Introduction to Software Engineering

Why Learn Software Engineering ???

- Software engineers play an important role in today's information driven society.

- Software engineering is one of the fastest growing professions with excellent job prospects predicted throughout the coming decade.

- Software engineering brings together various skills and responsibilities. The typical design and development process involves these steps:

1. **Assess user needs** to determine the specifications of the software.
2. **Create flowcharts** and diagrams to draft the software program.
3. **Write the algorithms**, or detailed instructions, that direct the computer hardware.
4. **Code** these instructions in the appropriate programming language.
5. **Test** the program and fix any bugs.

What is software engineering?

- Software engineering is an engineering discipline that is concerned with all aspects of software production.

- That is from the early stages of system specification to maintaining the system after it has gone into use.

Software characteristics

- Software is developed or engineered; it is not manufactured.
- Software does not “wear out” but it does deteriorate.
- Software continues to be custom built, as industry is moving toward component based construction.

Software Engineering -Definition

- **General Definition**

- Software engineering is the **establishment and use of engineering principles** in order to obtain economically feasible software that is **reliable and works efficiently** on real machines.

- **IEEE Definition**

- Software Engineering:
- The application of a **systematic, disciplined, quantifiable** approach to the **development, operation, and maintenance of software**; that is, the application of engineering to software.

- “An organized, analytical approach to the design, development, use, and maintenance of software.”

WHY IS SOFTWARE ENGINEERING IMPORTANT?

- Producing a software application is relatively simple in concept: Take an idea and turn it into a useful program.

- Unfortunately for projects of any real scope, there are countless ways that a simple concept can go wrong.
- Programmers may not understand what users want or need so they build the wrong application.

- The program might be full of bugs that it’s frustrating to use, impossible to fix, and can’t be enhanced over time.

- **Software engineering includes techniques for avoiding the many pitfalls.**

- **It ensures the final application is effective, usable, and maintainable.**

- **It helps to meet milestones on schedule and produce a finished project on time and within budget.**

- Perhaps most important, software engineering gives us the flexibility to make changes to meet unexpected demands without completely affecting our schedule and budget constraints.

- There different steps or phases that we need to take to keep a software engineering project on track.

- These are more or less the same for any large project although there are some important differences.

Important steps or phases in Software Engineering

- They are:

- Requirements Gathering
- Design
  - High-level Design
  - Low-level Design
- Development
- Testing
- Deployment
- Maintenance

Requirements Gathering

- No big project can succeed without a plan. Sometimes a project doesn't follow the plan closely, but every big project must have a plan.

- The plan tells project members what they should be doing, when and how long they should be doing it, and most important what the project's goals are.

- One of the first steps in a software project is **figuring out the requirements.**

- **we need to find out what the customers want and what the customers need.**

- Depending on how well defined the user's needs are, this can be time-consuming.

- Once the customers' wants and needs are clearly specified, then we can turn them into **requirements documents.**

- Those documents tell the customers what they will be getting, and they tell the project members what they will be building.

- Throughout the project, both customers and team members can refer to the requirements to see if the project is heading in the right direction.

Design

- HIGH-LEVEL DESIGN

- The high-level design includes such things as decisions about **what platform to use** (such as desktop, laptop, tablet, or phone), **what data design to use**.

- The high-level design should also include information about the **project architecture** at a relatively high level.

- We break the project into different modules that handle the project's major areas of functionality.

- we should make sure that the high-level design covers every aspect of the requirements.

- It should specify what the pieces (modules) do and how they should interact, but it should include as few details as possible about how the pieces do their jobs.

- LOW-LEVEL DESIGN

- After high-level design breaks the project into pieces, we can assign those pieces to groups within the project so that they can work on low-level designs.

- **The low-level design includes information about how that piece of the project should work.**

- Better interactions between the different pieces of the project that may require changes here and there.

Development

- After we have created the high- and low-level designs, it's time for the programmers to get to work.
- **The programmers continue refining the low-level designs until they know how to implement those designs in code.**
- As the programmers write the code, they test it to make sure it doesn't contain any bugs.

Testing

- Even if a particular piece of code is thoroughly tested and contains no (or few) bugs, there's no guarantee that it will work properly with the other parts of the system.

- One way to address the problems like this, is to **perform different kinds of tests**.

- First developers test their own code. Then testers who didn't write the code test it. After a piece of code seems to work properly, it is integrated into the rest of the project, and the whole thing is tested to see if the new code broke anything.

Deployment

- The software is **delivered to the customer** who evaluates the delivered product and provides **feedback** based on the evaluation.

Maintenance

- As soon as the users start pounding away on our software, they will find bugs.

- Of course, **when the users find bugs, we need to fix them.** Fixing a bug sometimes leads to another bug, so now we get to fix that one as well.

- If our application is successful, users will use it a lot, and they'll be even more likely to find bugs. They also think about of **enhancements, improvements, and new features** that they want added immediately.