Software Process:

- It is:
    - A structured set of activities, used by a team to develop software systems.
    - The standards, practices, and conventions of a team.
    - A description of how a team performs its work.
- Some synonyms of software process are:
    - Software Development Process
    - Software Development Methodology
    - Software Development Life Cycle
- In a nutshell, it is a structured description of how a software development team goes through.
  I.e.
  Deciding what to build.
  Building it.
  Deciding if they like what they built.
- A **software process** is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.
- Over time, people develop new software process models.
- A **software process model** represents the order in which the activities of software development will be undertaken. It describes the sequence in which the phases of the software lifecycle will be performed.
  A model is a template/description of a process.
  Different processes can implement the same model.
  Think of “Process vs. Model” like “Class vs. Interface”
- Examples of software process models are:
    - Waterfall Model
    - Prototyping Model
    - Spiral Model

Software Process Model vs. Software Process:

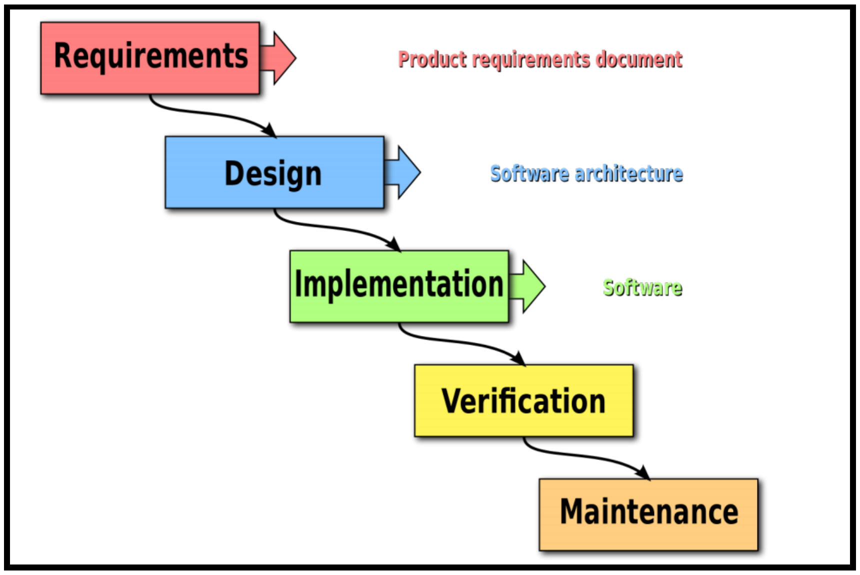
- Software process is a coherent set of activities for specifying, designing, implementing and testing software systems.
- A software process model is an abstract representation of a process that presents a description of a process from some particular perspective.
- There are many different software processes but all involve:
    - **Specification:** Defining what the system should do.
    - **Design and Implementation:** Defining the organization of the system and implementing the system.
    - **Validation:** Checking that it does what the customer wants.
    - **Evolution:** Changing the system in response to changing customer needs.

Waterfall Method:

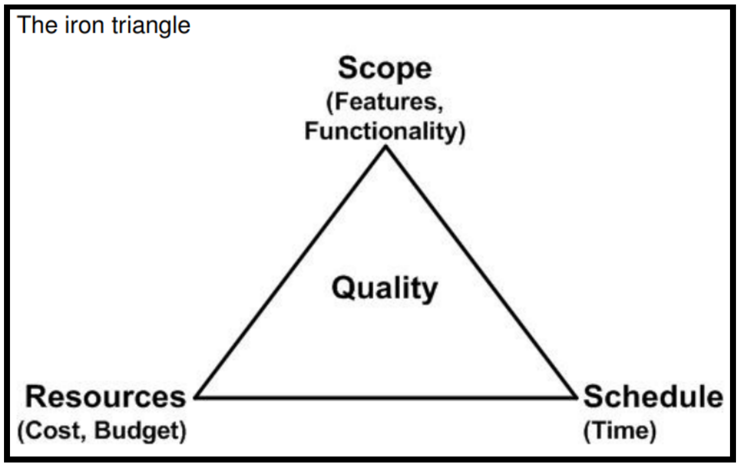
- The **waterfall model** is a sequential (non-iterative) design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of:
    1. Requirements analysis
    2. Design
    3. Implementation
    4. Testing (verification)

5. Maintenance

- I.e.



