

# **Unit II Agile Development Process**

**Prof. M.P.Karnik**  
**madhuri.chavan@viit.ac.in**  
**Department of Computer Engineering**



**BRAC'T'S, Vishwakarma Institute of Information Technology, Pune-48**

**(An Autonomous Institute affiliated to Savitribai Phule Pune University)**  
**(NBA and NAAC accredited, ISO 9001:2015 certified)**

# Unit II Agile Development Process

- Agile Development: Agile manifesto, agility and cost of change,
- Agile principles, myth of planned development
- Tools for Agile Project Management
- Scrum- process flow, scrum roles, events and artifacts
- Scrum cycle description, product backlog, sprint planning meeting
- sprint backlog, sprint execution, daily scrum meeting
- maintaining sprint backlog and burn-down chart, sprint review and retrospective.
- Agile Practices: test driven development, refactoring, pair programming, continuous integration, exploratory testing versus scripted testing

# What is Agile?

- Agile --readiness for motion, move quickly and easily, dexterity in motion
- Agility
  - The ability to both create and respond to change in order to profit in a turbulent business environment
    - Companies need to determine the amount of agility they need to be competitive.

# Agile Software Development

- **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, which may last from one to four weeks.
- Agile methods also emphasize working software as the primary measure of progress.

# Agile Software Development: Intro

- Characteristics of Agile Software Development
  - Light Weighted methodology
  - Small to medium sized teams
  - vague and/or changing requirements
  - vague and/or changing techniques
  - Simple design

# 1. Agile Software Development

- **Agile software development** is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
  - Methods
  - Iterative
  - Incremental
- It promotes **adaptive** planning, **evolutionary** development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to **change**.
- It is a conceptual framework that promotes **foreseen interactions** throughout the development cycle.
- The *Agile Manifesto*<sup>[</sup> introduced the term in 2001. (Wiki, 21 Aug 12)
- Let's take this definition apart.

## a. Software Development Method

- A **software development methodology** or **system development methodology** in software engineering is a framework that is used to structure, plan, and control the process of developing an information system.

## b. Software Engineering / Software Development

- **Software Engineering** (WIKI) (SE) is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.
- **Software development** is the process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in creating and maintaining applications, frameworks, or other software components.(WIKI)



## c. Information System

- An **information system** (IS) - is any **combination** of information technology (IT) and people's **activities** that support operations, management and decision making.
- In a very broad sense, the term information system is frequently used to refer to the **interaction** between people, processes, data and technology.
- In this sense, the term is used to refer
  - **not only** to the information and communication technology (ICT) that an organization uses,
  - **but also** to the way in which **people interact** with this technology in support of business processes. (Wiki)

# Iterative and Incremental Development

Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model.

It starts with an initial planning and ends with deployment with the cyclic interactions in between.

**Iterative and incremental development are essential parts of the Rational Unified Process, Extreme Programming and generally the various agile software development frameworks.**

It follows a similar process to the “plan-do-check-act” cycle of **business process improvement**.

## e. Time-Boxed Approach

- In time management, a **time box** allots a **fixed** period of time for an activity.
- **Timeboxing** plans activity by allocating time boxes.

## 2. Introductory Thoughts

- Fears regarding software development led to a number of pioneers / industry experts to develop the **Agile Manifesto** based up some firm values and principles.
- Practitioners had become afraid that repeated software failures could not be stopped without some kind of **guiding process** to guide development activities.

# Common Fears

- Practitioners were afraid that
  - The project will produce the wrong product
  - The project will produce a product of poor quality
  - The project will be late
  - We'll have to work 80 hour weeks
  - We'll have to break commitments
  - We won't be having fun.

# Agile Alliance/Agreement

- Several individuals, **The Agile Alliance**,
  - motivated to make activities
  - such that certain outputs and artifacts are **predictably** produced.
  - Around 2000, these notables got together to address common development problems.
- Goal: outline values and principles to allow software teams to
  - **develop quickly** and
  - **respond to change.**

- These activities stand up in large part to **runaway processes**.
  - Failure to achieve certain goals was met with ‘more process.’ Schedules slipped; budgets bloated, and processes became even larger.
- The Alliance (17) created a statement of **values**: termed the **manifesto** of the Agile Alliance.
- They then developed the **12 Principles of Agility**.

# The Manifesto for Agile Software Development

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

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

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

***Kent Beck et al***



## Value 1: Individuals and Interactions over Processes and Tools

- **Strong players:** a must, but can fail if don't work together.
- **Strong player:** not necessarily an 'ace;' work well with others!
  - Communication and interacting is **more important** than raw talent.
- **'Right' tools** are vital to smooth functioning of a team.
- **Start small.** Find a free tool and use until you can demo you've expand it. Don't assume bigger is better. Start with white board; flat files before going to a huge database.
- **Building a team** more important than **building environment**.
  - Some managers build the environment and expect the team to fall together.
  - Doesn't work.
  - Let the team build the environment on the **basis of need**.

## Value 2: Working Software over Comprehensive Documentation

- **Code** – not ideal medium for communicating rationale and system structure.
  - Team needs to produce human readable documents describing system and design decision rationale.
- **Too much documentation is worse than too little.**
  - Take time; more to keep in sync with code; Not kept in sync? it is a lie and misleading.
- **Short rationale and structure document.**
  - Keep this in sync; Only highest level structure in the system kept.

## Value 3: Customer Collaboration over Contract Negotiation

- Not possible to describe software requirements up front and leave someone else to develop it within cost and on time.
- Customers cannot just cite needs and go away
- Successful projects require **customer feedback on a regular and frequent basis** – and not dependent upon a contract.

## Value 4: Responding to Change over Following a Plan

- Course of a project cannot be predicted far into the future.
- **Better planning strategy – make detailed plans for the next few weeks, very rough plans for the next few months, and extremely rough plans beyond that.**
- Need to know what we will be working on the next few weeks; roughly for the next few months; a vague idea what system will do after a year.
- **Only invest in a detailed plan for immediate tasks;** once plan is made, difficult to change due to commitment.
  - But rest of plan remains flexible. The lower resolution parts of the plan can be changed with relative ease.

# What is Agility?

- Effective (rapid and adaptive) **response to change** (team members, new technology, requirements)
- Effective **communication** in structure and attitudes among all team members, technological and business people, software engineers and managers.
- Drawing the **customer into the team**.  
Planning in an uncertain world has its limits and plan must be **flexible**.
- Organizing a team so that it is in control of the work performed
- **Rapid, incremental delivery of software**

# An Agile Process

- Is driven by **customer descriptions** of what is required (scenarios). Some assumptions:
  - Recognizes that plans are **short-lived** (some requirements will persist, some will change. Customer priorities will change)
  - Develops software **iteratively** with a heavy emphasis on **construction** activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created. )
  - Analysis, design, construction and testing are not predictable.
- Thus has to **Adapt** as changes occur due to unpredictability
- Delivers multiple 'software **increments**', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

# Extreme programming

- Perhaps the best-known and most widely used agile method is XP.
- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

# Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.
- XP Planning
  - Begins with the listening, leads to creation of “**user stories**” that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a **cost** (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a **deliverable increment next release**.
  - A **commitment** (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. high priority stories first, or 3. the riskiest stories will be implemented first.
  - After the first increment “**project velocity**”, namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments. Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.



