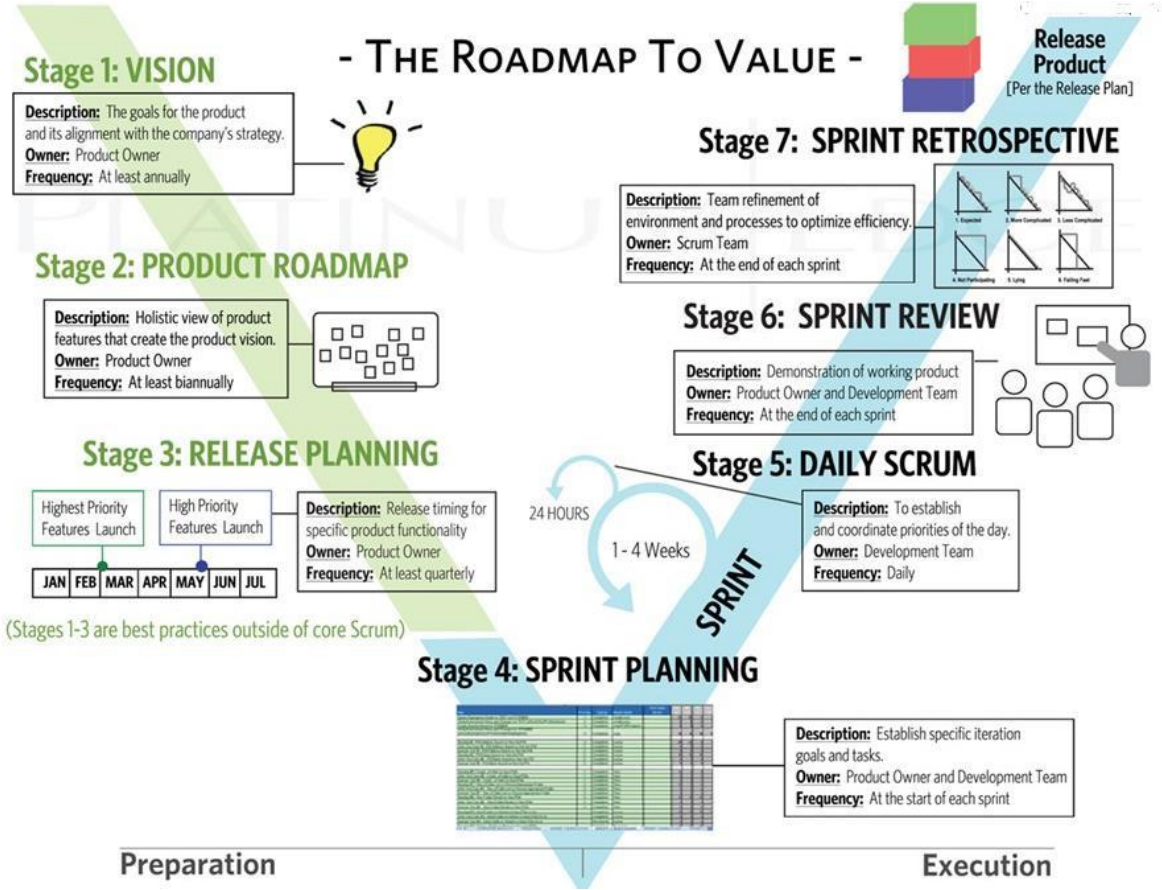
- The scrum of scrums meetings can be scaled up in a recursive manner – more scrum of scrums meetings. Each contains a representative from each of the teams. The work of scrum of scrums meetings can be coordinated through an even higher level meeting (“scrum of scrum of scrums”)
- Scrum of scrums meetings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

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

Ref: (T2-Chap9)

Sprint Reviews & Sprint Retrospectives

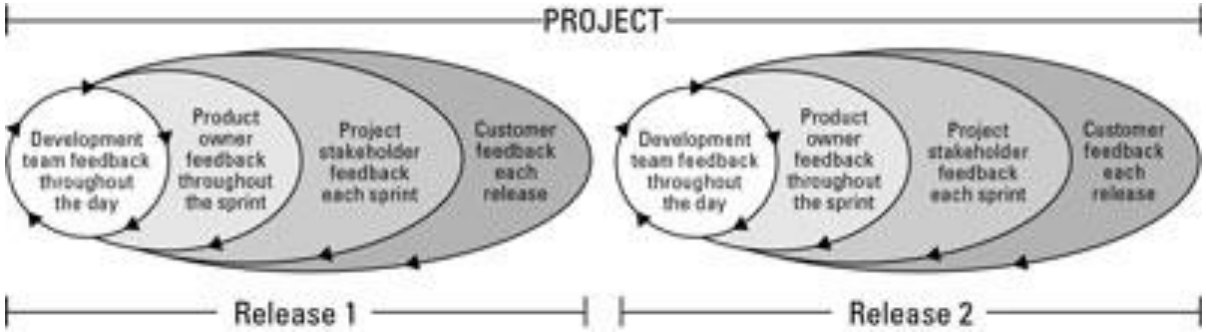
Every Stage in the SCRUM is a Review Stage!



Source: (T2)

Sprint Review

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



Ref: (T2-Chap9)

Sprint Review Meeting: Guidelines

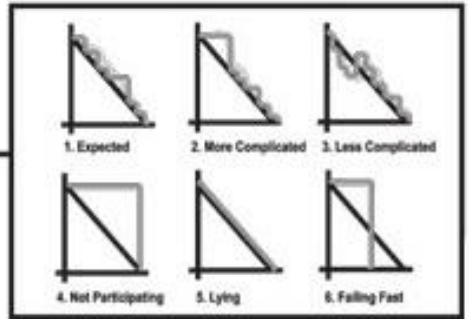
If my sprint is this long...	My sprint review meeting should last no more than...
One week	45 minutes
Two weeks	1.5 hours
Three weeks	2.25 hours
Four weeks	Three hours

Ref: (T2-Chap9)

Sprint Retrospective (for Process Improvement)

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

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



Ref: (T2-Chap9)

Summary: Executing a Sprint

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

Thank You

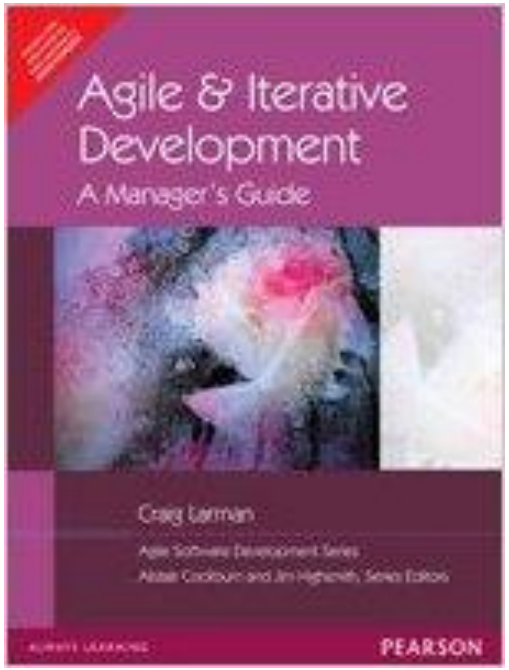
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Metrics & Tools

