Build COMPETENCY
across your TEAM

SYNERGETICS
GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Lesson 1. Agile Development

Google Cloud Platform

Windows Azure

amazon web services

APACHE kafka
A distributed streaming platform

Apache Software Foundation

APACHE Spark

Java

hadoop

Topics Overview

- Overview of Agile
- Iterative development
- Prioritization and Planning
- Continuous Integration
- Continuous Delivery

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Module 1

Overview of Agile

What is Agile?

- The ability to create and respond to change in order to succeed in an uncertain and turbulent environment.

What is Agile Software Development?

- Agile Software Development is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto.

- Solutions evolve through collaboration between self-organizing, cross-functional teams utilizing the appropriate practices for their context.

What is Agile Methodology?

- AGILE methodology is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing activities are concurrent unlike the Waterfall model

A Short History of Agile

- In the late 1990's, several methodologies began to gain increasing public attention, each having a different combination of old and new ideas.

- These methodologies emphasized close collaboration between the development team and business stakeholders; frequent delivery of business value, tight, self-organizing teams; and smart ways to craft, confirm, and deliver code.

- The term "Agile" was applied to this collection of methodologies in early 2001 when 17 software development practitioners gathered in Snowbird, Utah to discuss their shared ideas and various approaches to software development.

A Short History of Agile

- This joint collection of values and principles was expressed in the Manifesto for Agile Software Development and the corresponding twelve principles.
- Agile Alliance was formed shortly after this gathering to encourage practitioners to further explore and share ideas and experiences.
- Agile Alliance continues to curate resources to help you adopt Agile practices and improve your ability to develop software with agility.

Key Agile Concepts

- User Stories
- Daily Meeting
- Incremental Development
- Iterative Development
- Team
- Milestone Retrospective
- Personas

The Agile Manifesto

Individuals and interactions	Over	Process and tools
Working Software	Over	Comprehensive Documentation
Customer Collaboration	Over	Contract Negotiation
Responding to change	Over	Following a plan

The 12 Agile 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 shorted timescale.
- **Business people and developers** must work together daily throughout the project.

The 12 Agile Principles

- 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.

The 12 Agile Principles

- 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 behaviour accordingly.

People Oriented

- Agile methods are people oriented rather then process oriented.

- Unleash creativity and innovation by recognizing

Agile Project Management Forces

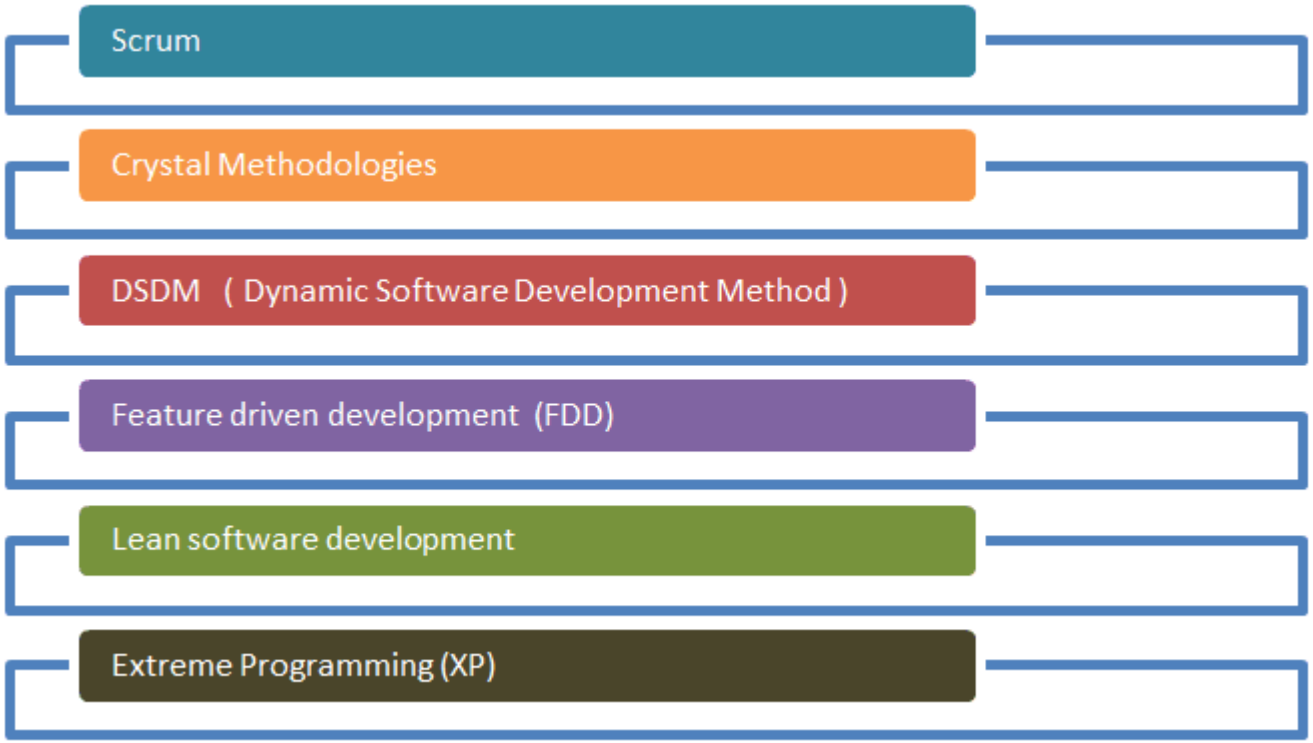
Courtesy of Sanjiv Augustine - http://www.ccpace.com

The Agile Practices Family

- Extreme Programming (XP)
- Crystal
- Dynamic System Development Method (DSDM)
- Test Driven Development (TDD)
- Features Driven Development (FDD)
- Essential Unified Process
- Scrum
- Kanban
- Lean

Agile Testing Methodology

- There are various **methods** present in agile testing

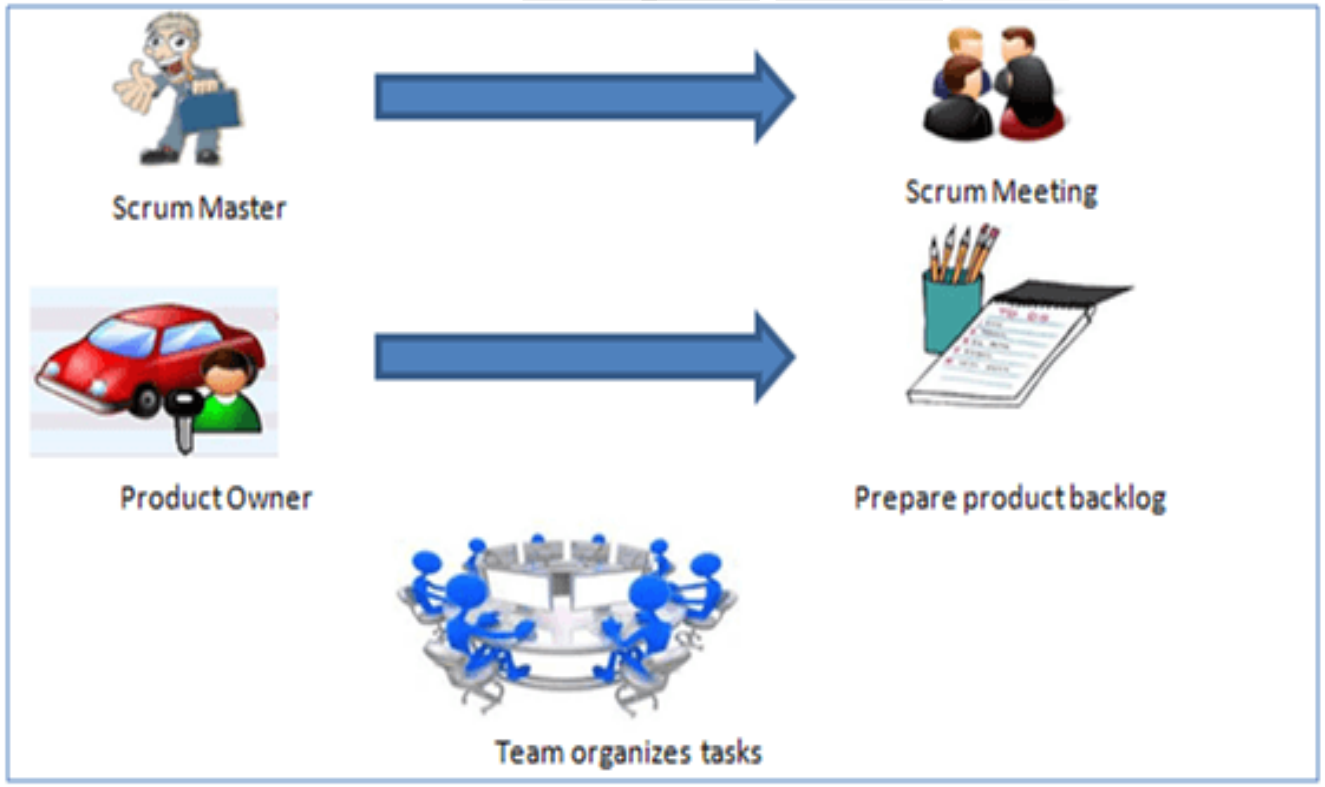


Scrum

- SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment.
- Basically, Scrum is derived from activity that occurs during a rugby match.
- Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members).
- It consists of three roles, Scrum Master, Product Owner, Scrum Team