# Extreme Programming (XP)

- XP Design ( occurs both before and after coding as refactoring is encouraged)
  - Follows the **KIS principle (keep it simple)** Nothing more nothing less than the story.
  - Encourage the use of **CRC (class-responsibility-collaborator) cards** in an object-oriented context. The only design work product of XP. They identify and organize the classes that are relevant to the current software increment.
  - For difficult design problems, suggests the creation of “**spike solutions**”—a design prototype for that portion is implemented and evaluated.
  - Encourages “**refactoring**”—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read.
- XP Coding
  - Recommends the **construction of a unit test** for a story *before* coding starts. So implementer can focus on what must be implemented to pass the test.
  - Encourages “**pair programming**”. Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles.
- XP Testing
  - All **unit tests are executed daily** and ideally should be automated. Regression tests are conducted to test current and previous components.
  - “**Acceptance tests**” are defined by the customer and executed to assess customer visible functionality.

# Extreme programming practices 1

---

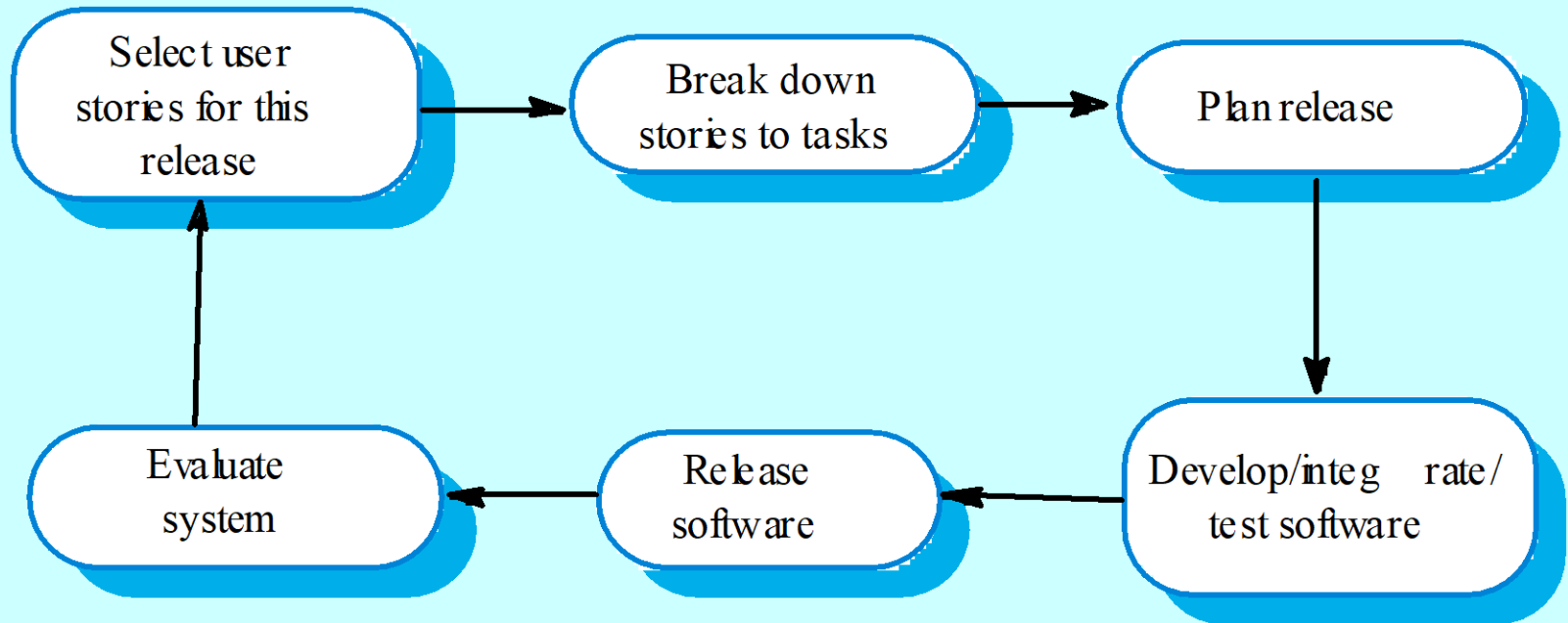
Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development Tasks
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

---

# Extreme programming practices 2

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

# The XP release cycle



# Agility Principles - I

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
2. **Welcome changing** requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together **daily** throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation.

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

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain **a constant pace** indefinitely.
9. Continuous attention to **technical excellence** and **good design** enhances agility.
10. **Simplicity** – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from **self-organizing** teams.
12. At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts** its behavior accordingly.

# Agile Principles (12)

- The following principles are those that differentiate agile processes from others.

# Principle 1: Our Highest Priority is to **Satisfy the Customer** through **Early** and **Continuous Delivery** of Valuable Software

- Number of practices have significant impact upon quality of final system:
- 1. Strong **correlation** between **quality** and **early delivery of a partially functioning system**.
  - The less functional the initial delivery, the higher the quality of the final delivery.
- 2. Another strong **correlation** exists between **final quality** and **frequently deliveries of increasing functionality**.
  - The more frequent the deliveries, the higher the final quality.
- **Agile processes deliver early and often**.
  - Simple system **first** followed by systems of **increasing functionality** every few weeks.
  - Customers may use these systems in production, or
  - May choose to review existing functionality and report on changes to be made.
  - Regardless, they must provide meaningful **feedback**.



**Principle 2: Welcome Changing Requirements**, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage.

- This is a statement of **attitude**.
- Participants in an agile process are **not afraid** of change.
  - Requirement changes are good;
  - Means team has learned more about what it will take to satisfy the market.
- Agile teams work to keep the **software structure flexible**, so requirement change impact is minimal.
- 
- Moreover, the **principles of object oriented** design help us to maintain this kind of flexibility.

## Principle 3: Deliver Working Software Frequently

(From a couple of weeks to a couple of months with a preference to the shorter time scale.)

- We deliver working software.
  - **Deliver early and often.**
- The **goal** of delivering software that satisfies the customer's needs.

## Principle 4: Business People and Developers Must **Work Together Daily** throughout the Project.

- For agile projects, there must be **significant** and **frequent interaction** between the
  - customers,
  - developers, and
  - stakeholders.

An agile project must be **continuously guided**.

**Principle 5: Build Projects around Motivated Individuals.** (Give them the environment and support they need, and trust them to get the job done.)

- **An agile project has people the most important factor of success.**
  - All other factors, process, environment, management, etc., are considered to be second order effects, and are subject to change if they are having an adverse effect upon the people.
- **Example:** if the office environment is an obstacle to the team, **change the office environment.**
- If certain process steps are obstacles to the team, **change the process steps.**

## **Principle 6: The Most Efficient and Effective Method of Conveying Information to and within a Development Team is face-to-face Communications.**

- In agile projects, developers ***talk*** to each other.
  - The **primary mode of communication is conversation.**
  - Documents may be created, but there is no attempt to capture all project information in writing.
- An agile project team **does not demand written specs, written plans, or written designs.**
  - They may create them if they perceive **an immediate and significant need**, but they are not the default.
  - The **default is conversation.**

## **Principle 7: Working Software is the Primary Measure of Progress**

- Agile projects measure their progress by measuring the amount of **working software**.
  - Progress **not measured** by phase we are in, **or**
  - by the volume of produced documentation **or**
  - by the amount of code they have created.
- **Agile teams are 30% done when 30% of the necessary functionality is working.**

## Principle 8: Agile Processes promote sustainable development

The sponsors, developers, and users should be able to **maintain a constant pace** indefinitely.

- An agile project is not run like a 50 yard dash; it is run like a marathon.
  - The team does not take off at full speed and try to maintain that speed for the duration.
  - Rather they run at a fast, but sustainable, pace.
- Running too fast leads to stress, shortcuts.
- Agile teams pace themselves.
  - They don't allow themselves to get too tired.
  - They don't borrow tomorrow's energy to get a bit more done today.
  - They work at a **rate** that allows them to maintain the highest quality standards for the duration of the project.