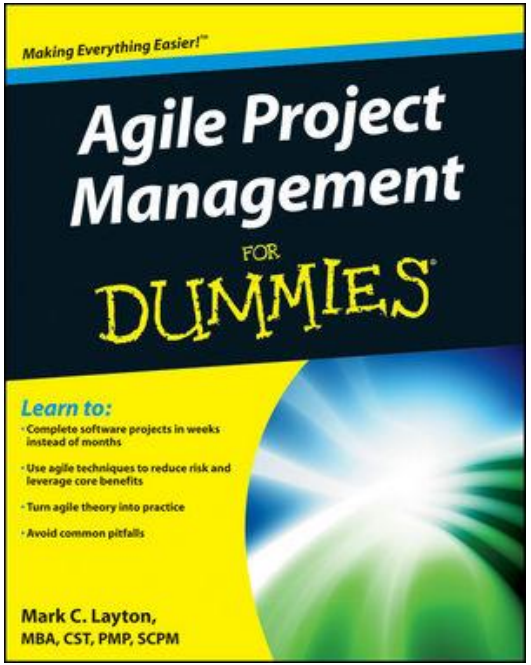
- Prof K G Krishna

Text/Reference Books

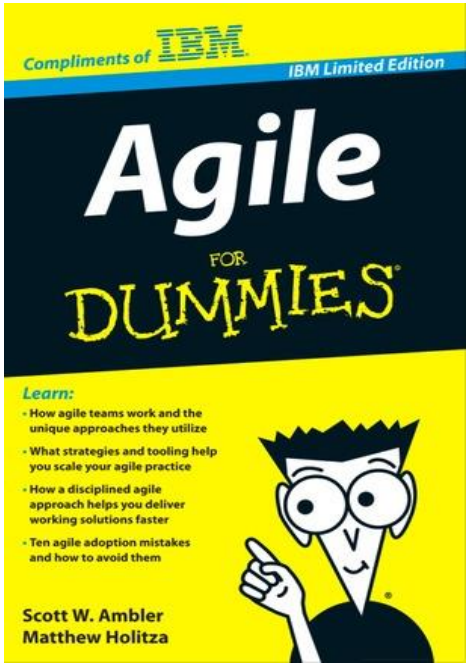
T1



T2



Compliments of IBM

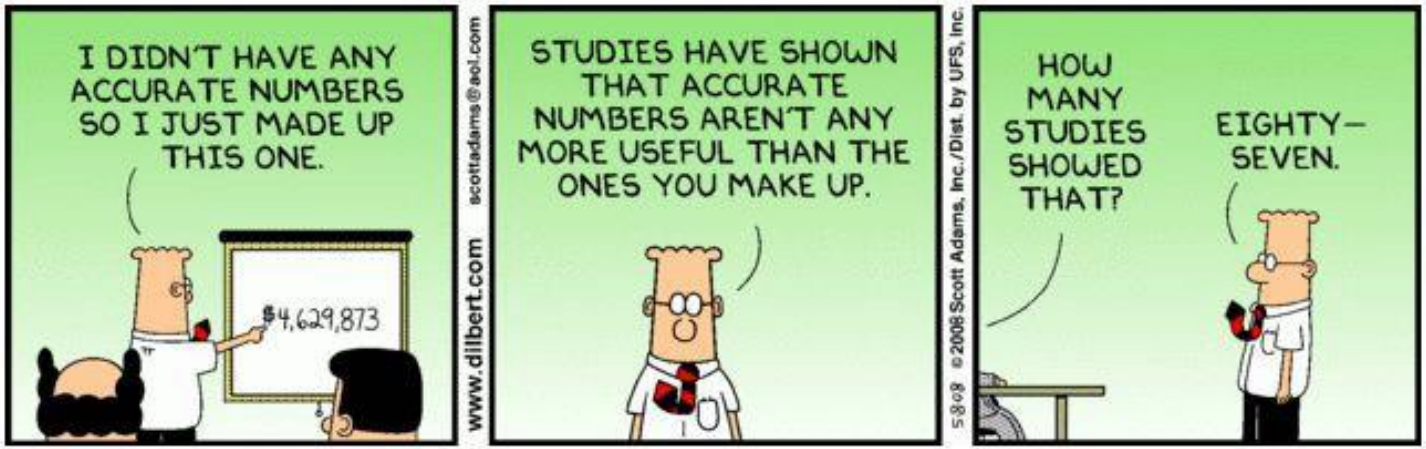


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

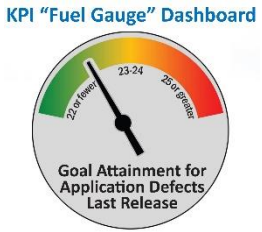
Agile Metrics & Tools

- Overview of Agile Metrics
- Tools for Metrics



Metrics – Track Performance / Progress

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



Metrics must be Actionable (➔KPIs)

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

Source: appdevelopermagazine.com

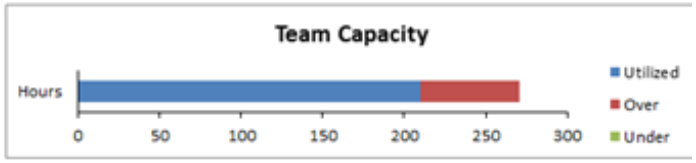
Ten *Key Metrics* for Agile Project Management

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

Sprint Estimation Metrics

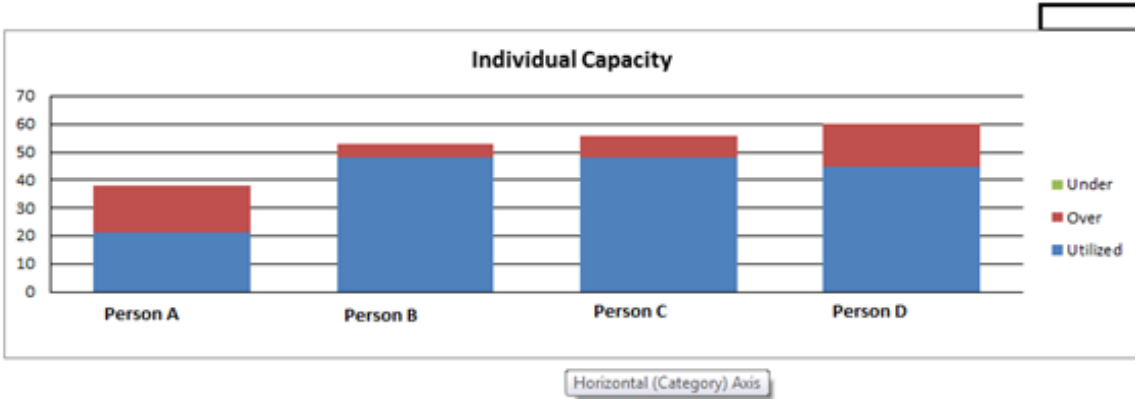
Team Capacity

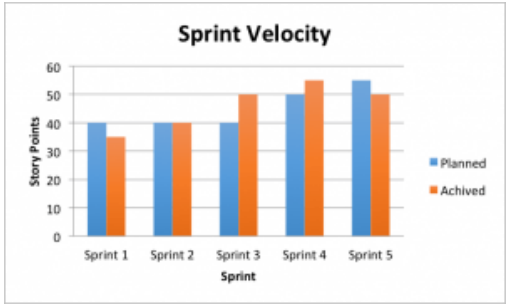
Totals	Hours
Remaining Work	271
Remaining Capacity	210
Utilized	210
Over	61
Under	0



Individual Capacity

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

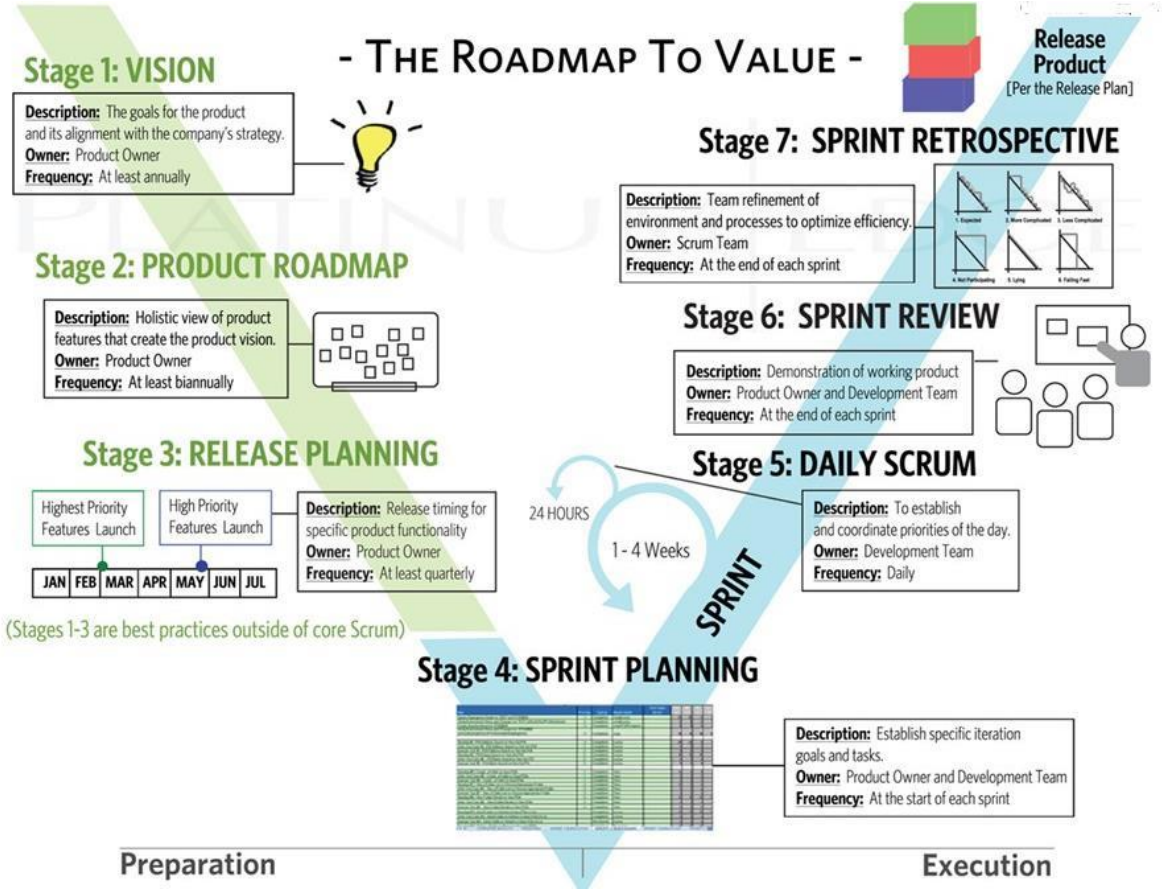




Source: blogs.technet.microsoft.com

Tools for Agile Management ➔➔

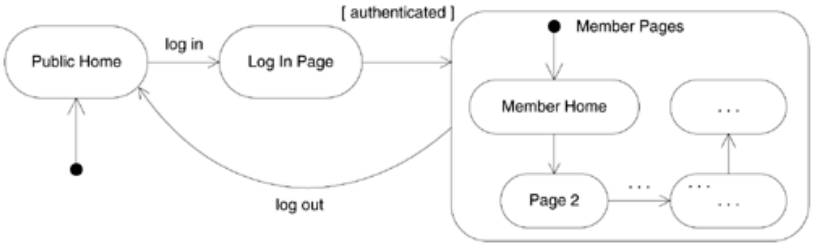
Metrics/Tools at every Stage of Agile Life-cycle



