ECSE 321 Introduction to Software Engineering

1. Specification
2. Development
3. Evaluation
4. Evolution

The Two Types of Software Engineering

- **Plan Driven Development (PDD)**: All activities are planned in advance, waterfall approach
- **Agile**: Planning is incremental, more flexible

Plan Driven Development

An example of PDD is the **waterfall model**. This is when work is done in linear steps, that must be finished in sequence. Consider the following steps as an example (as a waterfall):

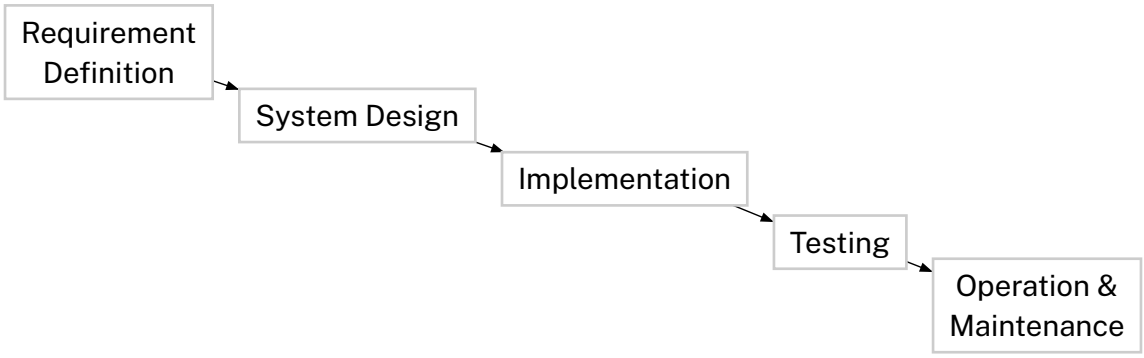


Figure 1: A waterfall driven approach.

There are pros and cons to this system. Generally, this system is good for projects who's requirements are defined upfront such as critical systems. However, if there are changes to the requirements after the respective phase has passed, it makes the system unable to accommodate changes.

Incremental Development Model

An incremental approach starts with an initial implementation with limited features, gathers feedback from stakeholders, and builds a new version with further refinements.

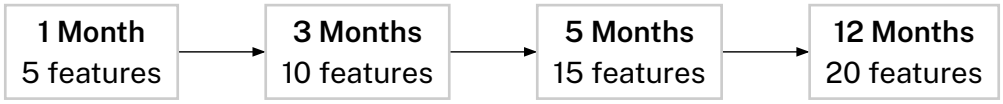


Figure 2: An incremental approach to software engineering

Once more, there are advantages and disadvantages to this approach. You get early development and delivery paired with flexibility. That comes at the cost of small regular

changes, which can lead to no messy code. There is the potential to have no long term vision. This can lead to degradation of the code over time.

The solution to the cons of incremental development is generally more refactoring/ refinement, which stops development. This is hard to manage with larger teams. **Incremental development is considered to be a hybrid between PDD and Agile.**

Agile Development

There are two components of agile. Agile and scrum are two different things that go hand in hand. Agile was designed for operating in a rapidly changing environment, for teams that need to deliver software quickly. All agile processes have some similar characteristics:

1. Specification, design, implementation, testing are intended. Design documentation is minimized and there is no detailed system specification.
2. The system is developed incrementally. Stakeholders give feedback frequently.
3. The development process is heavily supported by tools. Continuous integration and delivery are some of the most common tools.

The Agile Manifesto

The Agile Manifesto is a short document published in 2001 that guides all further agile practices. It has the following core principals:

1. Individuals and interactions are prioritized over processes and tools.
    - Build projects around motivated people.
    - Prioritized face-to-face communication.
2. Working software is better than comprehensive documentation.
    - Working software is the primary measure of progress.
    - Customer satisfaction is important.
3. Customer collaboration is prioritized over contract negotiation.
    - Work with customers for frequent feedback, with daily interactions.
    - Frequent delivery, but make sure it's happening at a sustainable pace.
4. Responding to change is prioritized over following a plan.
    - Changing requirements are welcome.
    - The team will often reflect on how they can be more effective.

Scrum Roles

Scrum is a unique terminology to describe people, artifacts, and time units of a project.

There are many roles within scrum on a software development team.

- Product Owner
    - Is a customer or project manager.
    - Decides features to implement.

    - Updates the product backlog.
- Scrum Master
    - Is in charge of enforcing the process.
    - Can be accountable for team effectiveness.
- Developers
    - Create and implement the spring backlog.

Backlog & Sprints
A backlog is an organized list of features to implement, any bugs to fix, and improvements/ refactorings of existing code. This list is also ordered by priority.

The backlog is comprised of tickets or issues that represent individual pieces of work. Then work is organized into a **sprint** of generally 1-4 weeks.

Sprints contain their own backlog of tasks, which is a way to focus work into smaller chunks of 1-4 weeks. The product owner is the one that decides what goes into each sprint's backlog.

User Stories
User stories are the majority of backlog items. They are essentially a requirement written in the customer's language. They are also written from the user's point of view.

> As a course instructor, I would like to add content to a course page so that students can access it on demand.

The example above is comprised of a role (course instructor), a feature (add content to the course page), and a rationale (so that students can access it on demand).

Epics
An epic is a set of related user stories that may span multiple sprints. They are for long term objectives of software projects.

Sprint Structure
Sprints generally have the following structure:

1. **Sprint Planning:** task selection for the sprint (product backlog → sprint backlog). Long, scheduled meeting.
2. **Daily Scrum Meeting:** where all members talk about what they worked on, what they are going to work on, and anything blocking them. Often called a "standup" and lasts about 15 minutes.
3. **Sprint Review:** inspects the sprint outcome, and adjusts the backlog based on what still needs to be done or what extra tasks were finished. Commonly aligns with product releases or other milestones. Stakeholders may be invited for feedback.
4. **Sprint Retrospective:** review problems affecting progress.

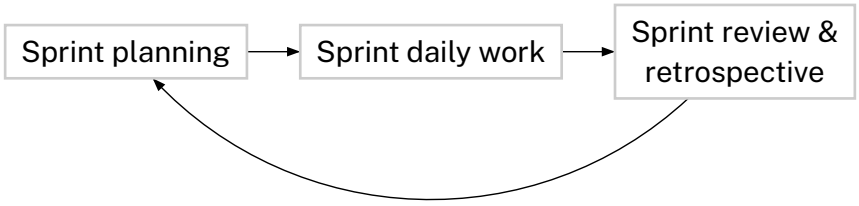


Figure 3: The repeating sprint cycle.

How To Plan for a Sprint

During sprint planning, every story selected is given some number of **story points** (unit of effort). The effort for each item is voted upon by the team. Depending on the size of the team, you can have an idea of how many story points can be completed in a sprint. This is considered the team's velocity and can be adjusted in the sprint retrospective.

There are some reasons why a team might not be able to complete everything assigned to them in a sprint (unable to achieve their velocity). It could be because of a poor estimation or a change in the team members, etc.

Requireents Engineering

Requirements are a capability/condition that a user needs to solve a problem or achieve an objective. It can also be a capability/condition that must be met by a system to satisfy either a contract, a standard, or a specification. The split between these requirements is either a user requirement or a system requirement.