## Principle 9: Continuous Attention to Technical Excellence and Good Design enhances Agility.

- **High quality is the key to high speed.**
  - The way to go fast is to **keep the software as clean and robust as possible.**
  - Thus, all agile team-members are **committed** to producing only the **highest quality code** they can.
  - They do not make messes and then tell themselves they'll clean it up when they have more time.
  - **Do it right the first time!**



## **Principle 10: Simplicity – the art of maximizing the amount of work not done – is essential.**

- Agile teams take the simplest path that is consistent with their goals.
  - They don't anticipate tomorrow's problems and try to defend against them today.
  - **Rather they do the simplest and highest quality work today, confident that it will be easy to change if and when tomorrows problems arise.**

## Principle 11: The Best Architectures, Requirements, and Designs emerge from **Self-Organizing Teams**

- An agile team is a self organizing team.
  - Responsibilities are **not handed to individual team members** from the outside.
  - Responsibilities are **communicated** to the team as a whole, and the **team determines** the best way to fulfill them.
- Agile team members work together on all project aspects.
  - Each is allowed input into the whole.
  - No single team member is responsible for the architecture, or the requirements, or the tests, etc.
  - The team shares those responsibilities and each team member has influence over them.

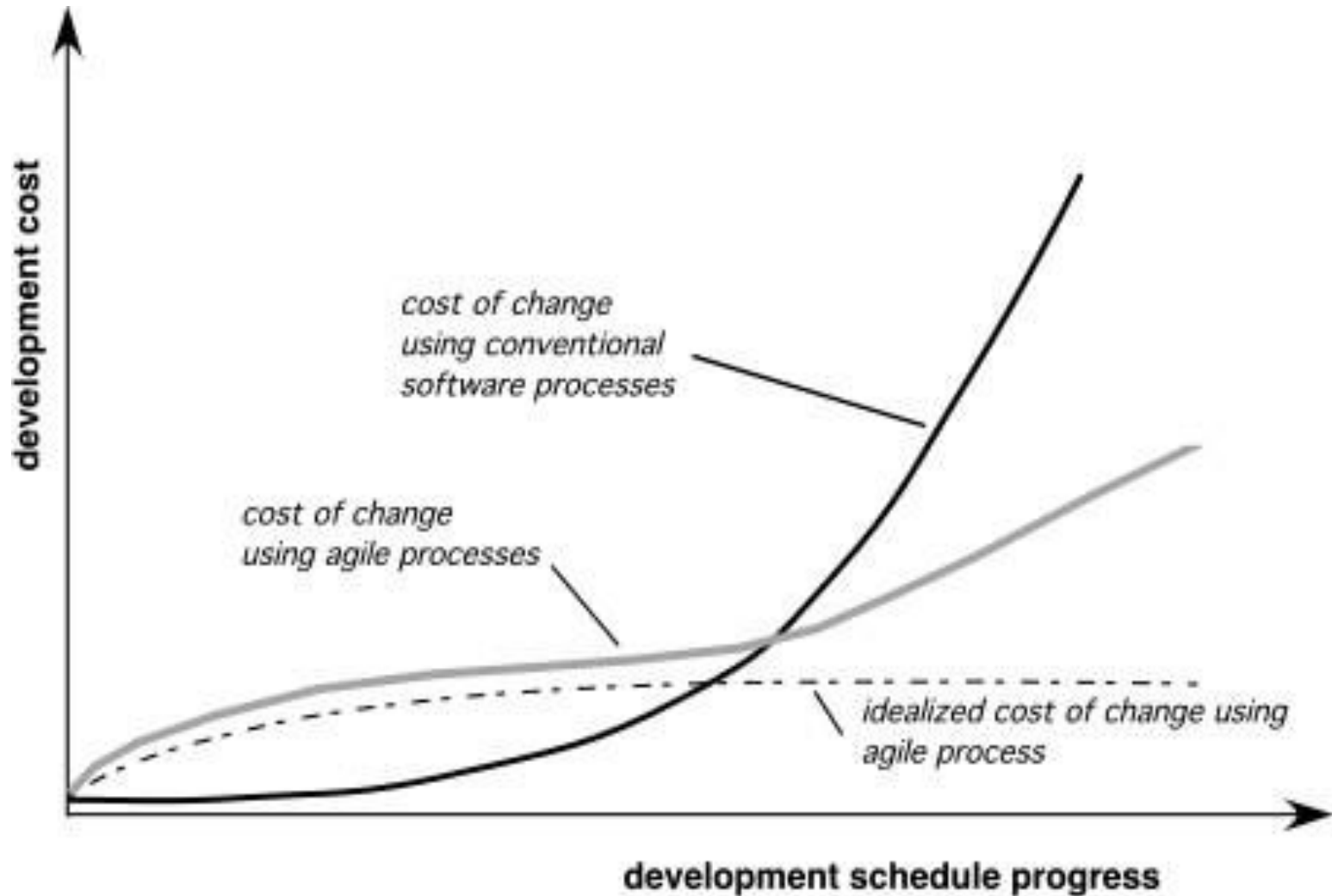
**Principle 12:** At regular Intervals, the **Team reflects** on how to become **more** effective, **then tunes and adjusts** its **behavior** accordingly.

- An agile team continually adjusts its organization, rules, conventions, relationships, etc.
- An agile team knows that its environment is continuously changing, and knows that they must change with that environment to remain agile.

# Agility and the Cost of Change

- **Conventional wisdom** is that the cost of change increases nonlinearly as a project progresses. It is relatively easy to accommodate a change when a team is gathering requirements early in a project. If there are any changes, the costs of doing this work are minimal. But if the middle of validation testing, a stakeholder is requesting a major functional change. Then the change requires a modification to the architectural design, construction of new components, changes to other existing components, new testing and so on. Costs escalate quickly.
- A well-designed **agile process** may “**flatten**” the cost of change curve by coupling **incremental delivery** with agile practices such as **continuous unit testing** and **pair programming**. Thus team can accommodate changes late in the software project without dramatic cost and time impact.

# Agility and the Cost of Change



# Tools for Agile Project Management

In agile development, the emphasis is on building the right product as per customer needs. Therefore, the agile testers need to monitor their project constantly. There are many tools available for this purpose.

## 1. Agilean

First one on the list of agile project management tools is Agilean. It is a SaaS enterprise workflow automation and project management software solution that is basically created to be used by small-medium IT enterprises. The main features of Agilean include project planning, execution, monitor, response plan, stand up meeting automation, release management, retrospective analysis, and visualized reports.

Another great one from the list of the best agile project management tools is Wrike that is one of the best in terms of integrating email with project management, having main features inside. It is built to scale and drive results by giving you the flexibility you might need to manage multiple projects and teams at one place.

Along the Agile process, you going to get the accurate, up-to-date information and you can insert & add any important information inside. Planning should be easier, there will be always accurate information and real-time reports and analytics that going to save your time and help in analyzing the situation.

JIRA is a tool developed for bug and issue tracking and project management to software and mobile development processes. The JIRA dashboard has many useful functions & features which are able to handle different issues easily. The dashboard on JIRA can be customized to match your business processes.

JIRA is a defect tracking tool which is used for Agile testing as well as project management. This tool is not only used for recording, reporting but also integrated with code development environment.

### **Features:**

JIRA Query Language helps to create quick filters with a single click

Estimations help your team become more accurate and efficient

Reporting functionality gives team critical insight into their agile process

Extensive reporting functionality gives your team critical insight into their agile process.



Trello, one of the most used and well-known project management applications. The structure of Trello is based on the kanban methodology. All the projects are represented by boards, that contain lists.

Every list has progressive cards that you make as drag-and-drop. Users that are related to the board can be assigned to said cards. Also, it has many nice, small but not less useful features: writing comments, inserting attachments, notes, due dates, checklists, colored labels, integration with other apps, etc. Additionally, Trello is supported by all mobile platforms.