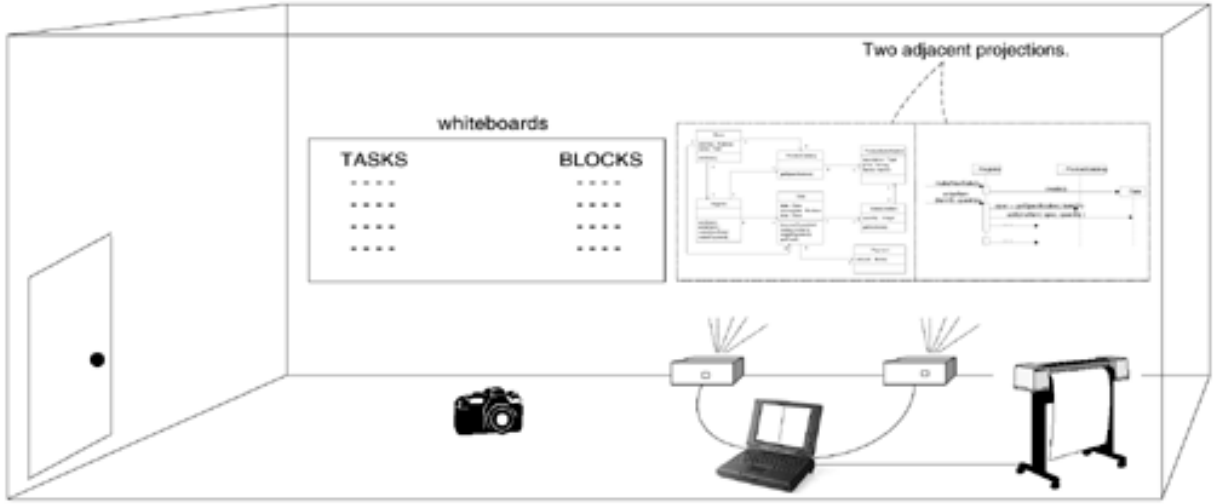
Source: (T2)

Tools for Agile Project/Knowledge Management

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

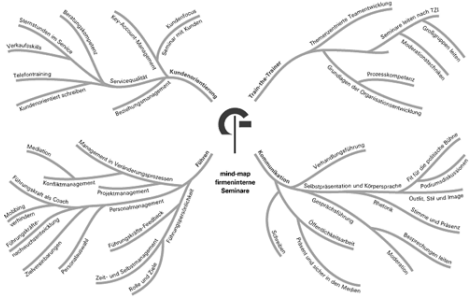


Sample *XP Room* for Collaboration, Communication and Brainstorming



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

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



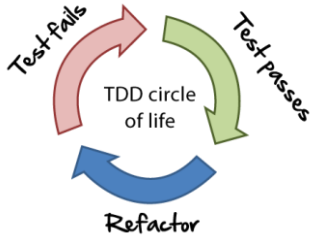
Source: (T1)

Summary: Agile Metrics & Tools

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

Thank You

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

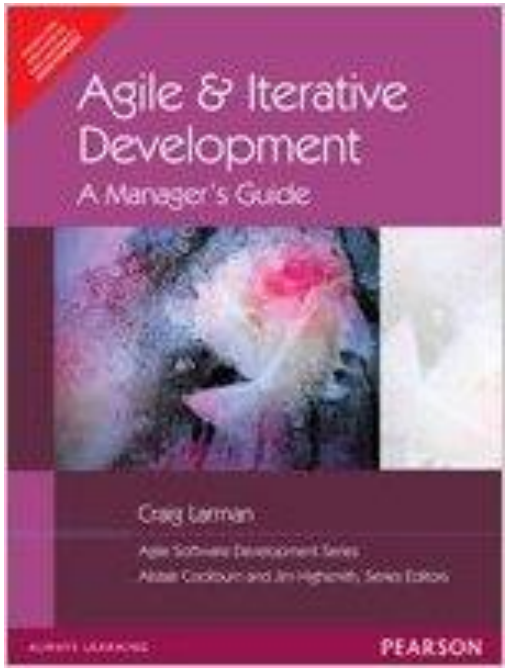


Quality Management in Agile

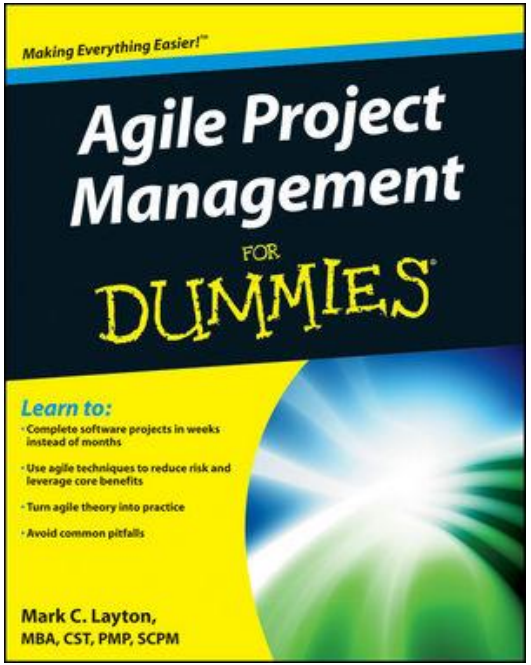
- Prof K G Krishna

Text/Reference Books

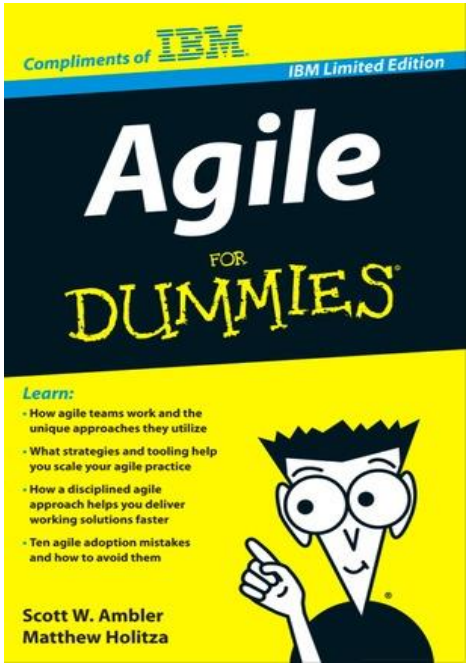
T1



T2



Compliments of IBM



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

Topics

Quality Management in Agile

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

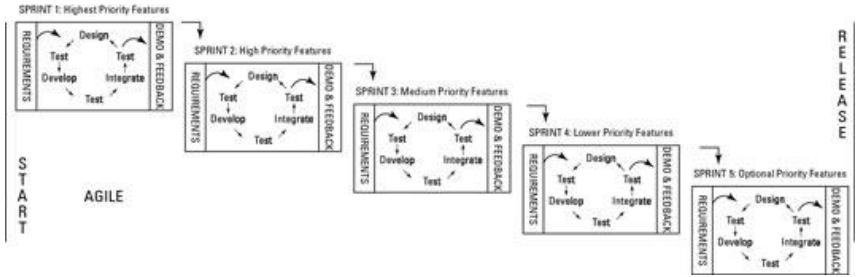
Ensuring Quality: Traditional vs. Agile

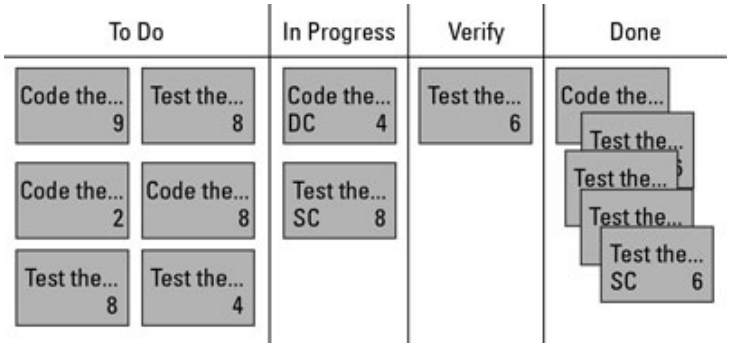
- Testing is the Last Phase of the Project vs. Daily Activity as part of Sprint
- Reliance of Manual Testing (unit) vs. Automated Test Tools (necessary)
- Reactive (fixing discovered bugs/issues) vs. Focus on Proactive practices (Pair Programming, Coding Standards, and Face-to-Face Communication)
- Risk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

Ref: (T2-Chap 15)