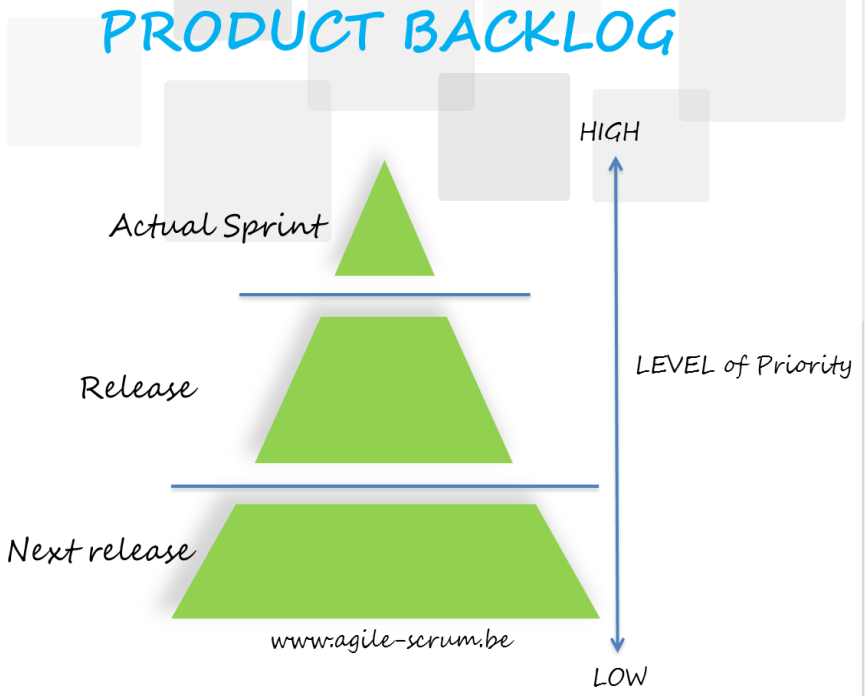
Scrum

- Scrum Master
  - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- Product owner
  - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- Scrum Team
  - Team manages its own work and organizes the work to complete the sprint or cycle

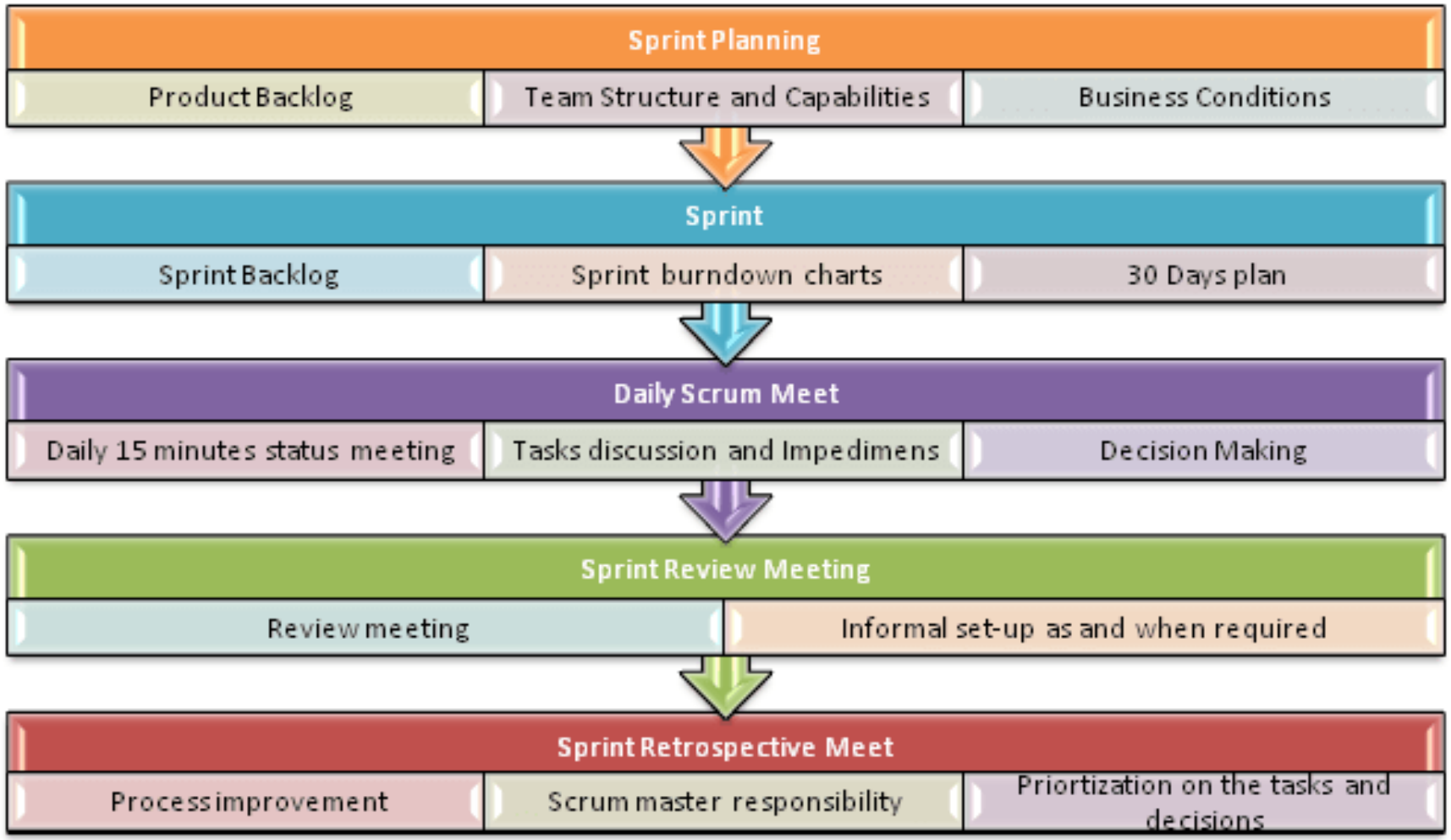


Scrum Product Backlog

- This is a repository where requirements are tracked with details on the no of requirements to be completed for each release.
- It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team.
- Team can also request for a new requirement addition or modification or deletion



Scrum Practices



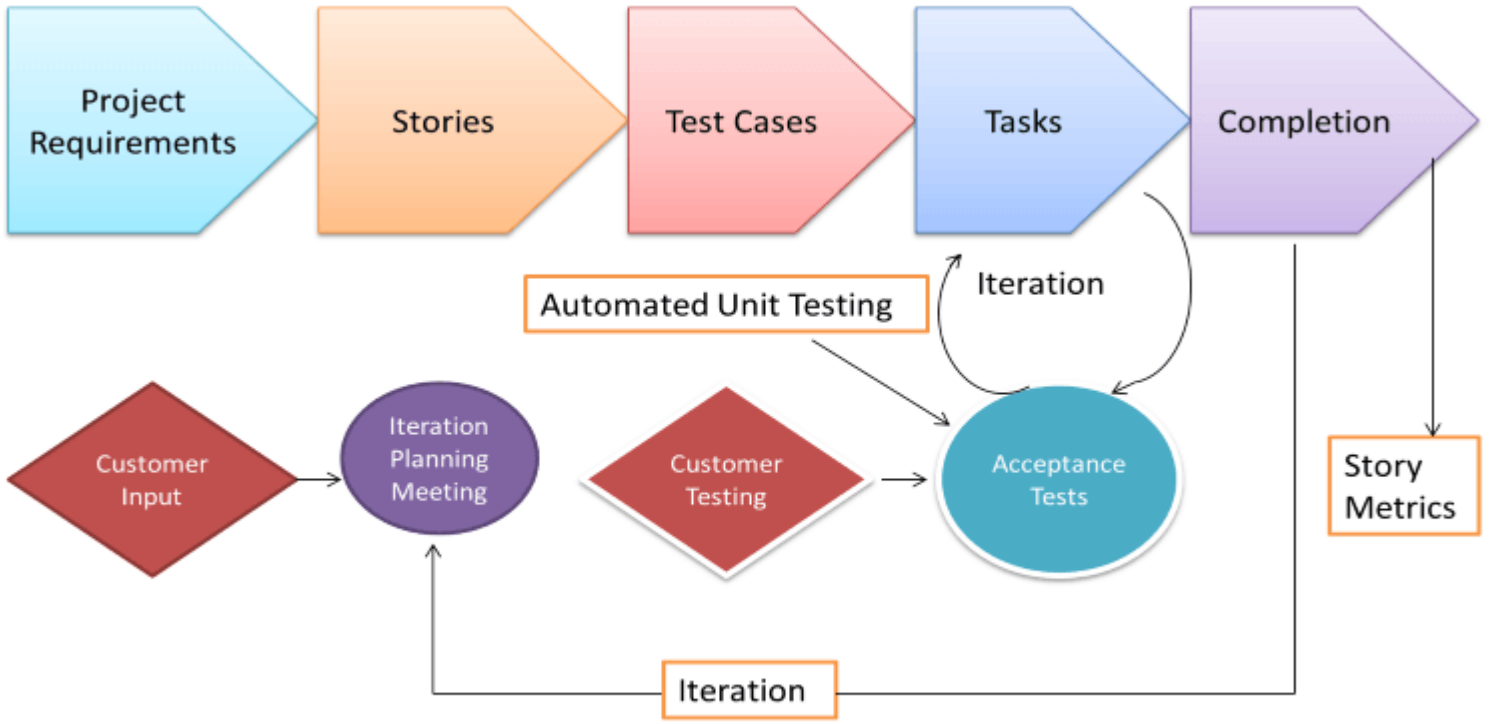
Process flow of Scrum Methodologies:

- Each iteration of a scrum is known as Sprint
- Product backlog is a list where all details are entered to get end product
- During each Sprint, top items of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

eXtreme Programming (XP)

- Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.
- It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented.
- The XP develops software keeping customer in the target.

eXtreme Programming (XP)



Extreme Programming (XP)

eXtreme Programming (XP)

- Business requirements are gathered in terms of stories. All those stories are stored in a place called the parking lot.
- In this type of methodology, releases are based on the shorter cycles called Iterations with span of 14 days time period.
- Each iteration includes phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.