- The result of each phase is one or more documents that should be approved and the next phase shouldn't be started until the previous phase has completely been finished. I.e. Each phase is carried out completely, for all requirements, before proceeding to the next. Furthermore, this process is strictly sequential. No backing up or repeating phases.
- The waterfall model should only be applied when requirements are well understood and unlikely to change radically during development as this model has a relatively rigid structure which makes it relatively hard to accommodate change when the process is underway.
- Pros:
    - Time spent early in the software production cycle can reduce costs at later stages.
    - Suitable for highly structured organizations.
    - It places emphasis on the documentation, contributing to corporate memory.
    - It provides a structured approach; the model itself progresses linearly through discrete, easily understandable and explainable phases and thus is easy to understand.
    - It also provides easily identifiable milestones in the development process.
    - It is well suited to projects where requirements and scope are fixed, the product itself is firm and stable, and the technology is clearly understood.
    - Simple, easy to understand and follow.
    - Highly structured, therefore good for beginners.
    - After specification is complete, low customer involvement is required.

- Cons:
    -
      

      Software development projects often fail because the organization sets unrealistic goals for the **iron triangle** of software development:
        - **Scope** (what must be built)
        - **Schedule** (when it must be built by)
        - **Resources** (how much it must cost)

      The fundamental problem is that each major group of stakeholders has a different, and often conflicting, set of priorities. IT professionals lean towards building high-quality systems, financial people seem more interested in the overall cost, senior management in the schedule, and end users in the scope. The problem is that when each issue has its own protagonist(s) it becomes difficult to negotiate a reasonable approach to a software project. When nobody budges from their position, or is forced to budge, the project team is positioned for failure.

      By breaking the iron triangle, you often:
        - Cancel the project
        - Deliver late and/or over budget
        - Deliver poor quality software
        - Under deliver

      The iron triangle refers to the concept that of the three critical factors, scope, cost, and time, at least one must vary otherwise the quality of the work suffers. The solution is to think of it as an elastic triangle instead of an iron triangle and vary the cost, schedule, and/or scope as required.
    - Inflexible - can't adapt to changes in requirements.

Users vs Software Engineers:

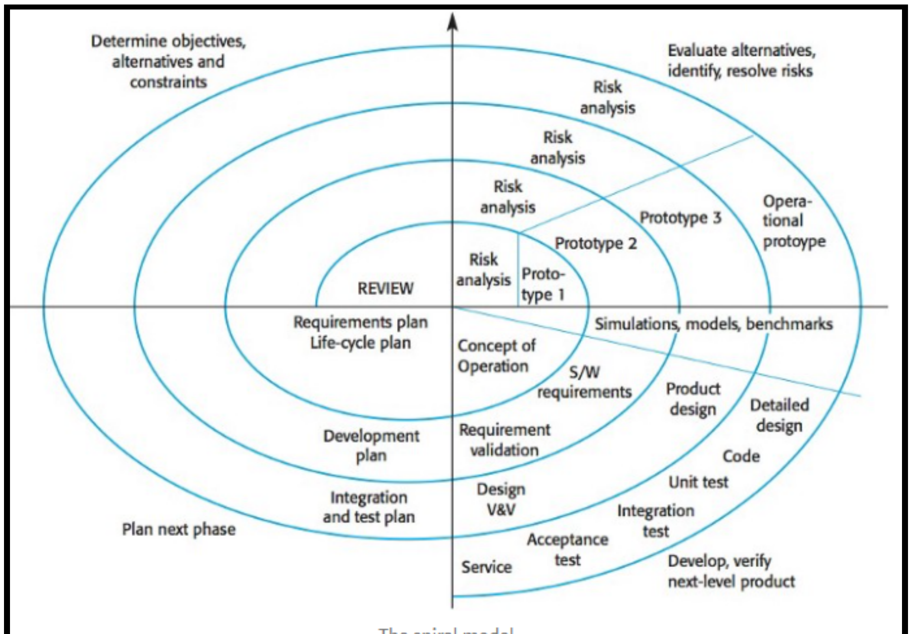
- They speak different languages.
- E.g.
  Software engineers speak:
    - Math
    - Code
    - Algorithms

Users speak:
- Cost
- Consumer value
- Steering mechanism

- **Cognitive friction** is the resistance encountered by a human intellect when it engages with a complex system of rules that change as the problem permutes.
- Software is loaded with cognitive friction because it puts the device in metastates. Users often face problems using software. Sometimes they don't know how to find certain information; other times, software doesn't work as expected. In both cases, cognitive friction is at play.