Quality is Integrated into Agile Methods

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

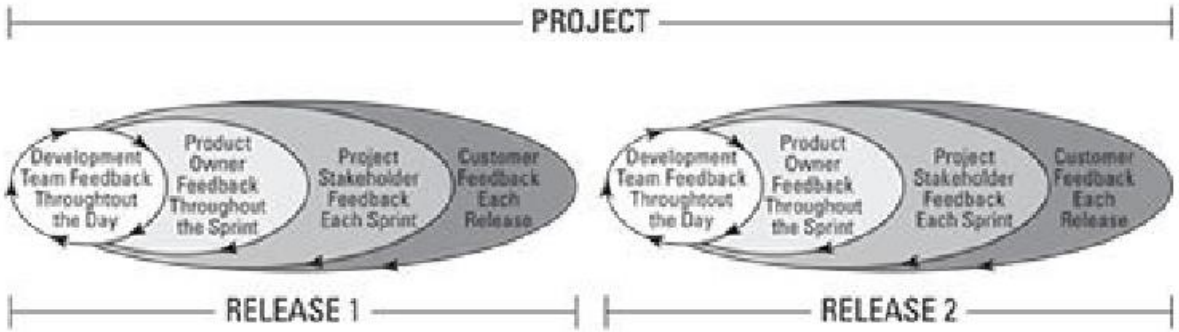




Ref: (T2-Chap 15)

Multiple *Feedback Loops* Feeding Quality

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



Ref: (T2-Chap 15)

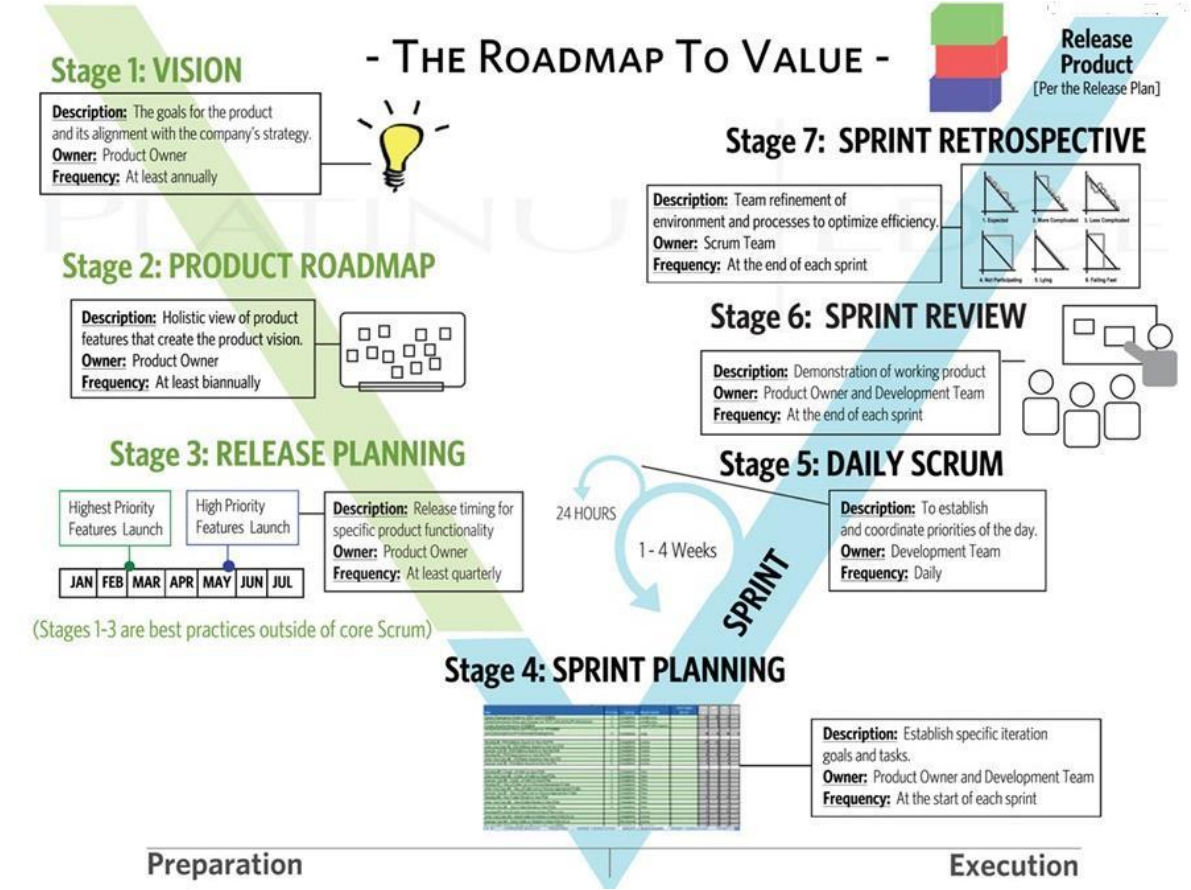
Agile Development Methods emphasize Quality

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

Ref: (T2-Chap 15)

Managing Risks in Agile ➔➔

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



Source: (T2)

Managing Risk in Agile Projects

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

➔ "Failing-fast" early subsumes Risk in Agile Projects by prioritizing High-value and High-risk Requirements during early Sprints

Ref: (T2-Chap 15)

Managing Risk: Traditional vs. Agile

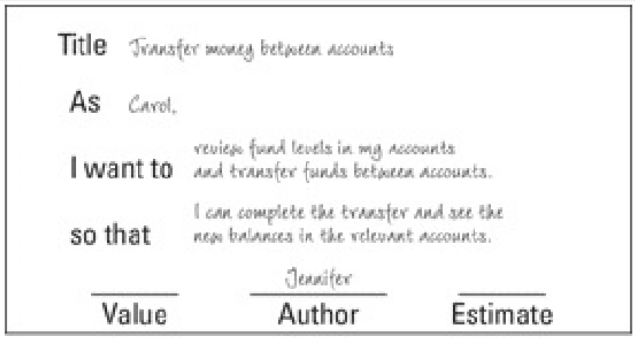
- Large Projects are challenged with Overruns and Scope-creep and nothing-to-show till the end vs. Catastrophic failures are eliminated with 'something-to-show' at every stage
- Conducting Testing (ST/UAT) at the end means finding serious problems pose grave risk to the project vs. Continuous testing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

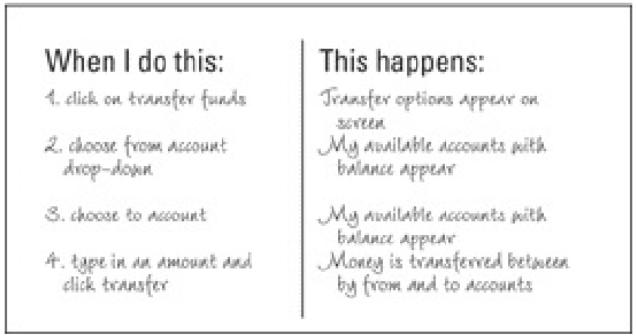
Ref: (T2-Chap 15)

.

Risk Management Artifacts/Meetings in Agile

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





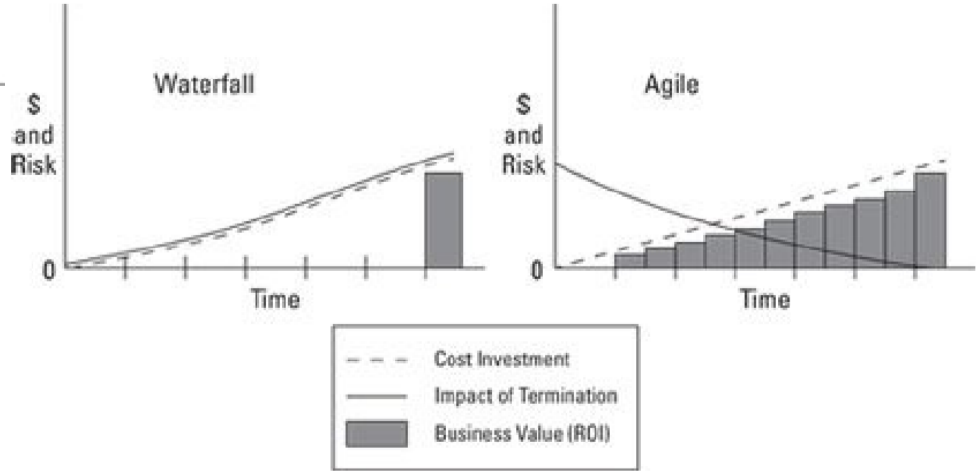
Ref: (T2-Chap 15)

Risk Mitigation is Inherent in Agile

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

Definition of Done

Sprint	Release	Risks Accepted
QA Environment Unit Tested Functional Tested Integration Tested User Acceptance Tested Regression Tested XDocs	Staging Environment Performance Tested Security Tested Enterprise System Integrated Focus Group Tested User Documentation Training Documentation	Load Testing



Ref: (T2-Chap 15)

