Program Design Methodology

Outline

- API
- Client and Services
- Characteristics of a Good Program
- Software Development Life Cycle and Methodologies

API

- Application Programming Interface
- Toolset for other developers (it could be you) to use.
- Clear distinction between ***clients and services.***

Clients – Services: Definitions

- **Clients:**
  - One requests and uses services from another developer.
- **Services**:
  - One provides the implementation for a particular piece of software required by the clients.
  - This is primarily done through outsourcing.

Software Development Process

- **Specifications**
  - Given by clients.
  - Defines what the program should do. i.e., what is the problem that we are trying to solve.
- **Design and Implementation**
  - Done by both clients and services.
    - Clients design the interface
    - Services writes the implementation.
- **Verification and validation – Testing**
  - Done by the services.
  - Clients will run their own set of tests to make sure that it passes the requirements and that the problem is indeed solved.

Characteristics of a **Good** Program

- **Correct**
  - Does what the user asks it to do
  - It really doesn't matter how fast or efficient a program is if it doesn't do what the user needs.
  - Programs must not solve the wrong problem.
- **Robust**
  - Programs need to handle all input, both the expected and the unexpected.
  - This is usually due to bad programming logic or bad data.
  - If the user provides bad data, it's still the programmer's fault if the program crashes.
  - Except for physical problems with the hardware, program crashes are almost always the fault of the programmer.

Characteristics of a **Good** Program

- **Portable**
  - Many times this is referred to as *cross-platform*, meaning that the program can run on different computers/devices.
  - This generally needs to be factored into the implementation at the outset.
  - Trying to "port" programs that were never intended to run on another system is very difficult.
  - Many times the developers will simply "start over" rather than to port code.
- **Maintainable**
  - Some programs can last for many years.
  - Because "any software worth using is worth enhancing", programs are in constant development.
  - Most code is in maintenance mode.
  - This means that 5 minutes after you write a brand new function, it is now just code that needs to be maintained for the rest of its life.

Characteristics of a **Good** Program

- **Readable/User friendliness**
  - Many people feel this is the most important aspect of a good program.
  - If a program is very readable, it's likely that the other characteristics will be much easier to attain.
  - C (and thus, C++) has been termed a "write-only" language because it allows programmers to write completely unintelligible code. (At least to the poor maintenance programmer who comes after.)

Characteristics of a **Good** Program

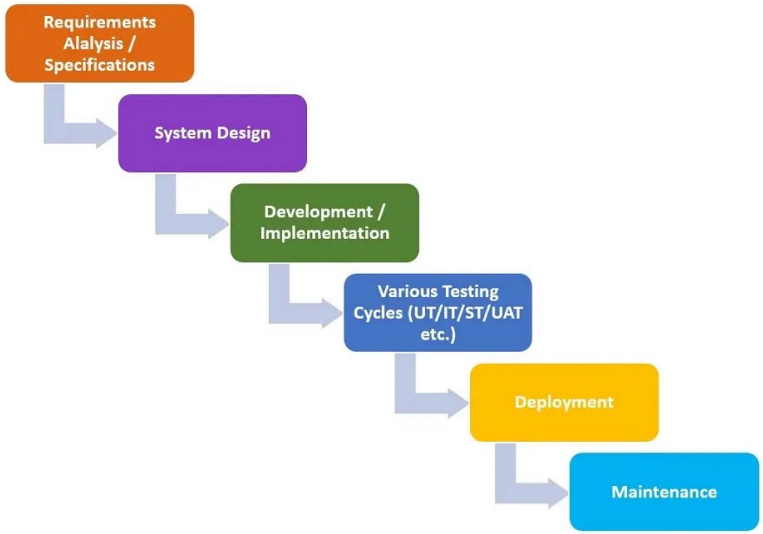
- **Efficient**
  - This relates to resources (such as memory or CPU time.)
  - The goal is for programs to use the least amount of memory, disk storage, CPU cycles, etc. to get the job done.
  - Unfortunately, all too often the other characteristics of a good program are traded for the promise of a very fast and efficient program.

Software Development Life Cycle (SDLC)

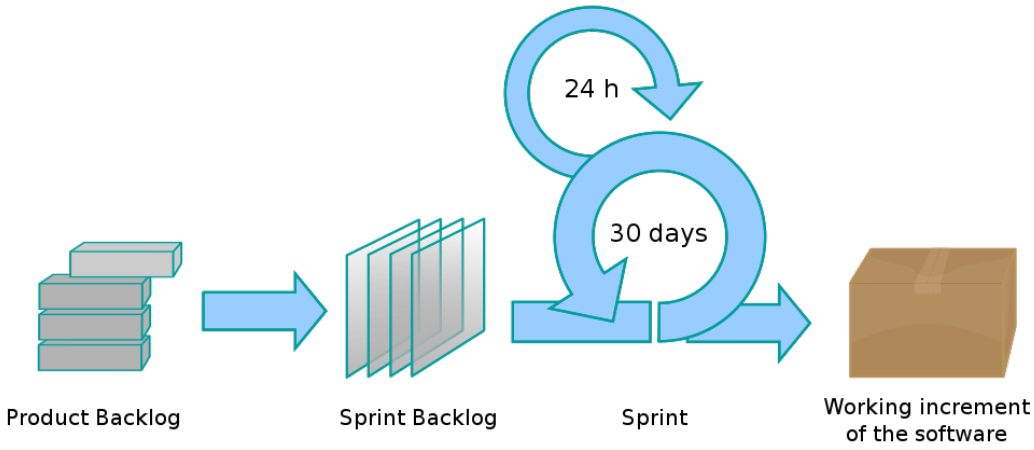
SDLC describes the key stages of software development.

1. User Requirement Analysis
2. Functional Specifications
3. Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance

Methodologies Examples



The Waterfall Method (Sequential)



The Agile Method (Iterative)

Methodology Examples

- The **waterfall method** was the predominant method in the early days of software development.
- The **agile method** slowly replaced the waterfall method because it was more suited with the modern client expectations, e.g., quick incremental tangible results, flexible development.

Other Methodologies

- **Extreme Programming (XP)**
  - A popular Agile software development
- **Test Driven Development (TDD)**
  - A process of creating the tests **before** you implement the functionality.
- Most complex projects use combination of those methods depending on the type and scale of the project.

Assignments

- This course adopts learning by doing.
- Assignments are structured based on **TDD**.
- I am the client;
  you are the service
- I provide the interface;
  You provide the implementation.
- C++ is the language of choice.
  - I will provide you with the public interface.
  - You will be in charge of the private interface.