## 5. Backlog

Backlog is an all-in-one online project management tool for task management, version control, and bug tracking. With features like subtasking, custom issue fields, and Gantt charts, it's easy for teams to define, organize, and track their work.

Bringing together the organizational benefits of project management with the power and convenience of code management, Backlog enhances team collaboration across organizations large and small.

## 6. Jmeter:

The Apache JMeter application is an open source agile performance testing tool. It is used to load functional test behavior and measure performance of the website.

### Features:

- Ability to load and performance test different applications/server and protocols
- Full featured Test IDE for fast Test Plan recording
- It offers complete portability and 100% Java purity
- Data analysis and visualization plugins offers great extensibility
- Functions can be used to offer dynamic input to test or provide data manipulation
- Easy Continuous Integration using third party libraries for tools like Maven, Gradle, and Jenkins

7. Pivotal Tracker is a tool that helps developers for planning project for software development. It's mainly based on agile development methods. However, it works effectively with all kinds of projects.

**Features:**

Support for ActiveResource

Ability to get a list of all the projects

Project transparency at a glance

Move and edit multiple stories at once

## ➤ List of other tools for Agile project management

1. Kanbanize
  2. Assembla
  3. Appium:
  4. Selenium
  5. Bug Shooting
  6. Usersnap
  7. SoapUI
  8. qTest Scenario
  9. Qmetry
  10. QAccomplete
  11. Enterprise Tester:
- ETC...**

# Agile Methods

- Agile methods:
  - **Scrum**
  - **Extreme Programming**
  - Adaptive Software Development (ASD)
  - Dynamic System Development Method (DSDM)

# Scrum - an agile process

- **SCRUM** is an agile, lightweight process for managing and controlling software and product development in rapidly changing environments.
  - Iterative, incremental process
  - Team-based approach
  - developing systems/ products with rapidly changing requirements
  - Controls the confusion of conflicting interest and needs
  - Improve communication and maximize cooperation
  - Protecting the team from disturbances
  - A way to maximize productivity

# Scrum

## ■ Scrum—distinguishing features

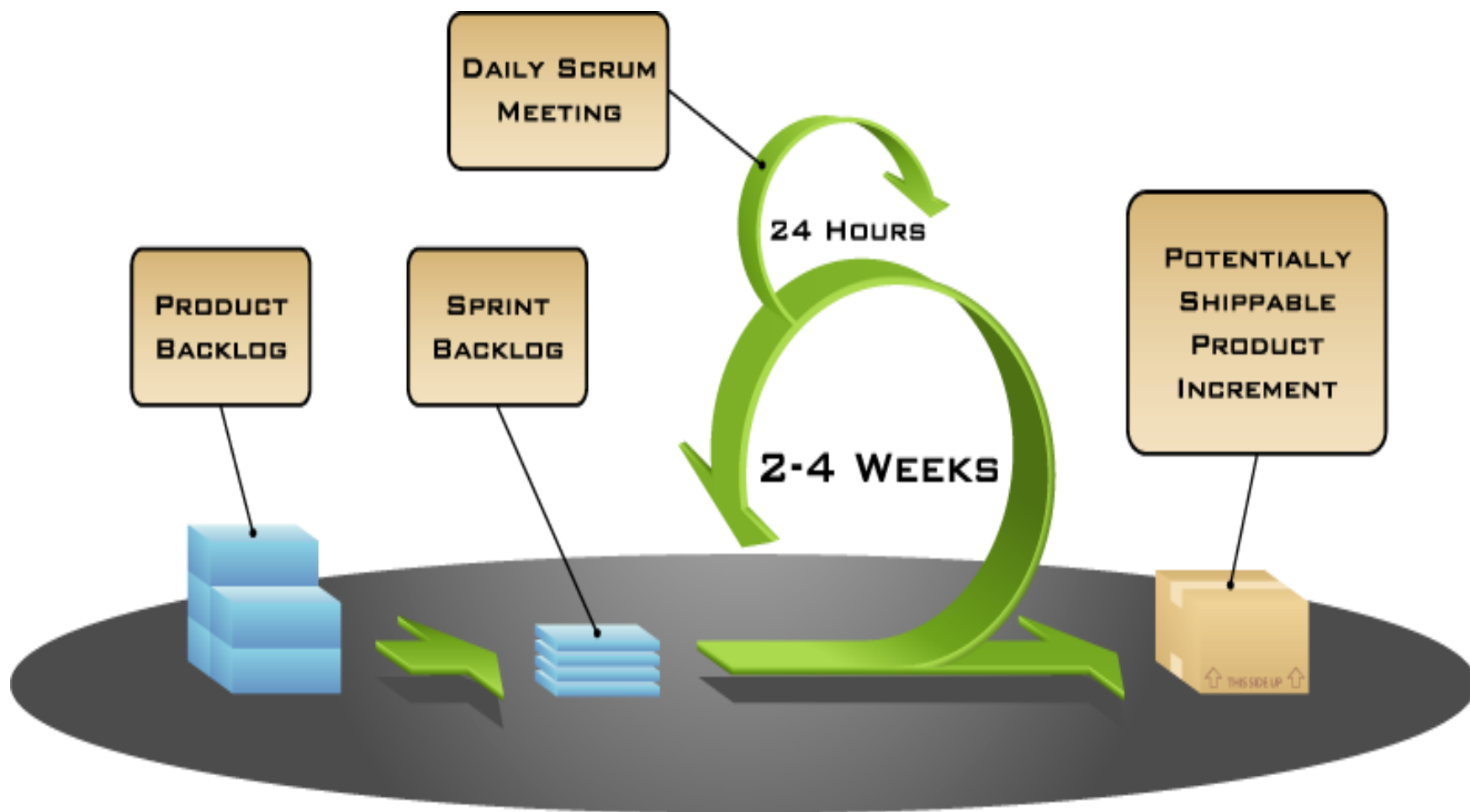
- Development work is partitioned into “**packets**”
- **Testing and documentation are on-going** as the product is constructed
- Work units occurs in “**sprints**” and is derived from a “**backlog**” of existing changing prioritized requirements
- Changes are not introduced in sprints (short term but stable) but in backlog.
- **Meetings are very short** (15 minutes daily) and sometimes conducted without chairs ( what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)
- “**demos**” are delivered to the customer with the time-box allocated. May not contain all functionalities. So customers can evaluate and give feedbacks.



# Characteristics

- Self-organizing teams.
- Product progresses in a series of month-long “sprints”.
- Requirements are captured as items in a list of “product backlog”
- No specific engineering practices prescribed.
- Uses generative rules to create an agile environment for delivering projects.
- One of the “agile processes”.

# Functionality of Scrum



# Project Management Emphasis based on a Standard 30-day Sprint

- **Scrum:** a definite project management emphasis.
- **Scrum Master:** A **Scrum project** is managed by a Scrum Master, who can be considered as much a consultant or coach as a manager.
- **Sprint.** Scrum has a fundamental 30-day development cycle called a **Sprint**, preceded by
  - **pre-Sprint** activities and **post-Sprint** activities.
- **Daily Scrum:** A short (less than 30 minutes) daily Scrum Meeting allows the team to monitor status and communicate problems.

# Product Backlog for Planning

- **Project planning** is based on a **Product Backlog**, which contains
  - functions and
  - technology enhancements
- intended for the project.
- Two meetings are held –
  - one to decide the **features for the next Sprint** and
  - the other to **plan out the work**.

# Scrum - Queues

- Product Backlog → Sprint Backlog → Sprint → Working increment of the Software
- **Scrum** uses **lightweight queue-based management** and work-breakdown mechanisms.
- **Product Backlog queue**: a low-tech customer-managed queue of demand requests for products.
- **Sprint**: At launch time, a Sprint (30-day time-boxed iteration) does **just-in-time planning**
- **Sprint Backlog**: queue for Sprint work-mgmt.