Summary: Quality and Risk Management in Agile

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

Thank You

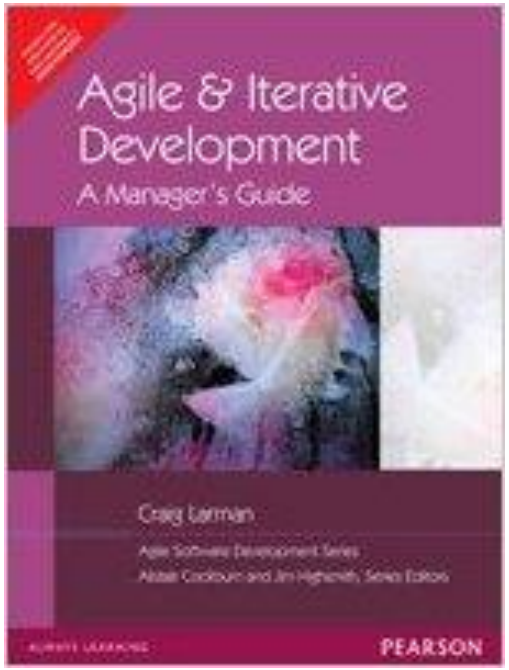
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Agile Pitfalls

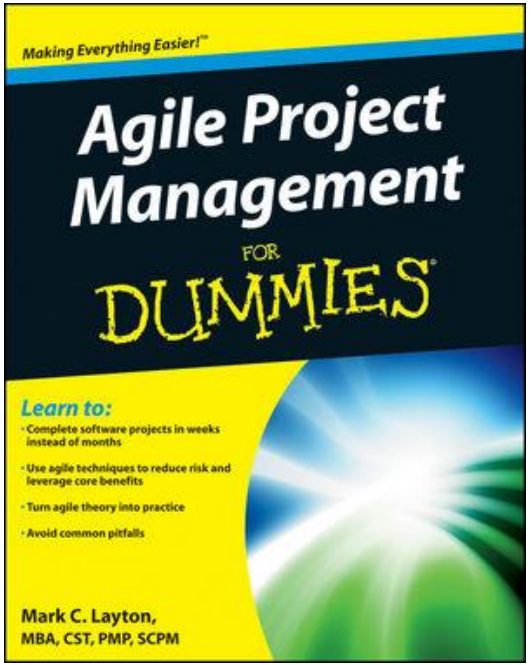
- Prof K G Krishna

Text/Reference Books

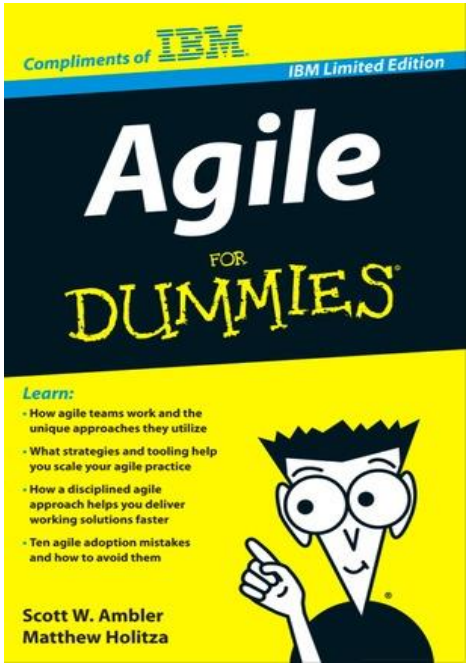
T1



T2



Compliments of IBM

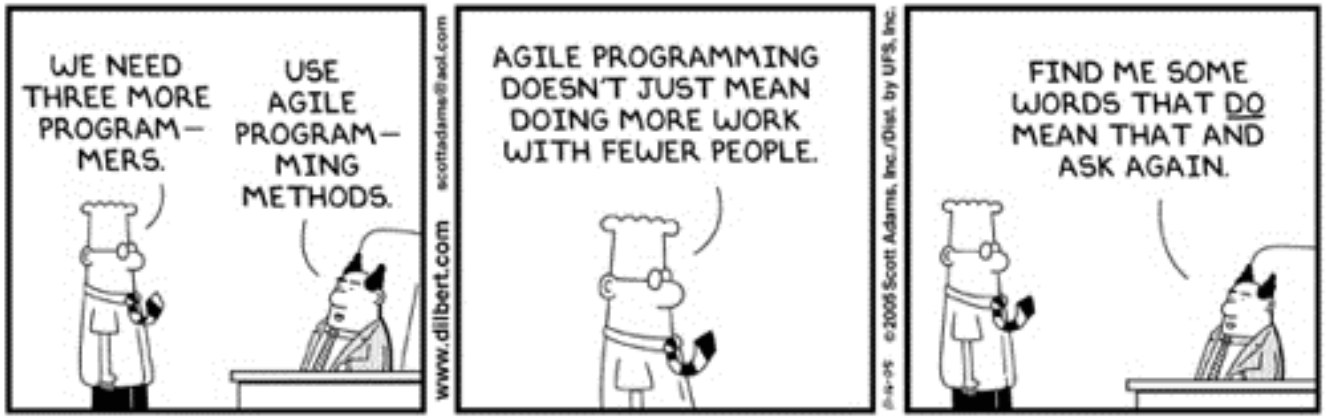


➔ As this field is evolutionary, the student is advised to stay tuned to the current and emerging practices by referring to their own organization's documentation as well as Net sources

Topics

Agile Myths & Pitfalls

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



Common mistakes/misunderstandings

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

  “…We have adopted iterative method for two months and finished use case analysis and plan and schedule of what we’ll be doing in each iteration. After review and approval of final requirements set and iteration schedule, we’ll start programming…”

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

Common mistakes/misunderstandings *(contd.,)*

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

Common mistakes/misunderstandings *(contd.,)*

- Not integrating the QA Team
  - Many organizations have a separate Quality Assurance team, used to having a completed system “thrown over the wall” to them.
- Post development design is wrong
  - XP isn't anti-documentation, but prefers programming if that's sufficient to succeed. XP can support the creation of design documentation to reflect the existing code base, once the program is complete
- In Pair programming, not willing to learn or explain
  - For successful pair programming, an attitude of openness to learning and of explaining yourself is required**.**
- Iterations are NOT Time-boxed
  - It is a misunderstanding to let the iteration length expand when it appears the goals can't be met within the original time frame--rather, the preferred strategy is usually to remove or simplify goals for the iteration and analyze why the estimates were off the mark
- Each iteration ends in a production release
  - The baselined software produced at the end of an iteration is an internal release rather than shippable code. It represents a subset of the final production release, which may only be ready after a dozen or more short iterations
- Predictive Planning
  - It is a misunderstanding to create, at the start of the project, a believable plan laying out exactly how many iterations there will be for a long project, their lengths, and what will occur in each. This is contrasted with the agile approach: adaptive planning.

Agile Myths in a Nutshell…

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

Agile Pitfalls/Challenges

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

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

Predictive Planning vs. Adaptive Planning (Agile)

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

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

Taiichi Ono
Father of Lean (Toyota Production System)

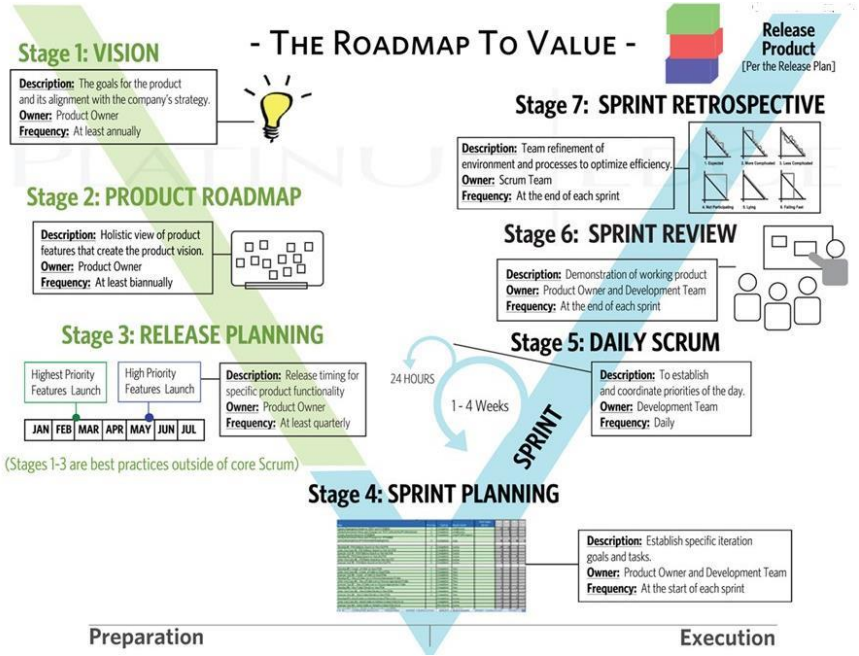
Planning: Waterfall vs. Iterative (Predictive Planning) vs. Agile (Adaptive Planning)

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

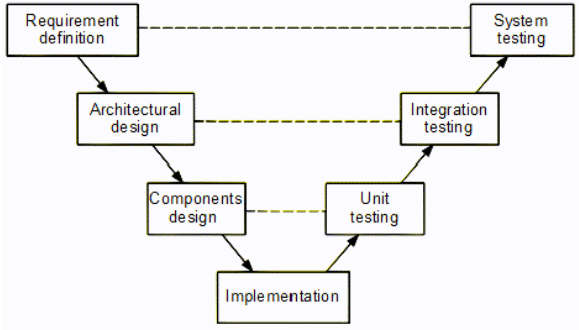
The ‘Big’ Cultural Differences between CMMI and Agile