Phases of eXtreme Programming (XP)

There are 6 phases available in Agile XP method, and those are explained as follows:

- Planning
- Analysis
- Design
- Execution
- Wrapping
- Closure

Phases of eXtreme Programming (XP)

- ***Planning***
  - Identification of stakeholders and sponsors
  - Infrastructure Requirements
  - Security related information and gathering
  - Service Level Agreements and its conditions

- ***Analysis***
  - Capturing of Stories in Parking lot
  - Prioritize stories in Parking lot
  - Scrubbing of stories for estimation
  - Define Iteration SPAN(Time)
  - Resource planning for both Development and QA teams

Phases of eXtreme Programming (XP)

- ***Design***
  - Break down of tasks
  - Test Scenario preparation for each task
  - Regression Automation Framework

- ***Execution***
  - Coding
  - Unit Testing
  - Execution of Manual test scenarios
  - Defect Report generation
  - Conversion of Manual to Automation regression test cases
  - Mid Iteration review
  - End of Iteration review

Phases of eXtreme Programming (XP)

- ***Wrapping***
  - Small Releases
  - Regression Testing
  - Demos and reviews
  - Develop new stories based on the need
  - Process Improvements based on end of iteration review comments

- ***Closure***
  - Pilot Launch
  - Training
  - Production Launch
  - SLA Guarantee assurance
  - Review SOA strategy
  - Production Support

eXtreme Programming (XP)

- There are two storyboards available to track the work on a daily basis, and those are listed below for reference.

- Story Cardboard
  - This is a traditional way of collecting all the stories in a board in the form of stick notes to track daily XP activities. As this manual activity involves more effort and time, it is better to switch to an online form.

- Online Storyboard
  - Online tool Storyboard can be used to store the stories. Several teams can use it for different purposes.

Crystal Methodologies

Crystal Methodology is based on three concepts

- Chartering
- Cyclic delivery
- Wrap Up

Crystal Methodologies

- **Chartering:** Various activities involved in this phase are creating a development team, performing a preliminary feasibility analysis, developing an initial plan and fine-tuning the development methodology

- **Cyclic delivery:** The main development phase consists of two or more delivery cycles, during which the
  - Team updates and refines the release plan
  - Implements a subset of the requirements through one or more program test integrate iterations
  - Integrated product is delivered to real users
  - Review of the project plan and adopted development methodology

- **Wrap Up:** The activities performed in this phase are deployment into the user environment, post-deployment reviews and reflections are performed.

Dynamic Software Development Method (DSDM)

- **DSDM** is a **Rapid Application Development (RAD)** approach to software development and provides an agile project delivery framework.
- The important aspect of **DSDM** is that the users are required to be involved actively, and the teams are given the power to make decisions.
- Frequent delivery of product becomes the active focus with DSDM.

Techniques used in DSDM

- Time Boxing
- MoSCoW Rules
- Prototyping

7 phases of DSDM project

- Pre-project
- Feasibility Study
- Business Study
- Functional Model Iteration
- Design and build Iteration
- Implementation
- Post-project

Feature Driven Development (FDD)

- This method is focused around "**designing & building**" features.
- Unlike other agile methods, **FDD** describes very specific and short phases of work that has to be accomplished separately per feature.
- It includes **domain walkthrough, design inspection, promote to build, code inspection and design.**

Feature Driven Development (FDD)

FDD develops product keeping following things in the target

- Domain object Modelling
- Development by feature
- Component/ Class Ownership
- Feature Teams
- Inspections
- Configuration Management
- Regular Builds
- Visibility of progress and results

Lean Software Development

- Lean software development method is based on the principle "Just in time production". It aims at increasing speed of software development and decreasing cost.

Lean development can be summarized in seven steps.

- Eliminating Waste
- Amplifying learning
- Defer commitment (deciding as late as possible)
- Early delivery
- Empowering the team
- Building Integrity
- Optimize the whole

Kanban

- Kanban originally emerged from Japanese word that means, a card containing all the information needed to be done on the product at each stage along its path to completion.
- This framework or method is quite adopted in software testing method especially in agile testing.

Scrum Vs Kanban

Scrum	Kanban
In scrum technique, test must be broken down so that they can be completed within one sprint	No particular item size is prescribed
Prescribes a prioritized product backlog	Prioritization is optional
Scrum team commits to a particular amount of work for the iteration	Commitment is optional
Burndown chart is prescribed	No particular item size is prescribed
Between each sprint, a scrum board is reset	A Kanban board is persistent. It limits the number of items in workflow state
It cannot add items to ongoing iteration	It can add items whenever capacity is available
WIP limited indirectly	WIP limited directly
Timeboxed iterations prescribed	Timeboxed iterations optional

Agile metrics

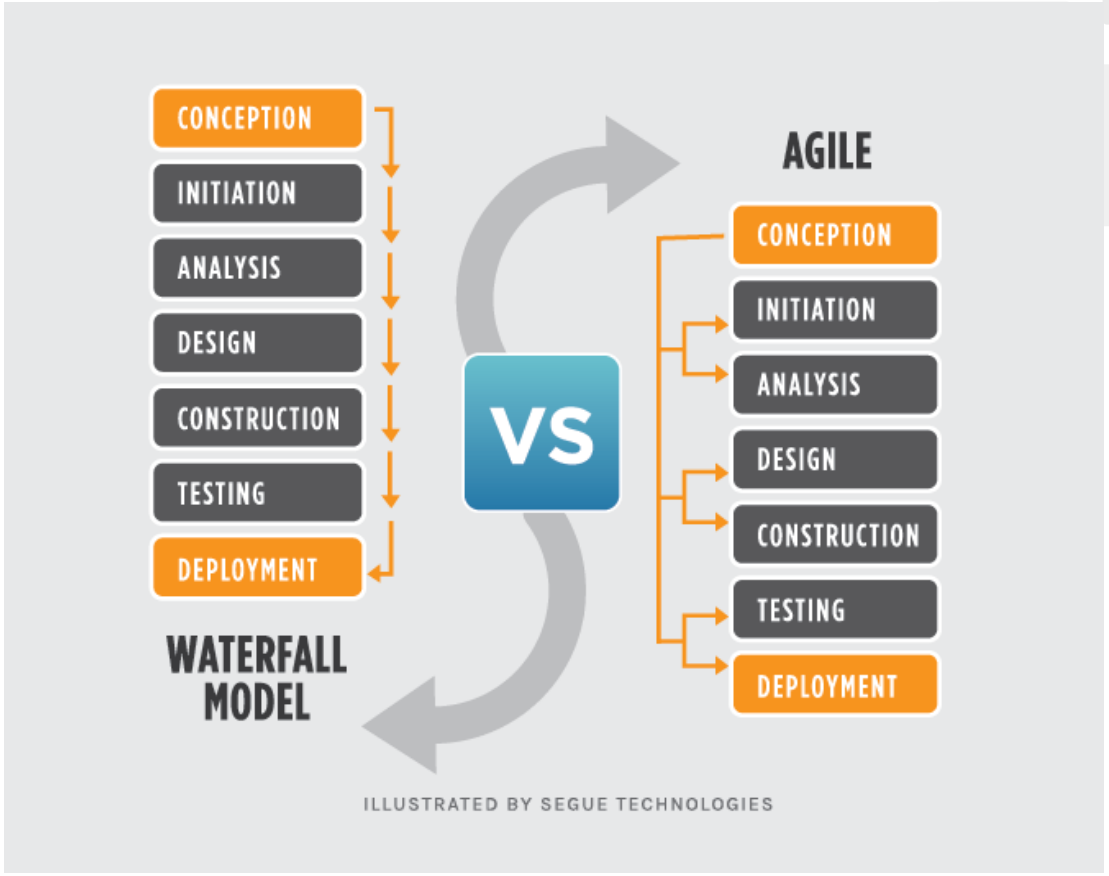
- Metrics that can be collected for effective usage of Agile is:
- Drag Factor
  - Effort in hours which do not contribute to sprint goal
  - Drag factor can be improved by reducing number of shared resources, reducing the amount of non-contributing work
  - New estimates can be increased by percentage of drag factor -New estimate = (Old estimate+drag factor)
- Velocity
  - Amount of backlog converted to shippable functionality of sprint
- No of Unit Tests added
- Time taken to complete daily build
- Bugs detected in an iteration or in previous iterations
- Production defect leakage

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Module 2

Iterative development

Waterfall model vs Agile Model



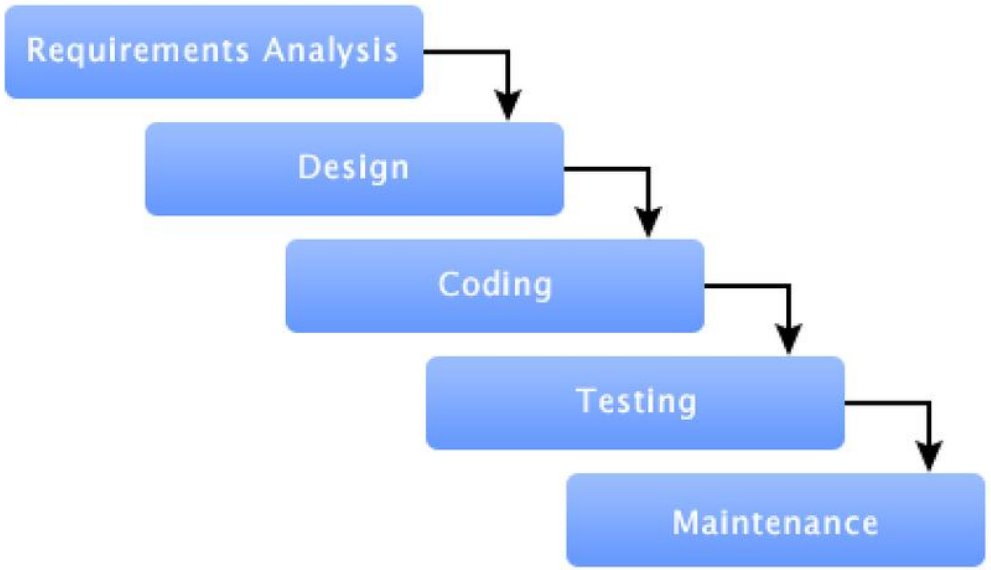
The two basic, most popular methodologies are:

- **Waterfall**: which might be more properly called the “traditional” approach, and
- **Agile**: a specific type of Rapid Application Development and newer than Waterfall, but not that new, which is often implemented using Scrum.