# Scrum - Management

- **Daily Scrum:** Very notable and very visible
- Is a **daily standup**,
  - **except** that it is the **team** that is participating and sharing coordination information **not** a **central project manager**.
- **Scrum Master**
  - holds daily scrum and
  - acts more as a **facilitator** and **runs interference** for the core team when **blocks** or **issues** arise.  
(Kennale, SDLC 3.0, p. 31)

# Components of Scrum

- Scrum Roles
- The Process
- Scrum Artifacts

# Core Roles – Scrum Master

- **Scrum** is facilitated by a Scrum Master –
- Accountable for **removing impediments** for team to deliver sprint goal / deliverables.
- **Scrum Master is not the team leader**, but acts as a **buffer** between the team and any distracting influences.
- Scrum Master ensures **process** is used as intended.
- Scrum Master is the **enforcer of rules**.
- Scrum Master's role: **protect** the Team and keep it **focused** on the tasks at hand.



# The Scrum Master

- Represents management to the project
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

# Core Roles – Development Team

- The Development Team is responsible for **delivering potentially shippable product increments** at end of each Sprint.
- Cross-functional
  - QA, Programmers, UI Designers, etc.
- Team = 3–9 people with cross-functional skills.
- Team does actual work
  - (analyze, design, develop, test, technical communication, document, etc.).
- Team is **self-organizing**, even though they may interface with project management organizations (PMOs).
- Membership can change only between sprints

# Core Roles – Product Owner

- The Product Owner represents **stakeholders** and is the **voice of the customer**.
- Product Owner is **accountable** for ensuring that the team **delivers value** to the business.
- **Product Owner**
  - **writes** customer-centric items (typically **user stories**),
  - **prioritizes** them, and
  - **adds** them to the **product backlog**.

Note:

- Scrum teams should have **one Product Owner**.
- May also be a member of the development team
- Not recommend this person be Scrum Master.

# Product Owner

- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results.
- Knows what needs to be build and in what sequence this should be done
- Typically a product manager

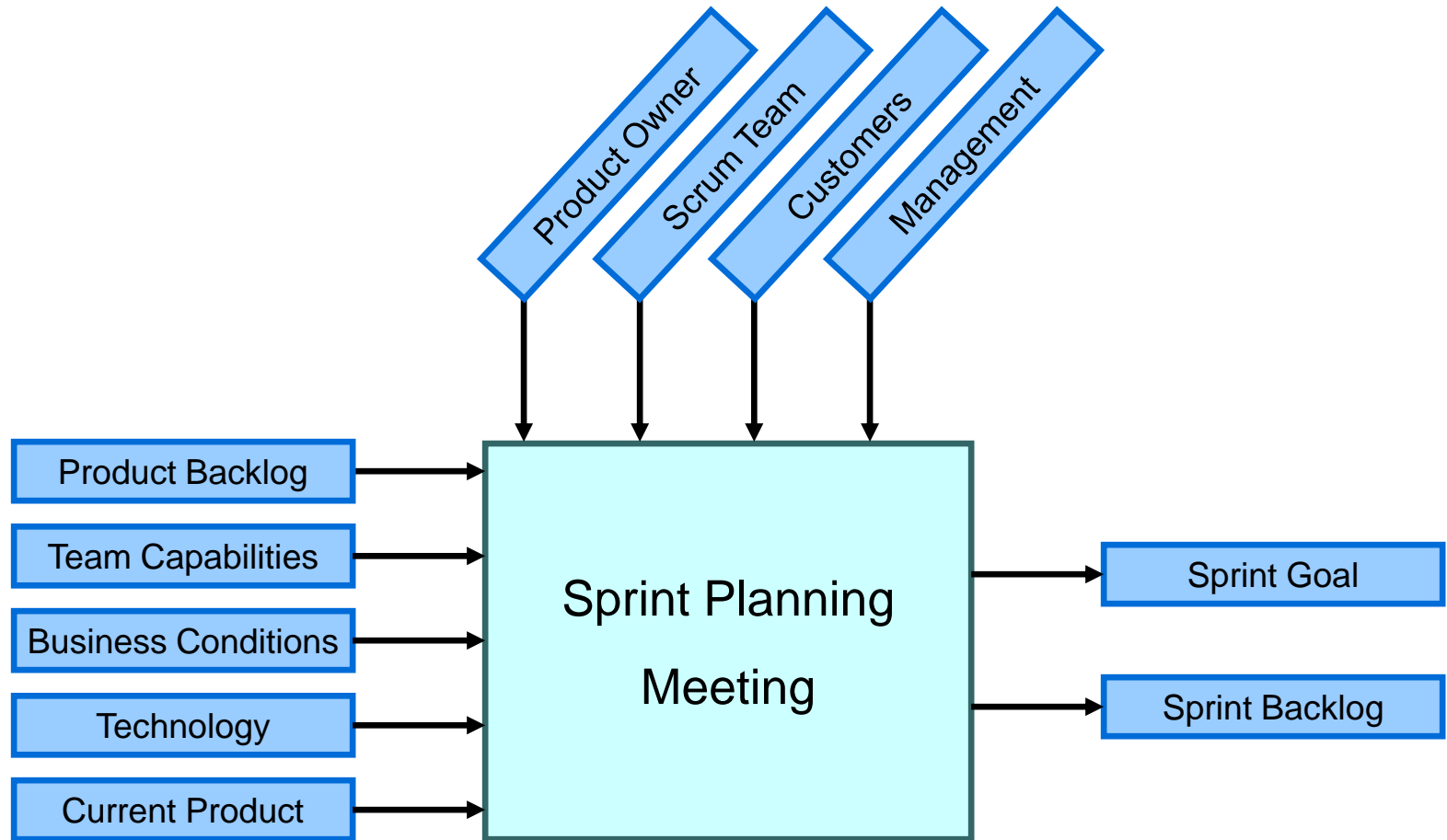
# The Process

- Sprint Planning Meeting
- Sprint
- Daily Scrum
- Sprint Review Meeting

# Sprint Planning Meeting

- A collaborative meeting in the beginning of each Sprint between the Product Owner, the Scrum Master and the Team.
- Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)

# Sprint Planning Meeting



# Parts of Sprint Planning Meeting

- 1<sup>st</sup> Part:
  - Creating Product Backlog
  - Determining the Sprint Goal.
  - Participants: Product Owner, Scrum Master, Scrum Team
- 2<sup>nd</sup> Part:
  - Participants: Scrum Master, Scrum Team
  - Creating Sprint Backlog



# Pre-Project/Kickoff Meeting

- A special form of Sprint Planning Meeting.
- Meeting before the begin of the Project.

# Sprint

- A month-long iteration, during which is incremented a product functionality
- NO outside influence can interference with the Scrum team during the Sprint
- Each Sprint begins with the Daily Scrum Meeting

# Sprints

- Scrum projects make progress in a series of “sprints”
  - Analogous to XP iterations.
- Target duration is one month
  - +/- a week or two
    - But, a constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint.

- **Sprint:** basic unit of development in Scrum.
- **Sprint duration:** one week to one month;
- **“Time Boxed”** effort of a constant length.
  
- Each sprint:
- Preceded by a **planning meeting**,
  - where the **tasks** for sprint are **identified** and an **estimated commitment for the sprint goal** made, and followed by a **review or retrospective meeting**, where the progress is reviewed and lessons for the next sprint are identified.