Spiral Model:

- The spiral model is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model.
  I.e. The spiral model is a risk-driven model where the process is represented as a spiral rather than a sequence of activities. Furthermore, it was designed to include the best features from the waterfall and prototyping models, and introduces a new component: risk-assessment.
- Each loop of the spiral is a phase of the software development process.



The spiral model

- Each loop in the spiral is split into four sectors:
  1. **Objective setting**: The objectives and risks for that phase of the project are defined.
  2. **Risk assessment and reduction:** For each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. For example, if there's a risk that the requirements are inappropriate, a prototype may be developed.
  3. **Development and validation:** After risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface then we must prototype the user interface. If the risk is in the development process itself then use the waterfall model.

4. **Planning:** The project is reviewed and a decision is made whether to continue with a further loop or not.

- Pros:
    - Manages uncertainty inherent in exploratory projects.
- Cons:
    - Difficult to establish stable documents; things keep getting modified during each iteration.
- The spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

Prototype Model:

- A prototype is a version of a system or part of the system that is developed quickly to check the customer's requirements or feasibility of some design decisions.
- A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.
- In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.
- In prototyping, the project is done in cycles.
- In each cycle, the sequence of phases is:
    1. Gather basic requirements.
    2. Implement prototype (e.g. UI with all the functionality mocked/faked).
    3. Collect feedback from users.
    4. Adjust
- Pros:
    - At the end of each cycle, the team can assess their work, adjust to new requirements, and improve its work.
    - More interaction with users helps us ensure that we are building the right product.
- Cons:
    - Building a prototype is not the same as building a product. This model ignores a big chunk of the system that is not trivial to implement.
    - You do not always have patient/forgiving users who will be willing to try your prototypes.

Alternative Methodologies:

- As companies began to realize that the waterfall method was failing them (large projects were failing completely or going way over budget) alternatives were sought.
- A gradual trend was in methods that used an incremental development of product using iterations (almost like smaller waterfalls).
- Iterations could be only a few weeks, but still included the full cycle of analysis, design, coding, etc.
- These various lightweight development methods were later referred to as **agile methodologies**.
- As our models evolve, they encourage software development teams to:
    - Be more flexible and adaptive to changing requirements.
    - Collect feedback from users more frequently.
    - Release code more frequently.
- And then came the term Agile.

- Agile is neither a process nor a model but is a term that describes a process, model, or a team. Essentially, it means “Flexible and adaptive process/team, suitable for projects with constantly changing requirements".

Agile Manifesto:

- We 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**.
- There is value to the items on the right, but the left is valued more.

Agile:

- Agility is flexibility. It is a state of dynamic, adapted to the specific circumstances.
- Agile refers to a number of different frameworks that share these values.
  I.e. Agile is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto that is a way of thinking that enables teams and businesses to innovate, quickly respond to changing demand, while mitigating risk.
- Examples of agile frameworks are:
    - Test Driven Development (TDD)
    - Extreme Programming (XP)
    - Scrum
    - Lean Software

Test Driven Development (TDD):

- A concept that started in the late 90s.
- Used (to some extent) by many Agile teams.
- The idea is to write the tests, before you write the code.
- The tests are the requirements that drive the development.
- A software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and make code simple and bug-free.
- TDD ensures that your system actually meets requirements defined for it. It helps to build your confidence about your system.
- In TDD more focus is on production code that verifies whether testing will work properly. In traditional testing, more focus is on test case design. Whether the test will show the proper/improper execution of the application in order to fulfill requirements.
- In TDD, you achieve 100% coverage test. Every single line of code is tested, unlike traditional testing.
- Traditionally, TDD means:
    - Write a failing test.
    - Write the (least amount of) code to pass the test.
    - Repeat.
    - Every now and then refactor / cleanup code.
- However, in practice, each team decides when and where it makes sense for tests to drive development.
- Most (good) teams borrow some of the concepts of TDD, such as the fact that tests are used as specification/documentation and the fact that we should automate tests in fragile/crucial areas of your system.
- TDD makes sense for agile teams:

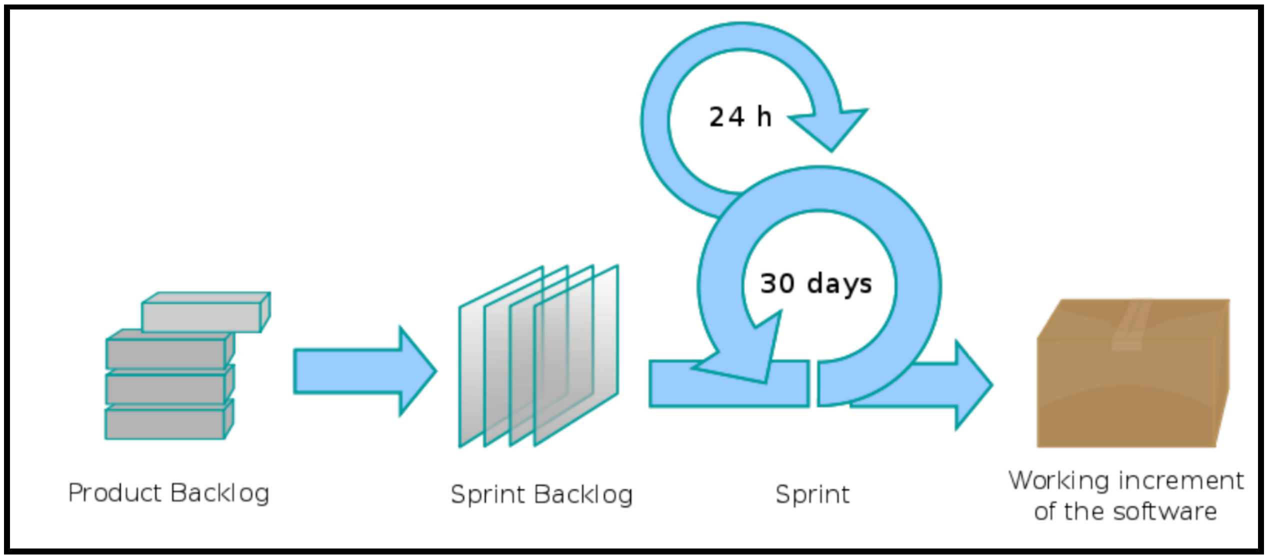
  - Agile is about “moving fast".
  - If your code is easy to test, it is usually also easy to build/deploy automatically.
  - Being able to automate your tasks helps your team move fast.
- TDD vs Unit Testing:
  - TDD: Cycles of writing a failing test, then writing code to pass it.
  - Unit Testing: The concept of testing atomic units individually.
- Software development teams can adopt TDD with different types of testings such as:
  - **Unit Test**
  - **Integration:** Test that all the different units in the system play nicely together.
  - **Acceptance:** Specify customer’s requirement.
  - **Regression:** Verify we didn’t break anything that was working before. Automated unit tests can be used as regression tests.
- Advantages of TDD:
  - **Early bug notification:**
    - Using TDD, over time, a suite of automated tests is built up that you and any other developer can rerun at will.
  - **Better designed, cleaner and more extensible code:**
    - TDD helps developers understand how the code will be used and how it interacts with other modules.
    - It results in better design decisions and more maintainable code.
    - TDD allows writing smaller code having single responsibility rather than monolithic procedures with multiple responsibilities. This makes the code simpler to understand.
    - TDD also forces you to write only production code to pass tests based on user requirements.
  - **Confidence to refactor:**
    - If you refactor code, the code might break. By having a set of automated tests, you can fix those bugs before release.
    - Using TDD should result in faster, more extensible code with fewer bugs that can be updated with minimal risks.
  - **Good for teamwork:**
    - In the absence of any team member, other team members can easily pick up and work on the code. It also aids knowledge sharing, thereby making the team more effective overall.
  - **Good for developers:**
    - Though developers have to spend more time writing TDD test cases, it takes a lot less time for debugging and developing new features. You will write cleaner, less complicated code.

Extreme Programming (XP):

- XP is a model that was getting a lot of hype in the late 90s.
- XP is an Agile model, consisting of many rules/practices, one of which is TDD.
- XP is a very detailed model, but in practice, most teams adopt a subset of its rules.
- Some highlights of XP:
  - Iterative incremental model.
  - Better teamwork.
  - Customer’s decisions drive the project.
  - Dev team works directly with a domain expert.
  - Accept changing requirements, even near the deadline.

- Focus on delivering working software instead of documentation.
- A key assumption of XP is that the cost of changing a program can be held mostly constant over time. This can be achieved with:
    - Emphasis on continuous feedback from the customer
    - Short iterations
    - Design and redesign
    - Coding and testing frequently
    - Eliminating defects early, thus reducing costs
    - Keeping the customer involved throughout the development
    - Delivering working product to the customer
- Extreme Programming involves:
    - Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminate defects early, thus reducing the costs.
    - Starting with a simple design just enough to code the features at hand and redesigning when required.
    - Programming in pairs, called **pair programming**, with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
    - Integrating and testing the whole system several times a day.
    - Putting a minimal working system into the production quickly and upgrading it whenever required.
    - Keeping the customer involved all the time and obtaining constant feedback.