The Waterfall Methodology

Waterfall is a linear approach to software development. In this methodology, the sequence of events is something like:

- Gather and document requirements
- Design
- Code and unit test
- Perform system testing
- Perform user acceptance testing (UAT)
- Fix any issues
- Deliver the finished product



The Waterfall Methodology

- In a true Waterfall development project, each of these represents a distinct stage of software development, and each stage generally finishes before the next one can begin.
- There is also typically a stage gate between each; for example, requirements must be reviewed and approved by the customer before design can begin.

The positive side

- Developers and customers agree on what will be delivered early in the development lifecycle. This makes planning and designing more straightforward.
- Progress is more easily measured, as the full scope of the work is known in advance.
- Throughout the development effort, it's possible for various members of the team to be involved or to continue with other work, depending on the active phase of the project.

For example, business analysts can learn about and document what needs to be done, while the developers are working on other projects. Testers can prepare test scripts from requirements documentation while coding is underway.

The positive side

Except for reviews, approvals, status meetings, etc., a customer presence is not strictly required after the requirements phase.

Because design is completed early in the development lifecycle, this approach lends itself to projects where multiple software components must be designed (sometimes in parallel) for integration with external systems.

- Finally, the software can be designed completely and more carefully, based upon a more complete understanding of all software deliverables. This provides a better software design with less likelihood of the “piecemeal effect,” a development phenomenon that can occur as pieces of code are defined and subsequently added to an application where they may or may not fit well.

The Negative Side

- One area which almost always falls short is the effectiveness of requirements. Gathering and documenting requirements in a way that is meaningful to a customer is often the most difficult part of software development. Customers are sometimes intimidated by details, and specific details, provided early in the project, are required with this approach. In addition, customers are not always able to visualize an application from a requirements document.

- Another potential drawback of pure Waterfall development is the possibility that the customer will be dissatisfied with their delivered software product. As all deliverables are based upon documented requirements, a customer may not see what will be delivered until it's almost finished. By that time, changes can be difficult (and costly) to implement.

Waterfall model vs Agile Model

Agile Model	Waterfall Model
Agile method proposes incremental and iterative approach to software design	Development of the software flows sequentially from start point to end point.
The **agile process** is broken into individual models that designers work on	The design process is not broken into an individual models
The customer has early and frequent opportunities to look at the product and make decision and changes to the project	The customer can only see the product at the end of the project
Agile model is considered unstructured compared to the waterfall model	Waterfall model are more secure because they are so plan oriented
Small projects can be implemented very quickly. For large projects, it is difficult to estimate the development time.	All sorts of project can be estimated and completed.
Error can be fixed in the middle of the project.	Only at the end, the whole product is tested. If the requirement error is found or any changes have to be made, the project has to start from the beginning

Waterfall model vs Agile Model

Agile Model	Waterfall Model
Development process is iterative, and the project is executed in short (2-4) weeks iterations. Planning is very less.	The development process is phased, and the phase is much bigger than iteration. Every phase ends with the detailed description of the next phase.
Documentation attends less priority than software development	Documentation is a top priority and can even use for training staff and upgrade the software with another team
Every iteration has its own testing phase. It allows implementing regression testing every time new functions or logic are released.	Only after the development phase, the testing phase is executed because separate parts are not fully functional.

Waterfall model vs Agile Model

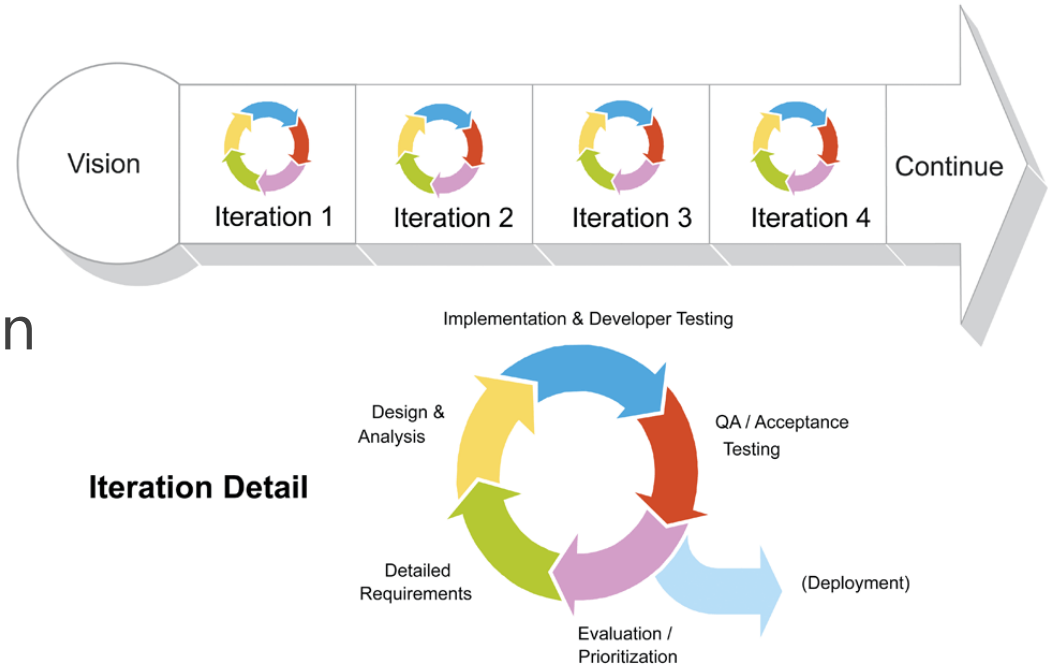
Agile Model	Waterfall Model
Testers and developers work together	Testers work separately from developers
At the end of every sprint, user acceptance is performed	User acceptance is **performed** at the end of the project.
It requires close communication with developers and together analyse requirements and planning	Developer does not involve in requirement and planning process. Usually, time delays between tests and coding
In agile testing when an iteration end, shippable features of the product is delivered to the customer. New features are usable right after shipment. It is useful when you have good contact with customers.	All features developed are delivered at once after the long implementation p

Iterative Development

- Iterative development is a way of breaking down the software development of a large application into smaller chunks.
- In iterative development, feature code is designed, developed and tested in repeated cycles.
- With each iteration, additional features can be designed, developed and tested until there is a fully functional software application ready to be deployed to customers.
- Typically iterative development is used in conjunction with incremental development in which a longer software development cycle is split into smaller segments that build upon each other.

Iterative Development in Agile

- Iterative and incremental development are key practices in Agile development methodologies.
- In Agile methodologies, the shorter development cycle, referred to as an iteration or sprint, is time-boxed (limited to a certain increment of time, such as two weeks).
- At the end of the iteration, working code is expected that can be demonstrated for a customer.



Iterative Development in Agile

- Iterative development contrasts with a traditional waterfall method in which each phase of the software development life cycle is “gated.”
- The purpose of working iteratively is to allow more flexibility for changes.
- When requirements and design of a major application are done in the traditional method (sometimes referred to as *BDUF* or *Big Design Up Front*), there can be unforeseen problems that don’t surface until development begins.
- By working iteratively, the project team goes through a cycle where they evaluate with each iteration, and determine what changes are needed to produce a satisfactory end product.


SYNERGETICS
GET IT RIGHT
Microsoft Partner
Gold Cloud Platform
Silver Learning

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Module 3

Prioritization and Planning

Agile Prioritization Techniques

- Prioritization in literary terms means the decision of arranging things in order of their importance.
- Prioritization in agile is the act of deciding in what order the agile team will work on the requirements in a project.
- Understanding prioritization is essential for all projects but it becomes specifically critical in agile as an agile project is time-boxed with fixed set of resources which requires prioritization in order to accommodate the time and budget constraints.
- Further prioritization process helps the agile team to consider the bare minimum features necessary to create customer value. In order to process agile prioritization, it is essential to understand the factors that a product owner needs to consider before determining the priorities.

Agile Prioritization Factors

- The financial value of the requirements is a major factor to be considered in prioritizing requirements. The value could be expressed as new revenue, incremental revenue or as operational efficiency.
- The cost of developing the requirements is another essential factor to be considered by the product owner. Value and cost together indicate the RoI for the requirements.
- The next factor to be considered in prioritization is the amount and significance of knowledge and capabilities that the team will gain while working on the requirements.
- Understanding the level of risks involved in introducing the new features is very essential in the process of prioritization.