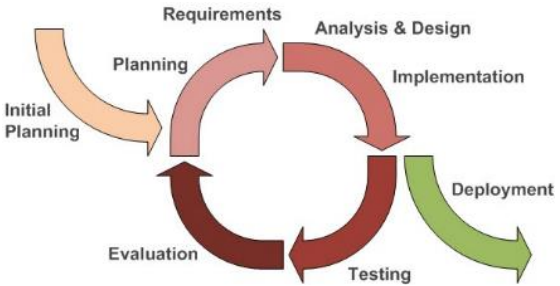
	CMMI	Agile
Application	Process improvements	Software development
Focus	Existing processes	New processes and products
Main goal	Organizational improvements	Shippable product
Management processes (risk, quality, etc)	Standardized	Not included

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



≠





Source: (T2)

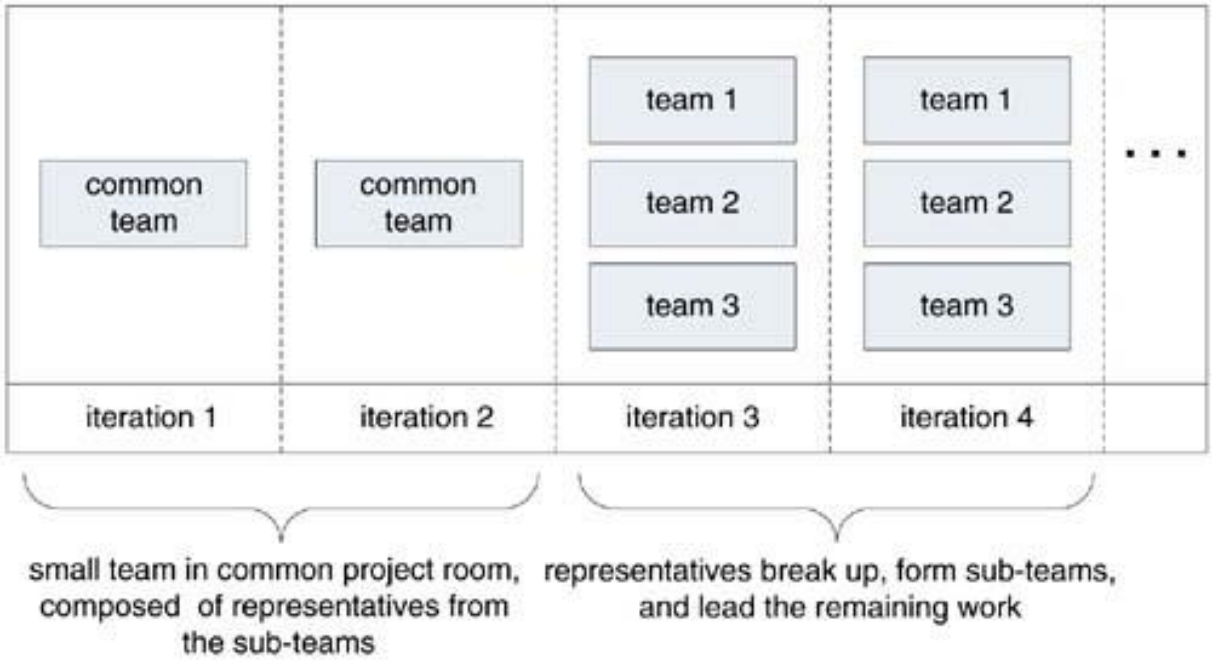
Distributed Agile (Multisite/Multiteam) ➔➔

Multisite / Multiteam Planning

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

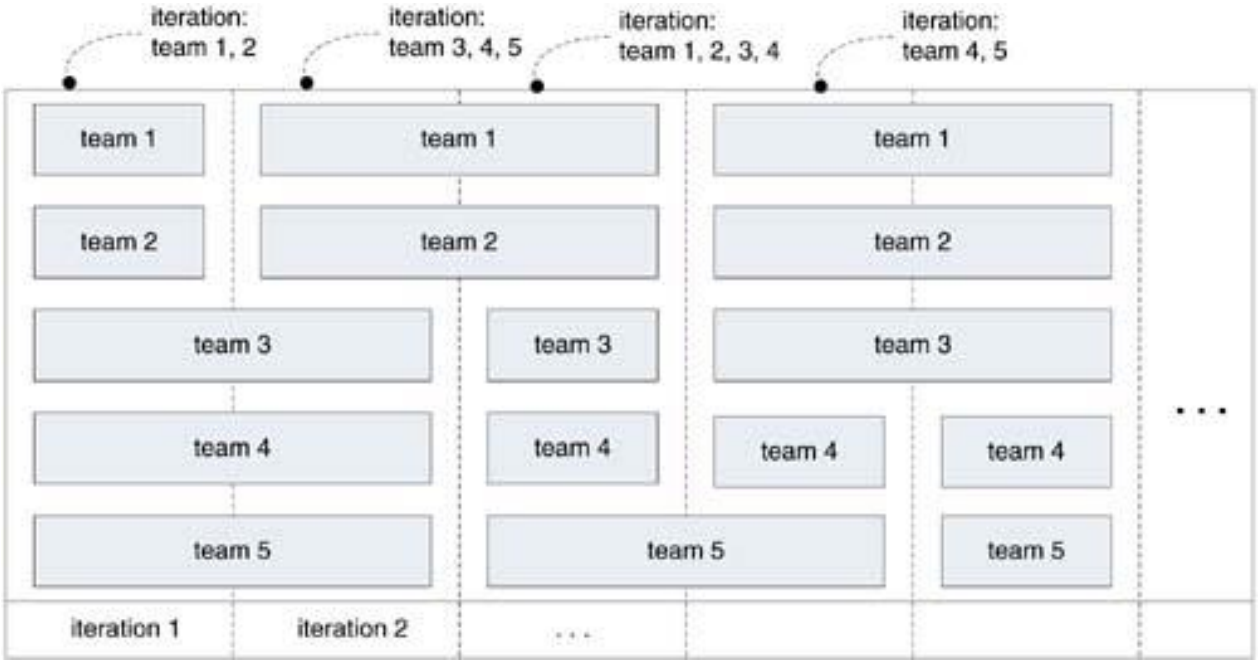
Multiteam Development

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



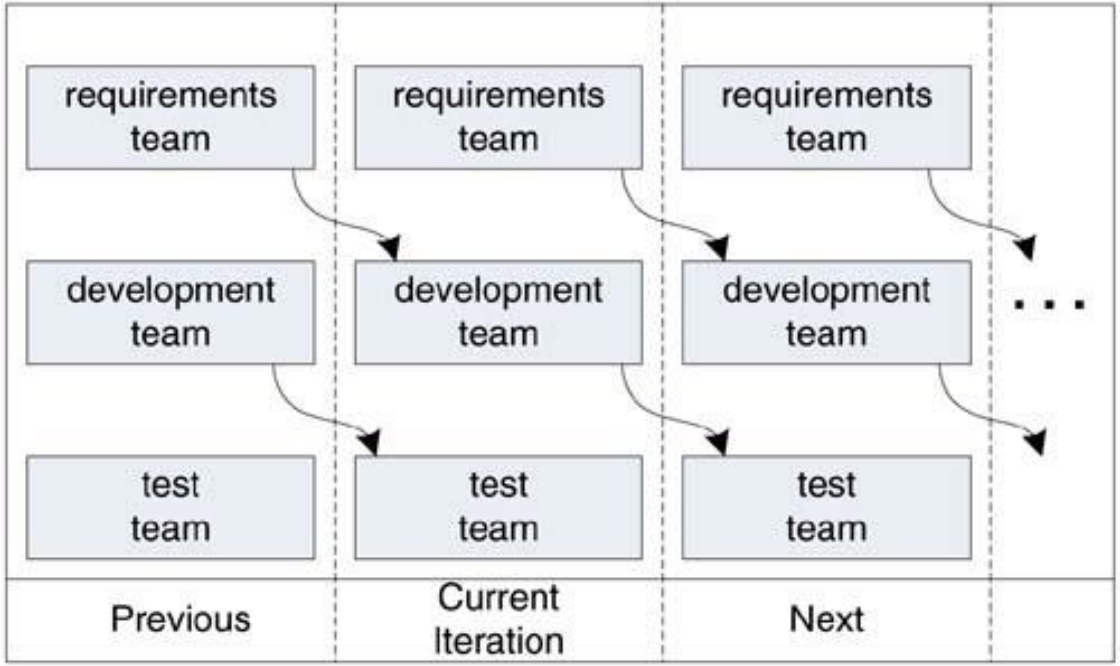
Subteam Iterations (for R&D Projects)

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



Pipelining Iterations (for large projects)