Scrum:



- **Scrum** is a flexible, holistic product development strategy where a development team works as a unit to reach a common goal. Scrum is mainly about the management of software development projects.
- **Sprint:** The actual time period when the scrum team works together to finish an increment. Two weeks is a pretty typical length for a sprint, though some teams find a week to be easier to scope or a month to be easier to deliver a valuable increment. During this period, the scope can be re-negotiated between the product owner and the

development team if necessary. This forms the crux of the empirical nature of scrum. Key features of sprints:

- It is a basic unit of development in a scrum.
- It is of fixed length, typically from one week to a month.
- Each sprint begins with a **planning meeting** to determine the tasks for the sprint and estimates are made.
- During each sprint a potentially deliverable product is produced.
- Features are pulled from a **product backlog**, a prioritized set of high level work requirements.

- **Sprint planning:** The work to be performed during the current sprint is planned during this meeting by the entire development team. This meeting is led by the **scrum master** and is where the team decides on the sprint goal. Specific **user stories** are then added to the sprint from the product backlog. These stories always align with the goal and are also agreed upon by the scrum team to be feasible to implement during the sprint. At the end of the planning meeting, every scrum member needs to be clear on what can be delivered in the sprint and how the increment can be delivered.
- **User Story:** An informal, general explanation of a software feature written from the perspective of the end user or customer. The purpose of a user story is to articulate how a piece of work will deliver a particular value back to the customer. User stories are a few sentences in simple language that outline the desired outcome. They don't go into detail. Requirements are added later, once agreed upon by the team.
- **Product Backlog:** The master list of work that needs to get done maintained by the product owner or product manager. This is a dynamic list of features, requirements, enhancements, and fixes that acts as the input for the sprint backlog. It is, essentially, the team's "To Do" list. The product backlog is constantly revisited, re-prioritized and maintained by the Product Owner because, as we learn more or as the market changes, items may no longer be relevant or problems may get solved in other ways.
- **Sprint Backlog:** The list of items, user stories, or bug fixes, selected by the development team for implementation in the current sprint cycle. Before each sprint, in the sprint planning meeting the team chooses which items it will work on for the sprint from the product backlog. A sprint backlog may be flexible and can evolve during a sprint.
- **Increment/Sprint Goal:** The usable end-product from a sprint.
- **Daily Scrum/Daily Standup:** A short meeting that happens at the same place and time each day. At each meeting, the team reviews work that was completed the previous day and plans what work will be done in the next 24 hours. This is the time for team members to speak up about any problems that might prevent project completion. Some features of the daily scrum:
    - No more than 15 minutes.
    - Meetings must start on-time, and happen at the same location.
    - Each member answers the following:
        - What have you done since yesterday?
        - What are you planning on doing today?
        - Are there any impediments or stumbling blocks?
    - A scrum master will handle resolving any impediments outside of this meeting
- **Scrum Master:** The person on the team who is responsible for managing the process, and only the process. They are not involved in the decision-making, but act as a lodestar to guide the team through the scrum process with their experience and expertise. The

scrum master is the team role responsible for ensuring the team follows the processes and practices that the team agreed they would use.

- In traditional agile development software is brought to release level every few months. **Releases**, which are sets of sprints, are used to produce shippable versions of software products.
- A key feature of scrum is that during a project a customer may change their minds about what they want/need. We need to accept that the problem cannot be fully understood or defined and instead allow teams to deliver quickly and respond to changes in a timely manner.

Extreme Programming vs Scrum:

Below are some differences between extreme programming and scrum:

Issue	**XP**	**Scrum**
Iteration Length	Typically one or two weeks long.	Typically from one week to one month long.
Whether requirements are allowed to be modified in an iteration	Much more amenable to change within their iterations. As long as the team hasn't started work on a particular feature, a new feature of equivalent size can be swapped into the XP team's iteration in exchange for the un-started feature.	Do not allow changes into their sprints. Once the sprint planning meeting is completed and a commitment made to deliver a set of product backlog items, that set of items remains unchanged through the end of the sprint.
Whether User Story is implemented strictly according to priority in iterations	Work in a strict priority order. Features to be developed are prioritized by the customer and the team is required to work on them in that order.	Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items. A Scrum team will very likely choose to work on the second most important.
Whether to adopt strict engineering methods to ensure progress or quality in the process of software implementation	Yes, for example, TDD.	Doesn't prescribe any engineering practices.