Popular Prioritization Techniques

- MoSCoW Agile Prioritization Technique
- MoSCoW prioritization – popularized by the DSDM methodology.
- Kano model – introduced by Prof. Noriaki Kano
- The relative weighting method – by Karl Wiegers

MoSCoW Prioritization in Agile

- In the DSDM methodology the priorities are expressed as per the MoSCoW model:

- **Must** – The must requirements is given the top most priority
- **Should** – Next priority is given to the requirements that are high desirable, though not mandatory
- **Could** – The next priority is given to the requirement that are nice to have
- **Wont** – And the final consideration is given to the requirements which will not work in the process at that point of time.

Kano Model of Prioritization in Agile

- Kano Model of Prioritization[/caption] Kano model of prioritization was propagated by Professor Noriaki Kano.
- This prioritization technique involves three levels that includes considering customer satisfaction from disappointment to not happy to immediate happiness to getting delighted.
- Two important factors that create an impact on the satisfaction level during this prioritization are the existence of features and the degree of implementation.
- The level of satisfaction is achieved along with full implementation.
- Some features lead to basic level of satisfaction while other creates more – the higher the implementation, the greater the level of satisfaction.

Relative Weighting Prioritization Technique

- Relative weighing scheme is a simple model where prioritization is done based upon all the factors mentioned covered earlier.

- The major factors considered in relative weighing prioritization technique are
- The value of a feature and the negative impact that might be caused by the absence of the feature

- Based on expert judgment made by the product owner and supported by the agile team in ranking the score of features in following way (a score board from 1 to 9 is usually used)
  - Benefit from having the feature
  - Penalty for not having the feature
  - Cost of producing the feature
  - Risk incurred in producing the feature

- The priority and rank is then determined by dividing the value score as below:

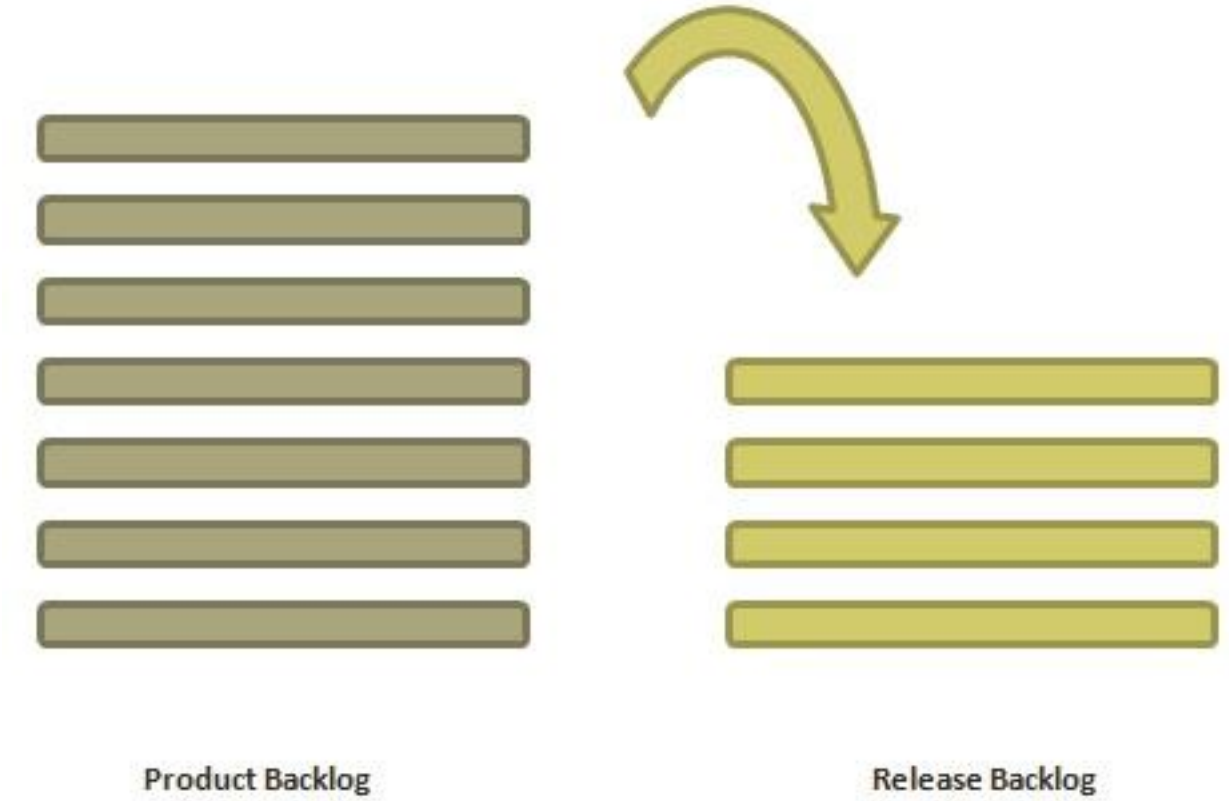
(Benefit score + Penalty score) / (Cost score + Risk score)

Relative Weighting Prioritization Technique

- In relative weighting prioritization if the results come out in numerical value it becomes easier for the product owner to arrive at a faster prioritizing decision.
- Using all these three techniques, a product owner performs the prioritization exercise towards achieving customer satisfaction and customer value.
- The whole process of prioritization in agile is followed in order to create customer value which is possible with innovation, focused execution and lean delivery.

Agile - Release Planning

- The purpose of release planning is to create a plan to deliver an increment to the product.
- It is done after every 2 to 3 months.



Agile - Release Planning

- Who is Involved?
  - **Scrum Master** – The scrum master acts as a facilitator for the agile delivery team.
  - **Product Owner** – The product owner represents the general view of the product backlog.
  - **Agile Team** – Agile delivery team provides insights on the technical feasibilities or any dependencies.
  - **Stakeholders** – Stakeholders like customers, program managers, subject matter experts act as advisers as decisions are made around the release planning.

Prerequisites of Planning

- The prerequisites of release planning are as follows –
  - A ranked product backlog, managed by the Product Owner. Generally five to ten features are taken which the product owner feels that can be included in a release
  - Team's input about capabilities, known velocity or about any technical challenge
  - High-level vision
  - Market and Business objective
  - Acknowledgement whether new product backlog items are needed

Materials Required

- The list of materials required for release planning is as follows –
  - Posted agenda, purpose
  - Flip charts, whiteboards, markers
  - Projector, way to share computers having data/tools required during planning meeting
  - Planning data

Planning Data

- The list of data required to do release planning is as follows –
  - Previous iterations or release planning results
  - Feedback from various stakeholders on product, market conditions, and deadlines
  - Action plans of previous releases / iterations
  - Features or defects to be considered
  - Velocity from previous releases/ estimates.
  - Organizational and personal calendars
  - Inputs from other teams and subject matter experts to manage any dependencies

Output

- The output of a release planning can be the following –
  - Release plan
  - Commitment
  - Issues, concerns, dependencies, and assumptions which are to be monitored
  - Suggestions to improve future release plannings

Agenda

- **Opening ceremony** – Welcome message, review purpose and agenda, organizing tools and introduction to business sponsors.
- **Product Vision, Roadmap** – Show the large picture of the product.
- **Review previous releases** – Discussion on any item which can impact the plan.
- **Release name / theme** – Inspect the current status of roadmap themes and do the required adjustments, if any.
- **Velocity** – Present the velocity for the current release and of previous releases.
- **Release schedule** – Review key milestones and decision on time boxes for release and iterations within release.
- **Issues and concerns** – Check any concerns or issue and record them.

Agenda

- **Review and Update the Definition of Done** – Review the definition of **done** and make appropriate changes based on technology, skill, or changes in team members since the last iteration / release.
- **Stories and items to be considered** – Present the user stories and features from the product backlog to be considered for scheduling in the current release.
- **Determine sizing values** – If the velocity is unknown, then plan the sizing values to be used in the release planning.
- **Coarse the size of stories** – The delivery team determines the appropriate size of the stories under consideration and splits the stories into multiple iterations if a story is too large. The product owner and the subject matter experts clarify the doubts, elaborate the acceptance criteria, and make proper story splits. The scrum master facilitates the collaboration.

Agenda

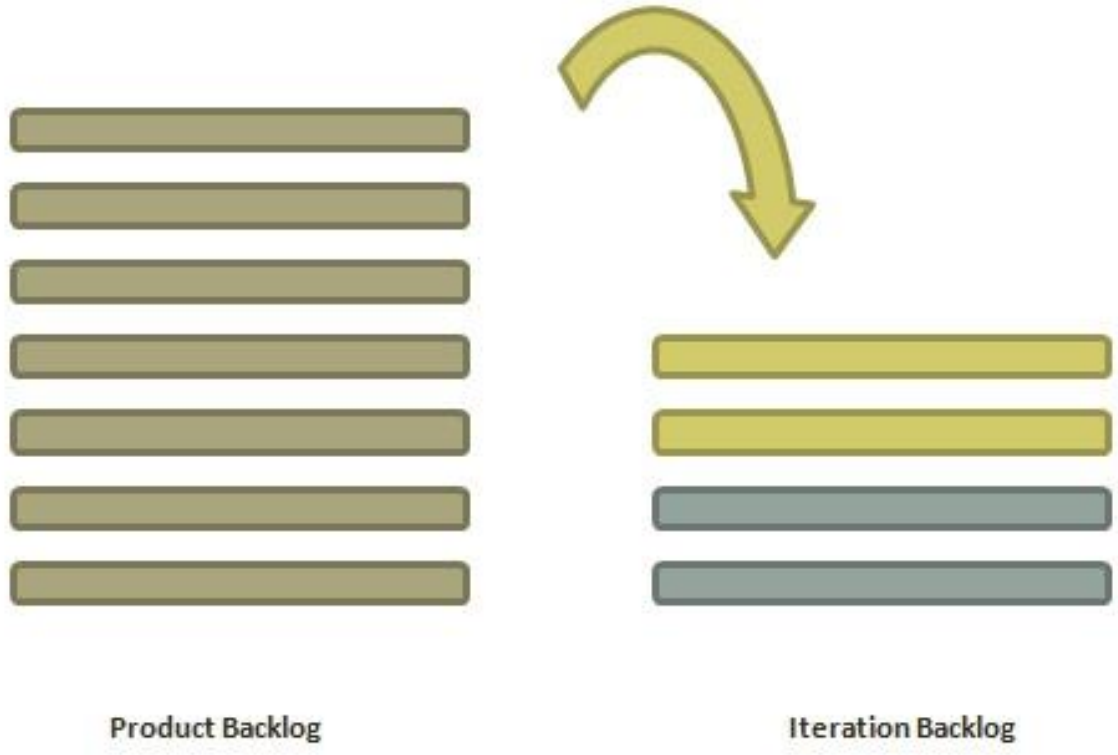
- **Map stories to iterations** – The delivery team and the product owner move the stories/defects in the iterations based on the size and velocity. The scrum master facilitates the collaboration.
- **New concerns or issues** – Check any new issues based on previous experience and record the same.
- **Dependencies and assumptions** – Check any dependencies/assumptions planned during the release planning.
- **Commit** – The scrum master calls for the planning. Delivery team and Product owner signal it as the best plan and then commit to move to the next level of planning, that is, iteration planning.