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



Summary: Agile Myths – Debunked / Major Challenges

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

Thank You

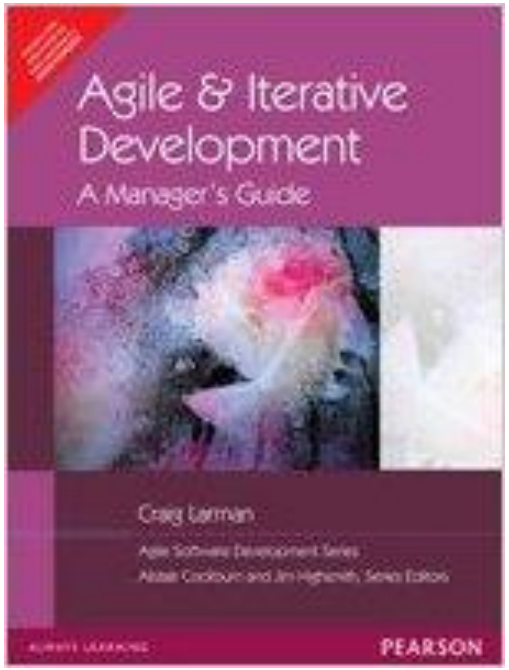
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Ensuring Agile Success

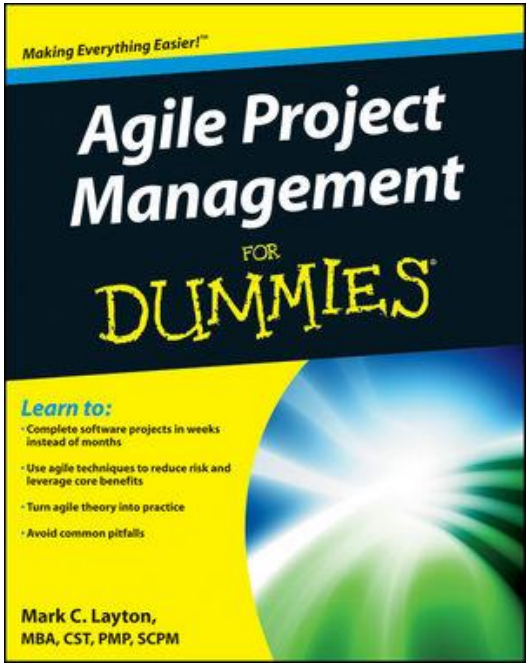
- Prof K G Krishna

Text/Reference Books

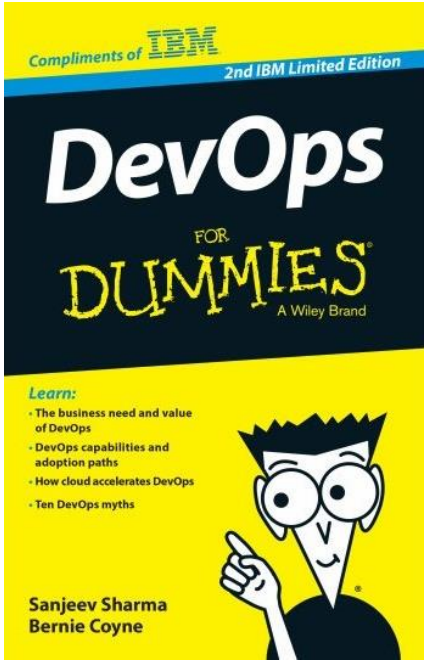
T1



T2



Compliments of IBM



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

Topics

Ensuring Agile Success

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

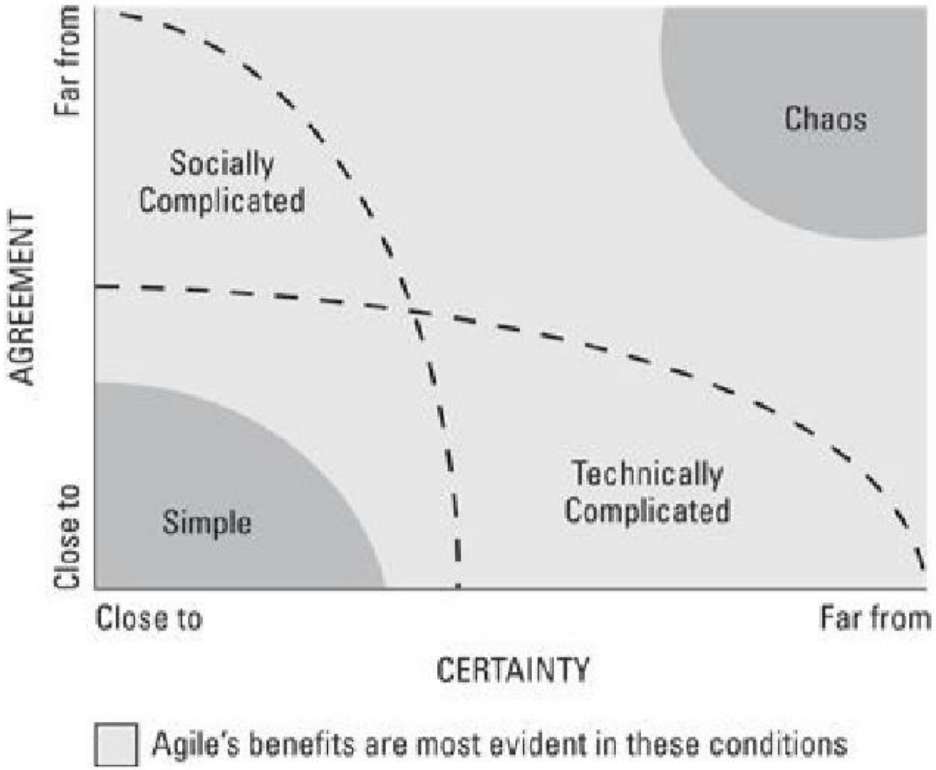
10 Key Factors for Agile Success

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

Getting Commitment to Agile Transition

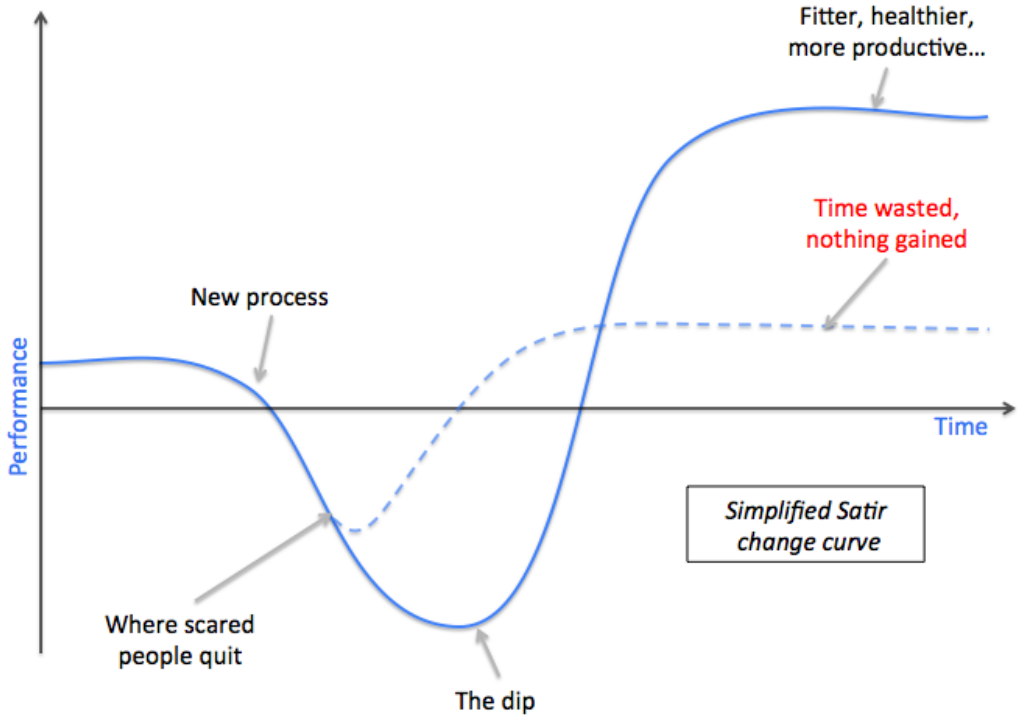
- **Identify an Agile Champion**: a senior-level manager or executive who can help ensure organizational change; the fundamental process changes that accompany agile transitions require support from the people who make and enforce business decisions and a good agile champion is able to rally the organization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

Projects that can benefit from going Agile



Source: (T2-Chap17)

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



Source: (T2-Chap17)

Managing Change (12 Steps)