- During Sprint, **team creates finished portions** of a product. (an **increment**)
- **Features** going into a Sprint come from the ***product backlog***: a prioritized list of requirements.
  - **Which backlog items** go into sprint (**sprint goals**) determined during Sprint Planning Mtg.
- **Sprint Goal**
  - sets up **minimum success criterion** for the Sprint and
  - **keeps the team focused** on the broader picture rather than narrowly on the task at hand.

- The **team** then determines **how many selected items** can be **completed** during the next sprint.
- These then go into the **Sprint Backlog**.
- **Sprint Backlog** is **property** of the development team, During a sprint, **no one is allowed to edit the sprint backlog except for development team**.
- **Development: time-boxed**; Sprint **must** end on time;
- Requirements not completed for any reason?  
are omitted and **returned** to **Product Backlog**.
- When Sprint is done, team **demonstrates** software.

# Daily Scrum

- Is a short (15 minutes long) meeting, which is held every day before the Team starts working
- Participants: Scrum Master (which is the chairman), Scrum Team
- Every Team member should answer on 3 questions

# Daily Scrum

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team



# Questions

- What did you do since the last Scrum?
- What are you doing until the next Scrum?
- What is stopping you getting on with the work?

# Sprint Review Meeting

- Is held at the end of each Sprint
- Business functionality which was created during the Sprint is demonstrated to the Product Owner
- Informal, should not distract Team members of doing their work

# Sprint Retrospective Meeting

- Scrum Team only
- Feedback meeting
- Three questions
  - Start
  - Stop
  - Continue
- Don't skip for the first 5-6 sprints!!!

# Scrum Artifacts

- Product Backlog
- Sprint Backlog
- Burn down Charts

# Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
- Is managed and owned by a Product Owner
- Spreadsheet (typically)
- Usually is created during the Sprint Planning Meeting
- Can be changed and re-prioritized before each Planning Meeting

# Product Backlog

- A list of all desired work on the project
  - Usually a combination of
    - story-based work (“let user search and replace”)
    - task-based work (“improve exception handling”)
- List is prioritized by the Product Owner
  - Typically a Product Manager, Marketing, Internal Customer, etc.

# Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
- Is managed and owned by a Product Owner
- Spreadsheet (typically)
- Usually is created during the Sprint Planning Meeting
- Can be changed and re-prioritized before each PM

# Artifact: Product Backlog

- **Product backlog** is an ordered list of "requirements" that is maintained for a product
- Contains Product Backlog Items **ordered** by the Product Owner based on
  - considerations like risk,
  - business value,
  - dependencies,
  - date needed, etc.
- **Features** added to backlog commonly written in story format
- The product backlog is the “**What**” that will be built, sorted in the relative order it should be built in.
  - **Product Owner is ultimately responsible for ordering the stories** on the backlog for the Development Team.



# Artifact: Product Backlog

- The product backlog contains rough estimates of both business value and development effort, these values are often stated in **story points** using a rounded **Fibonacci** sequence.
- Those **estimates** help the Product Owner to gauge the timeline and may influence ordering of backlog items.
  - Example, if the “add spellcheck” and “add table support” features have the same business value, the one with the smallest development effort will probably have higher priority, because the Return on Investment is higher.

## Artifacts – The Product Backlog 2

- **Product Owner:** responsible for **the product backlog** and the **business value** of each item listed.
- **Development Team:** responsible for the **estimated effort** to complete each backlog item.
- Team contributes by estimating Items and User-Stories, either in “**Story-points**” or in “**estimated hours.**”

# Sprint Backlog

- A subset of Product Backlog Items, which define the work for a Sprint
- Is created ONLY by Team members
- Each Item has it's own status
- Should be updated every day

# Sprint Backlog

- No more than 300 tasks in the list
- If a task requires more than 16 hours, it should be broken down
- Team can add or subtract items from the list. Product Owner is not allowed to do it.
- Is a good warning monitor

# From Sprint Goal to Sprint Backlog

- Scrum team takes the Sprint Goal and decides what tasks are necessary
- Team self-organizes around how they'll meet the Sprint Goal
  - Manager doesn't assign tasks to individuals
- Managers don't make decisions for the team
- Sprint Backlog is created

# Sprint Backlog during the Sprint

- Changes
  - Team adds new tasks whenever they need to in order to meet the Sprint Goal
  - Team can remove unnecessary tasks
  - But: Sprint Backlog can only be updated by the team
- Estimates are updated whenever there's new information

# Sprint Backlog

- A subset of Product Backlog Items, which define the work for a Sprint
- Is created **ONLY** by Team members
- Each Item has it's own status
- Should be updated every day

# Sample Sprint Backlog

		Days Left in Sprint	15	13	10	8	
Who	Description		7/22/2002	7/24/2002	7/26/2002	7/31/2002	
<b>Total Estimated Hours:</b>		<b>554</b>	<b>458</b>	<b>362</b>	<b>270</b>	<b>0</b>	
-	<b>User's Guide</b>	-	-	-	-	-	
SM	Start on Study Variable chapter first draft	16	16	16	16		
SM	Import chapter first draft	40	24	6	6		
SM	Export chapter first draft	24	24	24	6		
<b>Misc. Small Bugs</b>							
JM	Fix connection leak	40					
JM	Delete queries	8	8				
JM	Delete analysis	8	8				
TG	Fix tear-off messaging bug	8	8				
JM	View pedigree for kindred column in a result set	2	2	2	2		
AM	Derived kindred validation	8					
<b>Environment</b>							
TG	Install CVS	16	16				
TBD	Move code into CVS	40	40	40	40		
TBD	Move to JDK 1.4	8	8	8	8		
<b>Database</b>							
KH	Killing Oracle sessions	8	8	8	8		
KH	Finish 2.206 database patch	8	2				
KH	Make a 2.207 database patch	8	8	8	8		
KH	Figure out why 461 indexes are created	4					



# Artifacts: Sprint Backlog

- **Sprint Backlog:** list of work the Development Team must address during the next sprint.
- List derived by selecting stories/features from the top of the product backlog until the Development Team feels it has enough work to fill the sprint.
- **Thinking:** This is done by the Development Team asking "Can we also do this?" and adding stories/features to the sprint backlog.
- **History:** Development Team should note **velocity** of previous Sprints (total story points completed from each of the last sprints stories) when selecting stories/features for the **new sprint**.
- Use number as **guide** for "effort" they can complete.

# Artifacts: Sprint Backlog

- **Stories/features:** broken down into **tasks** by Development Team- should normally be between **four and sixteen hours** of work.
- With this level of detail the Development Team understands exactly what to do, and potentially, anyone can pick a task from the list.
- **Tasks** on sprint backlog are **never assigned**; tasks are **signed up for** by team members during **daily scrum**, according to **priority** and **member skills**.
- **Sprint backlog is property of Team**, and all included **estimates** are provided by the Development Team.

# Artifacts - Increment

- The “**increment**” is **sum** of all Product Backlog Items completed during a sprint **and** all previous sprints.
- At end of a sprint, Increment must be done according to **Scrum Team's definition** of done.
- The increment must be in **usable condition** regardless of whether the Product Owner decides to actually release it.

# Burn down Charts

- Are used to represent “work done”.
- Are wonderful Information Radiators
- 3 Types:
  - Sprint Burn down Chart (progress of the Sprint)
  - Release Burn down Chart (progress of release)
  - Product Burn down chart (progress of the Product)

# Artifacts: Burn Down

- The sprint **burn down chart** is a publicly displayed chart showing **remaining work** in the sprint backlog.
- Updated every day; gives a simple view of the sprint progress.
- Other types of burn down:
- **Release burn down chart:** shows amount of work left to **complete** the target commitment for a Product Release
  - This normally spans multiple iterations
- **Alternative Release burn down chart:** basically does the same, but clearly shows scope changes to Release Content, by resetting the baseline.