Agenda

- **Communication and logistics planning** – Review/Update the communication and logistics planning for the release.
- **Parking lot** – Process parking lot means all items should be either resolved or set as action items.
- **Distribute Action items and action plans** – Distribute the action items among their owners, process the action plan.
- **Retrospect** – Solicit feedback from participants to make the meeting successful.
- **Close** – Celebrate the success.

Agile - Iteration Planning

- The purpose of iteration planning is for the team to complete the set of top-ranked product backlog items.
- This commitment is time boxed based on the length of iteration and team velocity.



Agile - Iteration Planning

- Who is Involved?
  - **Scrum Master** – The scrum master acts as a facilitator for the agile delivery team.
  - **Product Owner** – The product owner deals with the detailed view of the product backlog and their acceptance criteria.
  - **Agile Team** – Agile delivery defines their tasks and sets the effort estimates required to fulfil the commitment.

Prerequisites of Planning

- Items in product backlog are sized and have a relative story point assigned.
- Ranking has been given to portfolio items by the product owner.
- Acceptance criteria has been clearly stated for each portfolio item.

Planning Process

- Determine how many stories can fit in an iteration.
- Break these stories into tasks and assign each task to their owners.
- Each task is given estimates in hours.
- These estimates help team members to check how many task hours each member have for the iteration.
- Team members are assigned tasks considering their velocity or capacity so that they are not overburdened.

Velocity Calculation

- An agile team calculates velocity based on past iterations. Velocity is an average number of units required to finish user stories in an iteration. For example, if a team took 12, 14, 10 story points in each iteration for the last three iterations, the team can take 12 as velocity for the next iteration.

- Planned velocity tells the team how many user stories can be completed in the current iteration. If the team quickly finishes the tasks assigned, then more user stories can be pulled in. Otherwise, stories can be moved out too to the next iteration.

Task Capacity

- The capacity of a team is derived from the following three facts –
  - Number of ideal working hours in a day
  - Available days of person in the iteration
  - Percentage of time a member is exclusively available for the team.
- Suppose a team has 5 members, committed to work full time (8 hours a day) on a project and no one is on leave during an iteration, then the task capacity for a two-week iteration will be –

$5 \times 8 \times 10 = 400$ hours

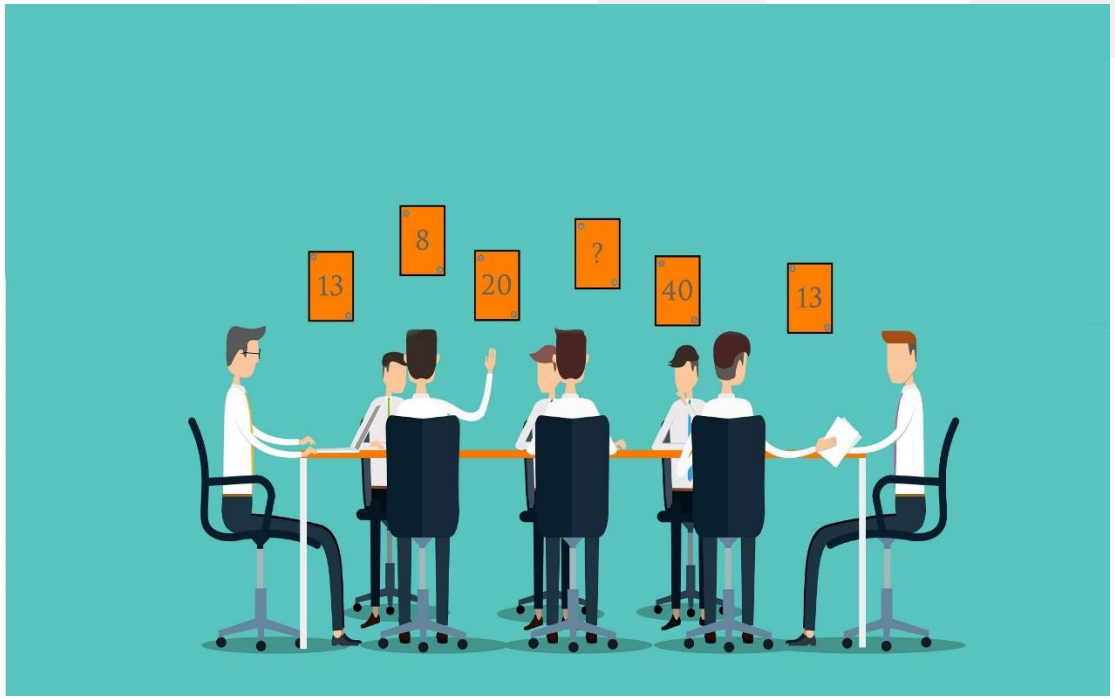
Planning Steps

- Product Owner describes the highest ranked item of product backlog.
- Team describes the tasks required to complete the item.
- Team members own the tasks.
- Team members estimate the time to finish each task.
- These steps are repeated for all the items in the iteration.
- If any individual is overloaded with tasks, then his/her task is distributed among other team members.

Planning Poker

- Planning Poker is an agile estimating and planning technique that is consensus based. To start a poker planning session, the product owner or customer reads an agile user story or describes a feature to the estimators.

- Each estimator is holding a deck of Planning Poker cards with values like 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100, which is the sequence we recommend. The values represent the number of story points, ideal days, or other units in which the team estimates.



Planning Poker

- The estimators discuss the feature, asking questions of the product owner as needed. When the feature has been fully discussed, each estimator privately selects one card to represent his or her estimate. All cards are then revealed at the same time.

- If all estimators selected the same value, that becomes the estimate. If not, the estimators discuss their estimates. The high and low estimators should especially share their reasons. After further discussion, each estimator reselects an estimate card, and all cards are again revealed at the same time.

Planning Poker

- The poker planning process is repeated until consensus is achieved or until the estimators decide that agile estimating and planning of a particular item needs to be deferred until additional information can be acquired.

When should we engage in Planning Poker?

- Most teams will hold a Planning Poker session shortly after an initial product backlog is written. This session (which may be spread over multiple days) is used to create initial estimates useful in scoping or sizing the project.

- Because product backlog items (usually in the form of user stories) will continue to be added throughout the project, most teams will find it helpful to conduct subsequent agile estimating and planning sessions once per iteration. Usually this is done a few days before the end of the iteration and immediately following a daily stand-up, since the whole team is together at that time anyway.

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Module 4

Continuous Integration

Why Continuous Integration (CI)

- Traditional software development methods don't dictate how frequently or regularly you integrate all of the source on a project.
- Programmers can work separately for hours, days, or even weeks on the same source without realizing how many conflicts (and perhaps bugs) they are generating.
- Agile teams, because they are producing robust code each iteration, typically find that they are slowed down by the long diff-resolution and debugging sessions that often occur at the end of long integration cycles.
- The more programmers are sharing the code, the more problematic this is.
- For these reasons, agile teams often therefore choose to use Continuous Integration.

Continuous Integration (CI)

- Continuous Integration (CI) involves producing a clean build of the system several times per day, usually with a tool like CruiseControl, which uses Ant and various source-control systems.
- Agile teams typically configure CI to include automated compilation, unit test execution, and source control integration.
- Sometimes CI also includes automatically running automated acceptance tests such as those developed using FitNesse.
- In effect, CI means that the build is nearly always clean.