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

Agile Success Metrics

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

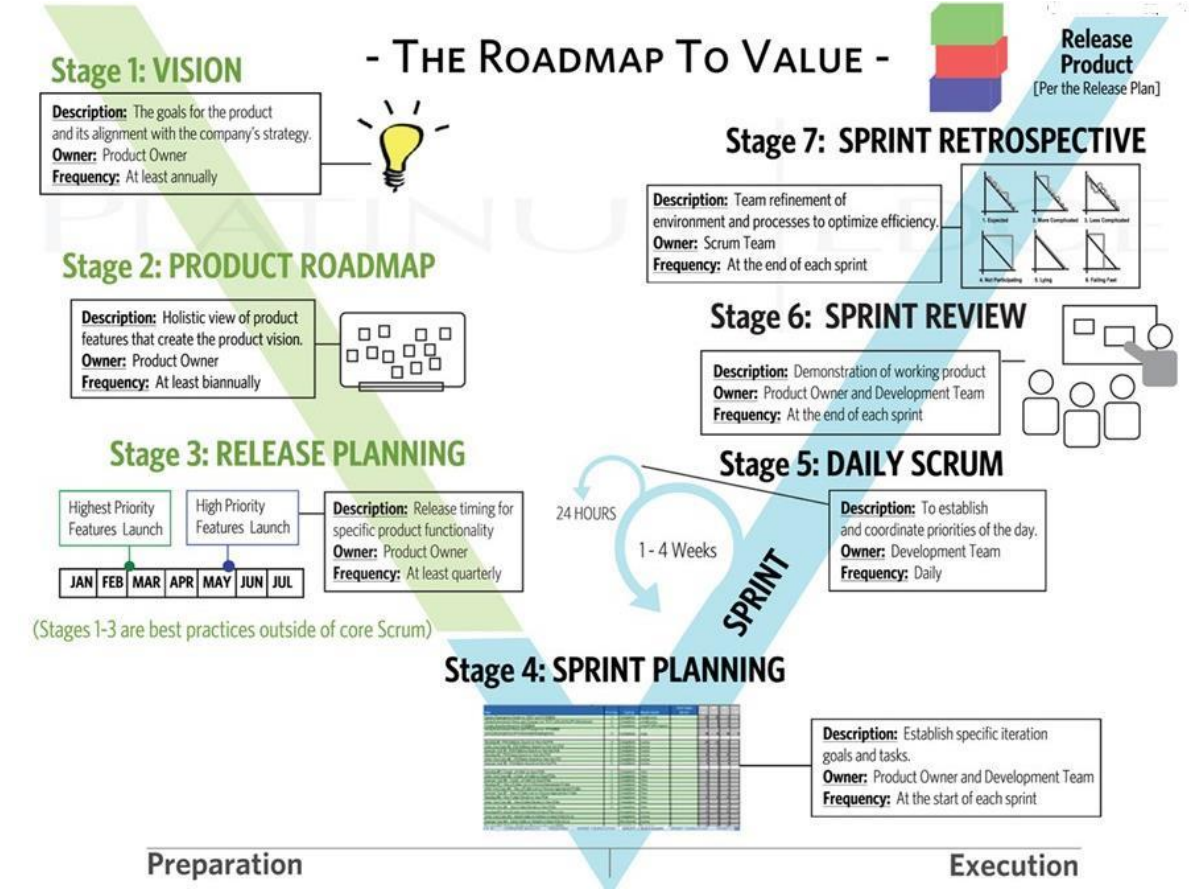
Agile is an Attitude, Not a Technique…

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

- *Alistair Cockburn* (Agile *Guru*)

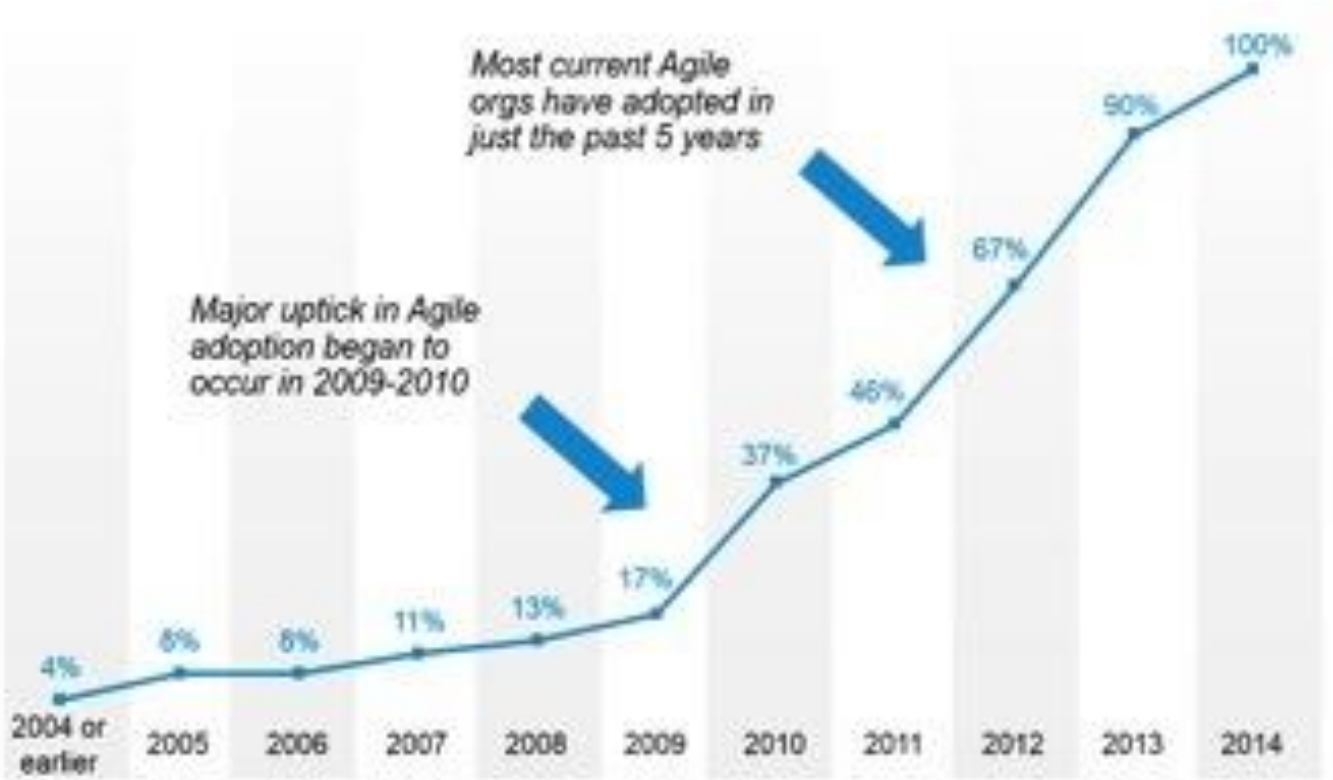
Evolution of Agile with the Times ➔➔

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



Source: (T2)

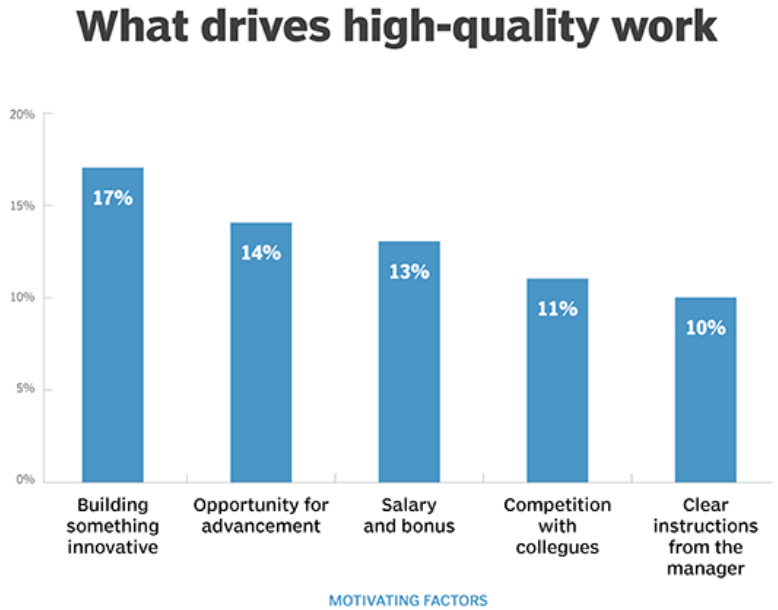
Agile Adoption in Recent Years…



Source: Tech Beacon

Agile-driven Transformations

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



Source: 2017 Survey of 5000 Development Professionals by CAST

Let's "Google-walk" the Agile

agile methods

All Images News Videos Maps More Settings Tools

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

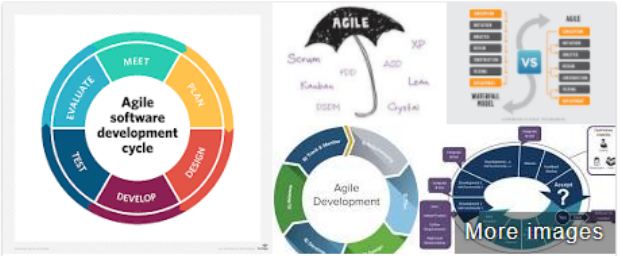
"**Agile** Development" is an umbrella term for several iterative and incremental software development **methodologies**. The most popular **agile methodologies** include Extreme Programming (XP), Scrum, Crystal, Dynamic Systems Development Method (DSDM), Lean Development, and Feature-Driven Development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

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

About this result Feedback

People also ask

- What is meant by agile model?
- What are the different agile methods?
- Which is the best Agile methodology?



Agile software development

Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams and their customer/end users. Wikipedia

People also search for View 15+ more

Summary: Ensuring Agile Success

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

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

Thank You