# Sprint Burn down Chart

- Depicts the total Sprint Backlog hours remaining per day
- Shows the estimated amount of time to release
- Ideally should burn down to zero to the end of the Sprint.

# Release Burn down Chart

- Will the release be done on right time?
- X-axis: sprints
- Y-axis: amount of hours remaining
- The estimated work remaining can also burn up

# Alternative Release Burn down Chart

- Consists of bars (one for each sprint)
- Values on the Y-axis: positive AND negative
- Is more informative then a simple chart



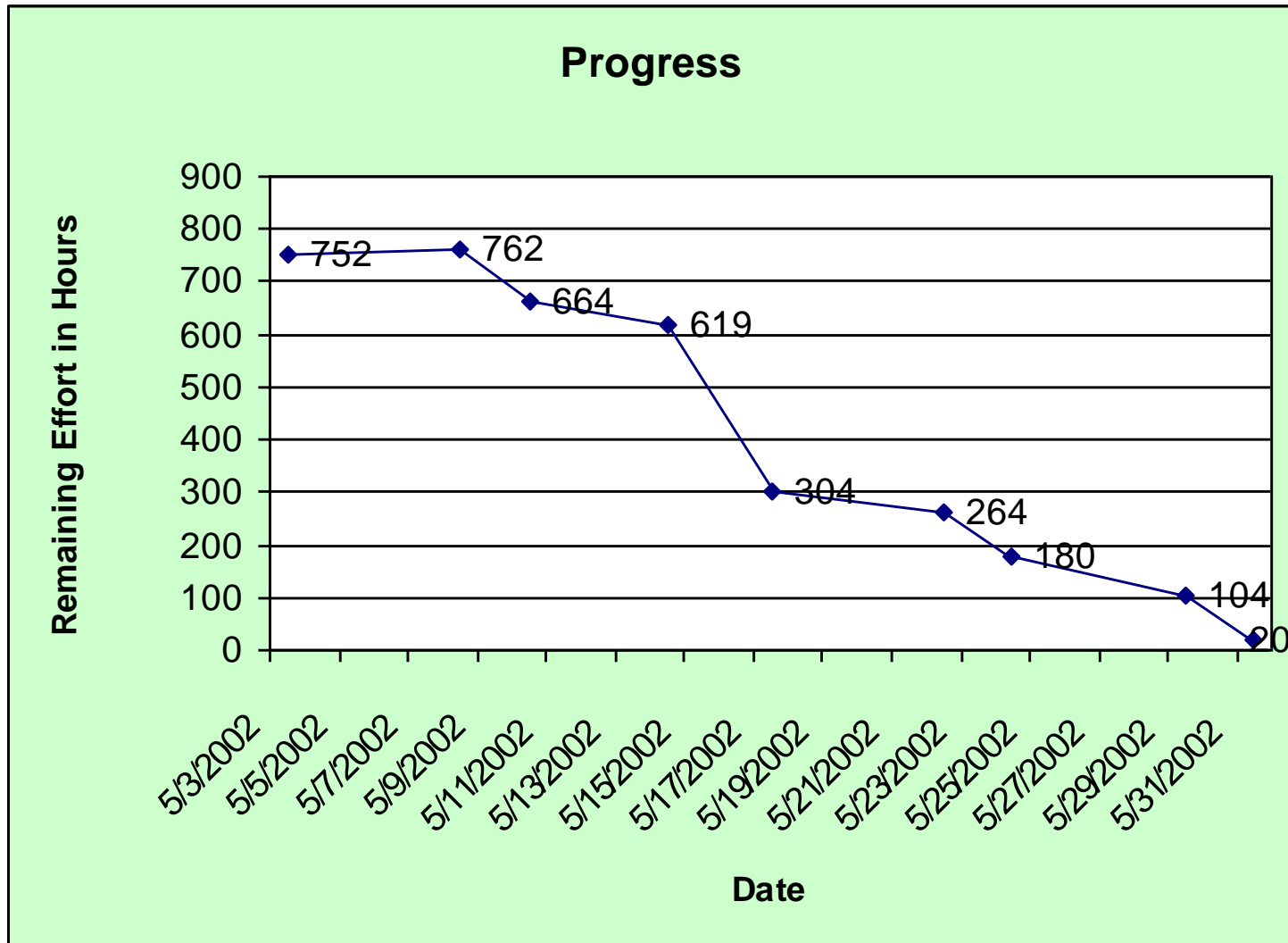
# Product Burn down Chart

- Is a “big picture” view of project’s progress (all the releases)

# Scaling Scrum

- A typical Scrum team is 6-10 people
- Jeff Sutherland - up to over 800 people
- "Scrum of Scrums" or what called "Meta-Scrum"
- Frequency of meetings is based on the degree of coupling between packets

# Sprint Burndown Chart



# XP@Scrum

Scrum is an effective project management wrapper for eXtreme Programming development practices, which enables agile projects to become scalable and developed by distributed teams of developers.

# Pros/Cons

## ■ Advantages

- *Completely developed and tested features in short iterations*
- *Simplicity of the process*
- *Clearly defined rules*
- *Increasing productivity*
- *Self-organizing*
- *each team member carries a lot of responsibility*
- *Improved communication*
- *Combination with Extreme Programming*

## ■ Drawbacks


- *“Undisciplined hacking” (no written documentation)*
- *Violation of responsibility*
- *Current mainly carried by the inventors*

**Rules**

 PO  
Product Owner:  
Set priorities

 SM  
ScrumMaster:  
Manage process,  
remove blocks

 T  
Team: Develop  
product

 SH  
Stakeholders:  
observe & advise

### Key Artifacts

#### Product Backlog

- List of requirements & issues
- Owned by Product Owner
- Anybody can add to it

#### Sprint Goal

- One-sentence summary
- Declared by Product Owner

#### Sprint Backlog

- List of tasks
- Owned by team

#### Blocks List

- List of blocks & unmade decisions
- Owned by ScrumMaster

#### Increment

- Version of the product
- Shippable functionality (tested,

### Key Meetings

#### Sprint Planning Meeting

- Hosted by ScrumMaster; ½-1 day
- In: Product Backlog, existing product, business & technology conditions

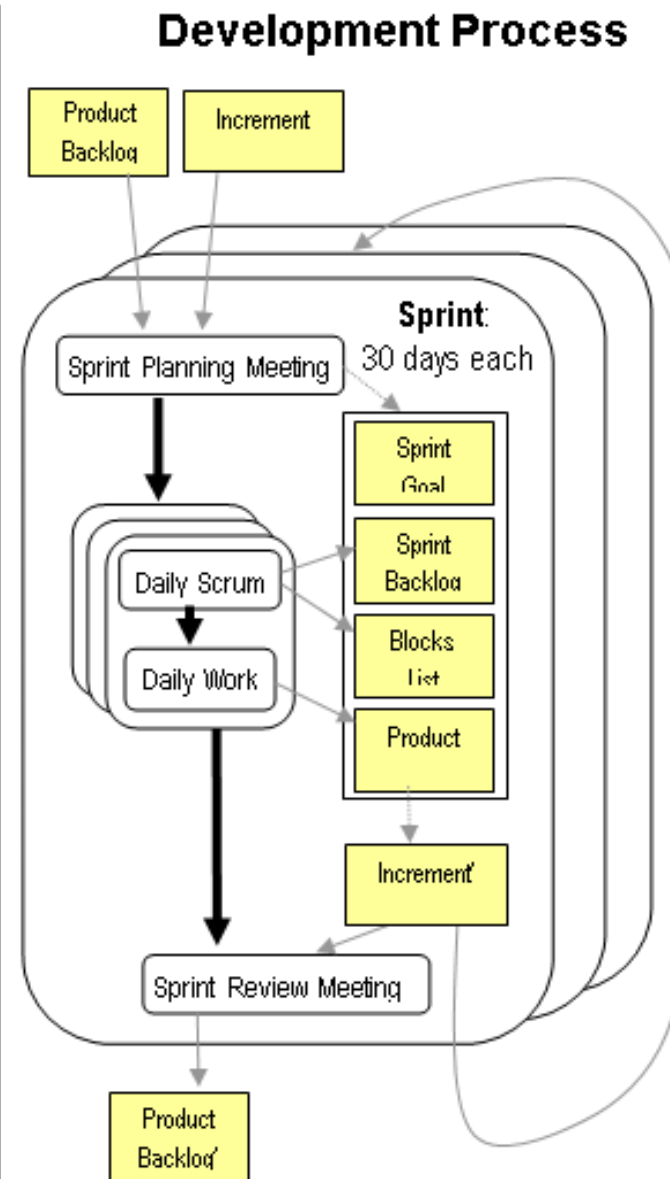
- Select highest priority items in Product Backlog; declare Sprint Goal
- Team turns selected items into