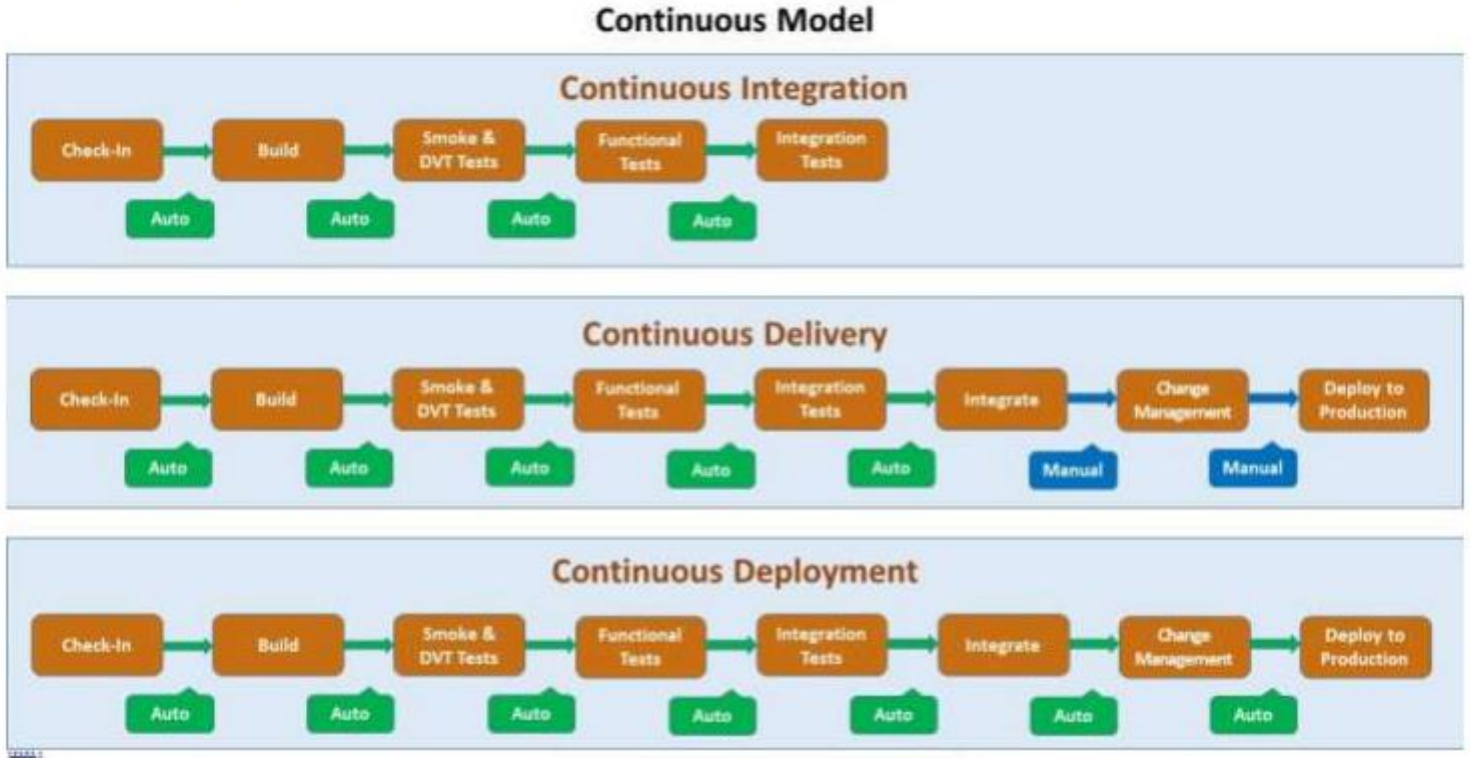
Continuous Integration (CI) by Martin Fowler

- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day.
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.
- Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

Continuous Model

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Continuous Integration Continuous deployment Continuous delivery



Continous Integration Tools

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

CI SERVERS:

- Jenkins
- Hudson
- Bamboo
- TeamCity
- cruisecontrol
- Travis

BUILD AUTOMATION TOOLS:

- Maven
- gradle
- MAKE
- RAKE

ARTIFACT REPOSITORIES:

- Nexus
- artifactory
- archiva

TEST FRAMEWORKS:

- JUnit
- TestNG
- Selenium

CODE ANALYSIS:

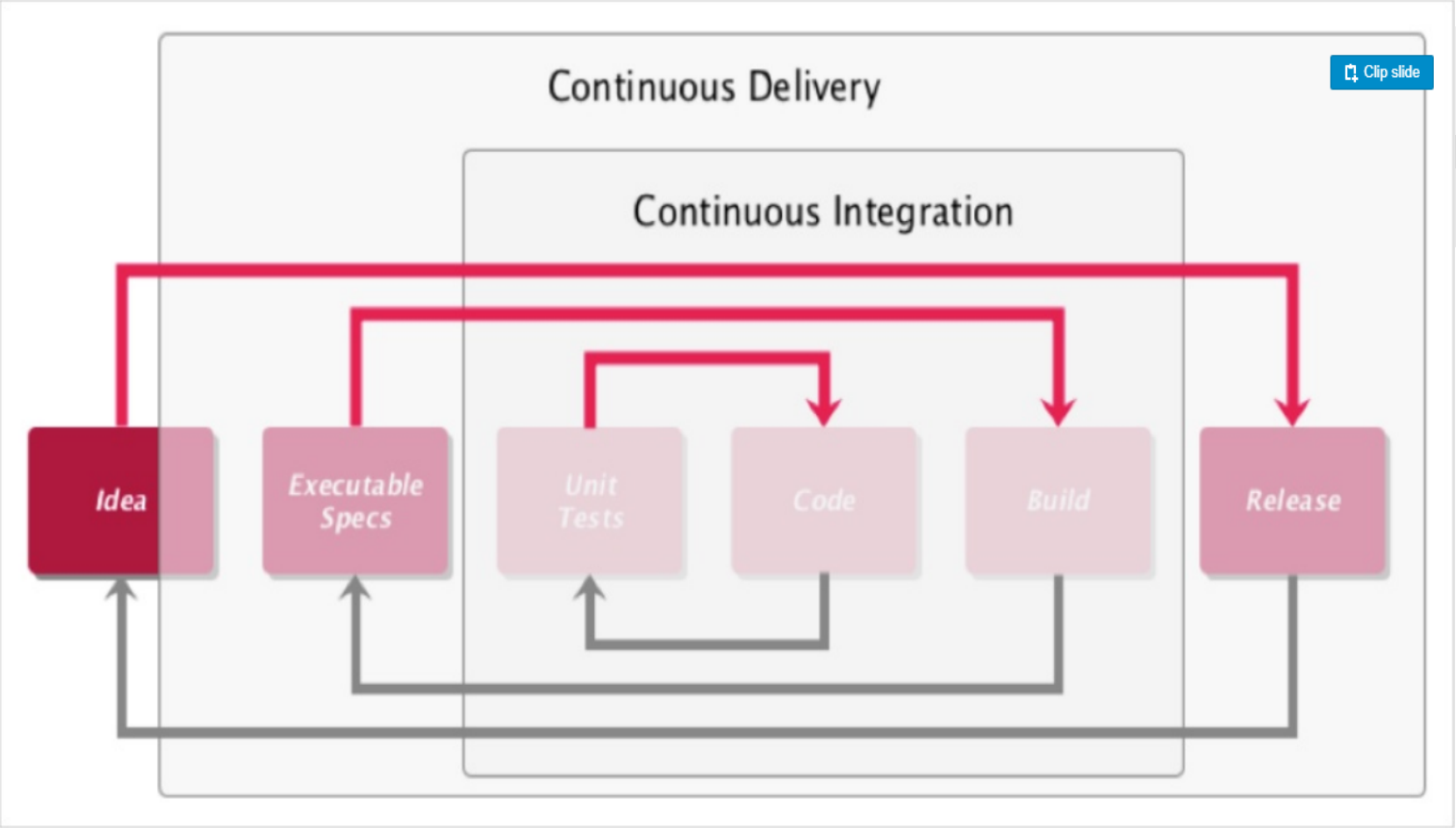
- sonar

Continuous Integration: The Practice

- Maintain a Single source repository
- Automate the build
- Make your build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what is happening
- Automate deployment

Continues integration: How to do it

- Developers check out code into their private workspaces
- When done, commit the changes to the repository
- The CI server monitors the repository and checks out the changes when they occur
- The CI server builds the system and runs unit and integration tests.
- The CI server releases deployable artefacts for testing.
- The CI server assigns a build label to the version of the code it just built.
- THE CI server informs the team of the successful build
- If the build or tests fail, the CI server alerts the team
- The team fix the issue at the earliest opportunity
- Continue to continually integrate and test throughtou the project