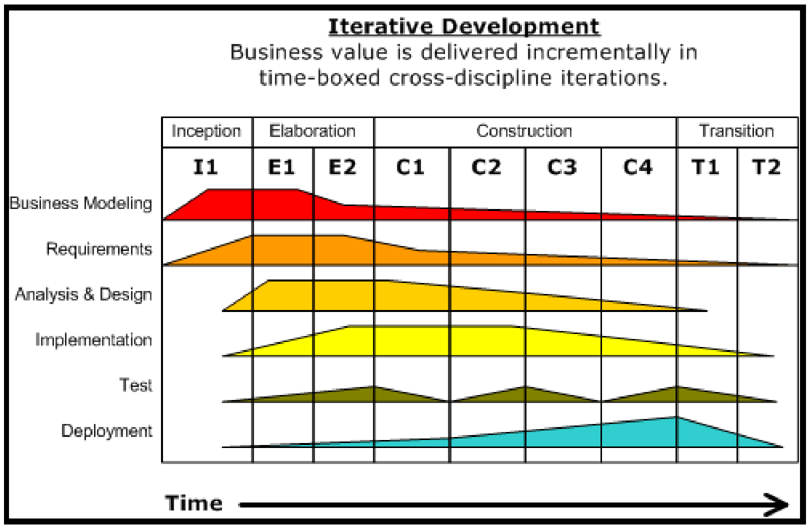
Lean Software Development:

- Lean software development is a concept that emphasizes optimizing efficiency and minimizing waste in the development. It is an agile framework based on optimizing development time and resources, eliminating waste, and ultimately delivering only what the product needs.

- Lean software development is an agile framework sharing the following values:
    - Eliminate waste
    - Amplify learning
    - Decide as late as possible
    - Deliver as fast as possible
    - Empower the team
    - Build flexibility in
    - See the whole

Agile Unified Process (AUP):

- AUP is an iterative-incremental process consisting of seven sub-processes or workflows:
    1. **Model:** Understand the business, the problem domain, and try to identify a viable solution.
    2. **Implementation:** Transform your model(s) into executable code and perform unit testing.
    3. **Test:** Perform an objective evaluation to ensure quality.
    4. **Deployment:** Plan and execute the delivery of the system.
    5. **Configuration Management:** Manage access to your project artifacts.
    6. **Project Management:** Direct the activities that take place on the project.
    7. **Environment:** Support the rest of the effort by ensuring that the proper resources are available for the team as needed.
- The AUP workflows go through four phases:
    1. **Inception:** The goal is to identify the initial scope of the project, a potential architecture for your system, and to obtain initial project funding and stakeholder acceptance.
    2. **Elaboration:** The goal is to prove the architecture of the system.
    3. **Construction:** The goal is to build working software on a regular, incremental basis which meets the highest-priority needs of your project stakeholders.
    4. **Transition:** The goal is to validate and deploy your system into your production environment.



Why Use Agile:

- **Demand for higher quality with lower cost.**
- **Post-mortems of software projects lead to a lot of knowledge gain about what went right/wrong during the development phase.**
    - With the waterfall model, we do not have a clear way to predict the future, so these lessons are always in hindsight.
    - Smaller, iterative schedules mean problems can be identified/addressed much earlier in the cycle.
- **Agile eliminates waste.**
    - Making changes at the end of a production cycle is costly. With agile we are more likely to detect these changes early enough to reduce the costs.
    - Iterative design means we build a product in small steps.
        - Incrementally add features.
        - Software is in working condition at least every few weeks.
        - Allows people to test earlier in the development cycle.
        - Software improvements happen much earlier and can be fine-tuned rather then trying to modify the design at the end of a waterfall cycle.

Agile Team Management:

- From the bottom up.
- Teams are empowered to manage the smallest level of details, while leaving the higher levels to upper management.
- Teams, upon seeing the small amount of ownership they get from solving smaller problems, take on responsibility for larger problems.
    - Head off issues before they become major problems.
    - Individuals solve problems with their colleagues.

Agile Project Structure:

- An agile project consists of a series of iterations of development.
- Each interval usually lasts only two to four weeks.
- Developers implement features, called user stories, during each iteration that add value to the project.
- Each iteration contains a full development cycle:
    - Concept
    - Design
    - Coding
    - Testing
    - Deployment
- The project is reviewed at the end of each iteration.
- Results are used to direct future iterations.
- Every three to six iterations the project is built up to a release state, meaning that most major goals are accomplished.