#### Daily Scrum

- Hosted by ScrumMaster
- Attended by all, but Stakeholders don't speak
- Same time every day
- Answer: 1) What did you do yesterday? 2) What will you do today? 3) What's in your way?
- Team updates Sprint Backlog;

#### Sprint Review Meeting

- Hosted by ScrumMaster
- Attended by all
- Informal, 4-hour, informational
- Team demos Increment
- All discuss
- Hold retrospective
- Announce next Sprint Planning



## Agile Practices:

- test driven development,
- refactoring
- pair programming
- continuous integration
- exploratory testing versus scripted testing

# Test driven development

- **TEST DRIVEN DEVELOPMENT (TDD)** approach first, the test is developed which specifies and validates what the code will do.
- It means, test cases are created before code is written. The purpose of TDD is to make the code clearer, simple and bug-free.
- Test-Driven Development starts with designing and developing tests for every small functionality of an application.
- TDD instructs developers to write new code only if an automated test has failed. This avoids duplication of code.



- The simple concept of TDD is to write and correct the failed tests before writing new code (before development).
- This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).
- Test-Driven development is a process of developing and running automated test before actual development of the application.
- Hence, TDD sometimes also called as **Test First Development**.

### ✓ **Example of TDD**

- A condition for Password acceptance:
- The password should be between 5 to 10 characters.

It is a process of modifying the code in order to pass a test designed previously.

- It more emphasis on production code rather than test case design.
- Test-driven development is a process of modifying the code in order to pass a test designed previously.
- TDD includes refactoring a code i.e. changing/adding some amount of code to the existing code without affecting the behavior of the code.
- TDD when used, the code becomes clearer and simple to understand.

# Refactoring

- **What is Refactoring ?**

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.

- **Why refactor?**

When you need to either fix a bug or add a new feature, then implementing the code necessary would be a lot easier once you had refactored it to be in a more stable, and understandable state.

The most suitable time to start refactoring is when you are fixing bugs or adding a new feature, and it should be done during the life cycle of the project not at the end.

- **Testing while Refactoring**

- Its important to perform regular tests during refactoring to make sure that you do not break anything during refactoring as refactoring main concern is not to change the API.

Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.

- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

# Examples of Refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

# Pair programming

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

All production code is written by two programmers sitting at one machine.

- This practice ensures that all code is reviewed as it is written and results in better Design, testing and better code.
- Some programmers object to pair programming without ever trying it.
  - It does take some practice to do well, and you need to do it well for a few weeks to see the results.
  - Ninety percent of programmers who learn pair programming prefer it, so it is recommended to all teams.
- Pairing, in addition to providing better code and tests, also serves to **communicate knowledge** throughout the team.

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.



In pair programming, programmers sit together at the same workstation to develop the software.

- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

# Continuous integration

- **Continuous Integration** (CI) is a development practice that requires developers to **integrate** code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.
- **Agile** methods allow software and systems teams to respond quickly to the changes. **Continuous Integration** helps systems development teams be **agile** and respond to rapid business changes, while at the same time ensuring that the actual hardware and software under development are in constant sync.

- With **continuous integration**, developers feed in their partially complete work back into the central repository on a regular basis and run tests.
- This helps developers identify what exactly broke a build and what steps to take to mitigate the error. **Continuous** running of tests makes the development process seamless.

# Exploratory testing versus scripted testing

- **EXPLORATORY TESTING** is a type of software testing where Test cases are not created in advance but testers check system on the fly.
- They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a "thinking" activity.
- Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning.
- It emphasizes personal freedom and responsibility of the individual tester.

- Under **scripted testing**, you design test cases first and later proceed with test execution.
- On the contrary, exploratory testing is a simultaneous process of test design and test execution all done at the same time.
- Scripted Test Execution is usually a non-thinking activity where testers execute the test steps and compare the actual results with expected results.
- Such test execution activity can be automated does not require many cognitive skills.

## Exploratory testing -

- Is not random testing but it is ad-hoc testing with a purpose of finding bugs
- Is cognitively (thinking) structured as compared to the procedural structure of scripted testing.
- It is not a technique but it is an approach. What actions you perform next is governed by what you are doing currently.

## When use exploratory testing?

- Exploratory testing can be used extensively when
  - The testing team has experienced testers
  - Early iteration is required
  - There is a critical application
  - New testers entered into the team
- In Software Engineering, Exploratory testing is performed to overcome the limitations of scripted testing. It helps in improving Test Case suite. It empathizes on learning and adaptability.

Scripted Testing	Exploratory Testing
Directed from requirements	Directed from requirements and exploring during testing
Determination of test cases well in advance	Determination of test cases during testing
Confirmation of testing with the requirements	Investigation of system or application
Emphasizes prediction and decision making	Emphasizes adaptability and learning
Involves confirmed testing	Involves Investigation
Is about Controlling tests	Is about Improvement of test design
Like making a speech - you read from a draft	Like making a conversation - it's spontaneous
The script is in control	The tester's mind is in control



# References

- [https://www.unf.edu › ~broggio › cen6016 › classnotes › 6a.Agile Software.](https://www.unf.edu/~broggio/cen6016/classnotes/6a.Agile%20Software.)
- [www.uml.org.cn › softwareprocess](http://www.uml.org.cn/softwareprocess)
- [https://www.unf.edu › ~broggio › cen6016 › classnotes › Lecture 12 - Agi...](https://www.unf.edu/~broggio/cen6016/classnotes/Lecture%2012%20-%20Agile%20Software.)
- Data taken from  
[Ganesh.Sambasivam@isoftplc.com](mailto:Ganesh.Sambasivam@isoftplc.com)
- SDLC 3.0 book; Google; Scattered Notes, Course Textbook

# References

- [1]. Abrahamsson P, Salo O and Ronkainen J. Agile software development methods (Review and analysis).
  - [2]. Scott W Ambler. Agile model driven development.
  - [3]. Cohen D, Lindvall M, Costa P. Agile software development.
  - [4]. [http://en.wikipedia.org/wiki/Agile\\_Modeling](http://en.wikipedia.org/wiki/Agile_Modeling).
  - [5]. [http://en.wikipedia.org/wiki/Extreme\\_Programming](http://en.wikipedia.org/wiki/Extreme_Programming).
  - [6]. [http://en.wikipedia.org/wiki/Agile\\_Unified\\_process](http://en.wikipedia.org/wiki/Agile_Unified_process).
  - [7]. <http://en.wikipedia.org/wiki/Scrum> 28development29.
- **Agile Development**-Dr. M. Zhu  
*Slide Set to accompany, Software Engineering: A Practitioner's Approach, 7/e, by Roger S. Pressman, Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman-*

# Thank You