It is the practice of releasing every good build to users

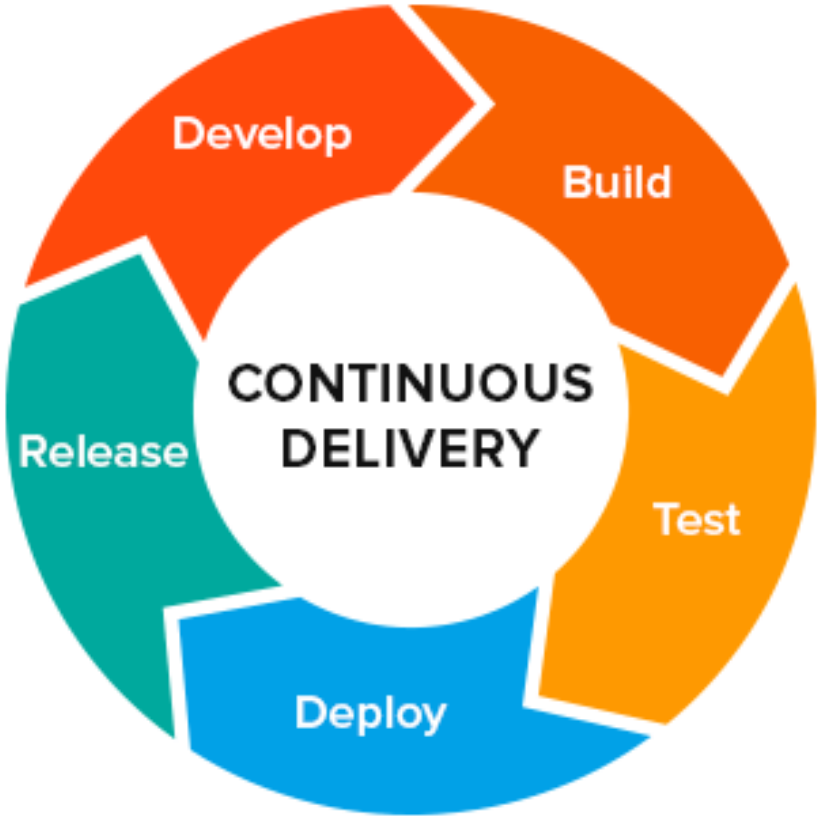

SYNERGETICS
GET IT RIGHT
Microsoft Partner
Gold Cloud Platform
Silver Learning

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

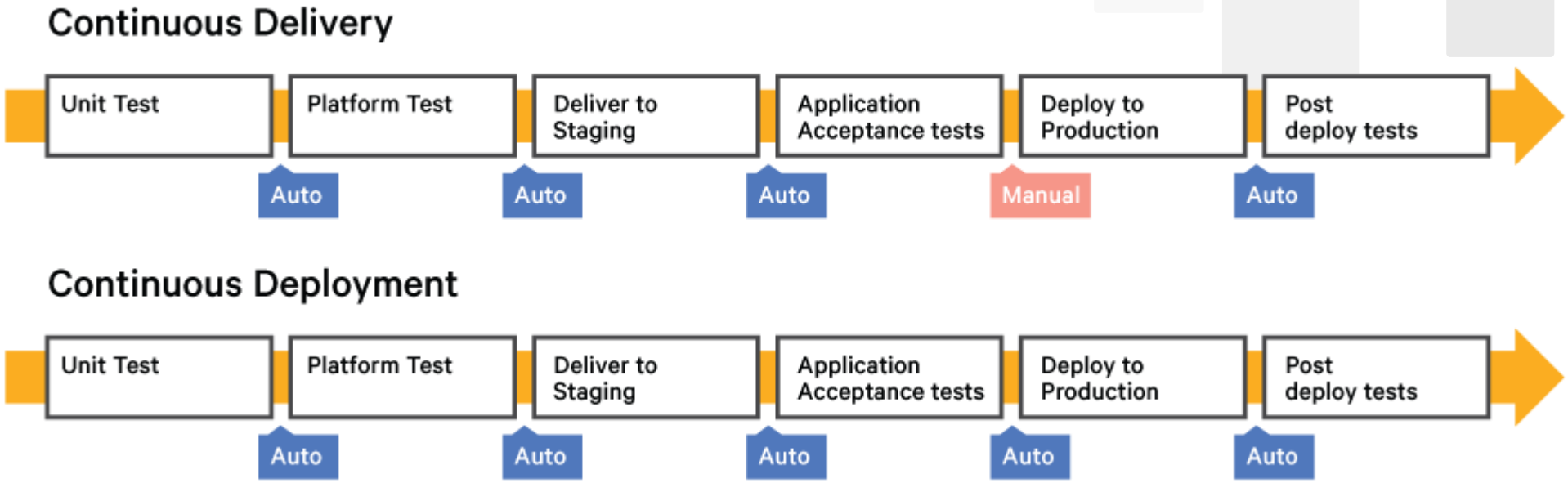
Module 5

Continuous Delivery

Continuous Delivery



Continuous Delivery VS Continuous Deployment



What is Continuous Delivery?

- Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

- Our goal is to make deployments—whether of a large-scale distributed system, a complex production environment, an embedded system, or an app—predictable, routine affairs that can be performed on demand.

- We achieve all this by ensuring our code is always in a deployable state, even in the face of teams of thousands of developers making changes on a daily basis. We thus completely eliminate the integration, testing and hardening phases that traditionally followed “dev complete”, as well as code freezes.

Why Continuous Delivery?

- It is often assumed that if we want to deploy software more frequently, we must accept lower levels of stability and reliability in our systems.

- In fact, peer-reviewed research shows that this is not the case—high performance teams consistently deliver services faster and more reliably than their low performing competition. (This is true even in highly regulated domains such as financial services and government.)

- This capability provides an incredible competitive advantage for organizations that are willing to invest the effort to pursue it.

Continuous Delivery

The practices at the heart of continuous delivery help us achieve several important benefits

- Low risk releases
- Faster time to market
- Higher quality
- Lower costs
- Better products
- Happier teams

Continuous Delivery

- **Low risk releases**. The primary goal of continuous delivery is to make software deployments painless, low-risk events that can be performed at any time, on demand. By applying patterns such as blue-green deployments it is relatively straightforward to achieve zero-downtime deployments that are undetectable to users.

- **Faster time to market**. It's not uncommon for the integration and test/fix phase of the traditional phased software delivery lifecycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and regression testing processes, developers can incorporate integration and regression testing into their daily work and completely remove these phases. We also avoid the large amounts of re-work that plague the phased approach.

Continuous Delivery

- **Higher qua**lity. When developers have automated tools that discover regressions within minutes, teams are freed to focus their effort on user research and higher level testing activities such as exploratory testing, usability testing, and performance and security testing. By building a deployment pipeline, these activities can be performed continuously throughout the delivery process, ensuring quality is built in to products and services from the beginning.

- **Lower costs**. Any successful software product or service will evolve significantly over the course of its lifetime. By investing in build, test, deployment and environment automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process.

Continuous Delivery

- **Better products**. Continuous delivery makes it economic to work in small batches. This means we can get feedback from users throughout the delivery lifecycle based on working software. Techniques such as A/B testing enable us to take a hypothesis-driven approach to product development whereby we can test ideas with users before building out whole features. This means we can avoid the 2/3 of features we build that deliver zero or negative value to our businesses.

- **Happier teams**. Peer-reviewed research has shown continuous delivery makes releases less painful and reduces team burnout. Furthermore, when we release more frequently, software delivery teams can engage more actively with users, learn which ideas work and which don't, and see first-hand the outcomes of the work they have done. By removing the low-value painful activities associated with software delivery, we can focus on what we care about most—continuously delighting our users.

Implementing Continuous Delivery

- Pick a Small, Manageable Project to Start
- Define a Process
- Ensure a Blameless Culture
- Set Metrics and Measure Your Success
- Adopt Configuration as Code
- Orchestrating a Process

Pick a Small, Manageable Project to Start

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

- A common mistake organizations make is trying to do too much too soon. CD enthusiasts believe in the methodology, and they tend to want to generate big gains quickly to justify the organization's commitment. So they go for broke and try to tackle a complicated project, with many twists and turns. The "big bang" approach promises big payoffs, but it tends to create big problems.

- A better way is to start with a small, greenfield project that lets the organization try out CD and get used to the new procedures. Try to pick a new area with new delivery expectations that isn't tied to a legacy pipeline and an old set of procedures. Small, incremental changes to applications are easier to test - and easier to remedy if something goes wrong. Each change can be pushed through a pipeline more quickly, allowing organizations to do shorter, faster pipeline runs that can produce measureable, positive results.

Define a Process

- Once you've picked your initial project, you'll need to define the process. This is as simple as writing the procedures on a board. Those of us that have read Gene Kim's book The Phoenix Project are familiar with this step. The company highlighted in the book was encountering every IT delivery problem imaginable—until Bill Palmer, VP of IT, implemented continuous delivery. The first step was to get each staffer to write out steps in the delivery process on a whiteboard and think about how to link them, creating an assembly line process. The team then rolled up their collective sleeves, created a workflow and automated it.

- The fact is, you can buy a great set of tools, create an aggressive set of goals and get your team to buy in. But until you map out a process, understand the process and assign roles, you can't get started.

Ensure a Blameless Culture

- One prerequisite to a CD implementation is a requirement that the development, QA and operations teams need to have shared goals. That's essential. But it's not the end of the "people work." As you implement CD, you should do an ongoing check that you're truly promoting a blameless culture. Issues will crop up in any CD implementation, and your organization needs to ensure that it can triage them in a positive way without having people point fingers at each other. Successful DevOps cultures accept failure and promote risk-taking. Moving to CD is a risk, and your team needs to keep its head in the game to continuously improve your processes.

Set Metrics and Measure Your Success

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

- A key aspect of continuous delivery is the ability to automate your configuration. This configuration-as-code DevOps practice ensures consistency in your CD process, clearing away problems that could result from rebuilding your configuration – potentially inconsistently - every time you want to push a release to production.

- If you're implementing CD, you'll want to make sure you're taking advantage of tools that enable configuration management – tools from Chef, Puppet and others. And we're seeing more DevOps operations implement these tools. According to a recent DZone survey sponsored by CloudBees, 49 percent of organizations use a configuration management tool and 48 percent use a version control system for infrastructure changes and system definition. But, on the flip side, 73 percent still have to use manual scripts for at least half of their infrastructure changes.

Orchestrating a Process

You've defined your pipeline; now you need to orchestrate it. This is a long process, but here are a few steps you'll want to take – as outlined in a whitepaper by Xebia Labs' Andrew Phillips and CloudBees' Kohsuke Kawaguchi.

- **Ensure reproducible builds** – Configure the build system to use a clean repository connected to the build job's workspace and use a central shared repository for build dependencies.

- **Share build artifacts through the pipeline** – Ensure the candidate artefacts is used by all subsequent builds in your pipeline.

- **Choose the right granularity for each job** – Distribute all steps in the pipeline across multiple jobs, allowing you to identify bottlenecks more easily.

- **Visualize the pipeline** – Create a clear, accessible view of the build pipeline to allow for smooth status communications and transparency of the process to the business managers and other stakeholders.

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT | Microsoft Partner
Gold Cloud Platform
Silver Learning

Thank You