# Software Engineering

# Software

- Everything is software. Even if you have a great idea, you need a software for that.
- More and more systems and software controlled.
- It can cost a lot.
  - There are programs which cost more than the hardware we use.
  - Maintenance cost is also important. Maintaining the software for a *long time* can cost a lot.
  - We need to be cost-effective.

# Software Development

- What is a good software?
  - Good software should deliver the required functionality and performance to the user. It should be **maintainable**, **dependable** and **usable**.
- What is the difference between **computer science** and **sw engineering**?
  - CS focuses on theory, SW engineering is concerned about the practical: developing and delivering useful software.
  - Software engineering is not only about programming.

# Software Development

- Engineering is not only about programming. You need know-how.
  - You need to know how the whole **system** works. How are all connected?
  - It is about problem solving.
    - It is about networking, databases, system, scaling, etc.
- In this course, we are going to use all of these knowledge you acquired before to design systems.

# Good software

- This is computer engineering not systems engineering, so our systems will consist of software.
- A good software must have these attributes:
- **Maintainability**
- **Dependability and security**
- **Efficiency**
- **Acceptability**

# Maintainability

- You should create software so that it can be evolved and changed.
- With technology, users will want **more** things. Their **needs** can change.
- Therefore, the software should adapt to it. When we are designing the software & systems, we should keep this in mind.
- This is **crucial**.

# Dependability & security

- The users must be able to depend on the program to run. It should not create physical or economic damage in the event of system failure.
  - We must prevent such things by planning ahead.
- It must be secure.
  - We must use the necessary and state of the art security systems to secure these systems.
  - You cannot say that you are not a **security expert**. You must have a decent knowledge on the matter.

# Efficiency

- Software should work efficiently and should not hoard the system resources.
- Nowadays, since RAM and Hardware is cheaper compared to old times, developers do not care about memory & resources.
    - This is not good!
- We should choose the programming language and do our calls accordingly.

# Acceptability

- Software must be understandable. It should be as simple as possible.
- Education is also important in this part. Although it is not the job of the developer, the developer must think about the users who will use this software.
- Again, just writing code is not enough.

# Usage

- Software has end users. The people who are using that software.
- Students sometimes think that they are going to build a software from top to bottom.
    - In addition, they sometimes think that the **end user** is themselves.
- You should keep in mind that maybe the end user is not a tech savvy person. Maybe you are building a product for the **company**. So, the end users are going to be *marketing* people. The software needs to be suitable for them.

# Software types

- Not all software have GUI (Graphical User Interface).
- Many software do not need or use UI, they just work on command line.
  - If you are not comfortable with command line, now is the time to improve yourself on that.
  - Especially Linux/Unix Shells (Bash, Zsh)

# Software Types

- **Stand-alone applications**
  - These apps work on local computers, such as our own. They include everything necessary and do not need Internet connection.
    - Paint.exe, Notepad.exe are examples.
- **Interactive transaction-based applications**
  - These programs run on other computers and we connect to them by using remote connections. Web applications are examples to it.
    - When we connect to a webpage, we are actually connecting to a collection of programs running on the remote machine.

# Software Types

- **Embedded control systems**
  - These manage hardware devices. You can think of software in a factory setting. In a car manufacturing shop, those robots are run by software.
- **Batch processing systems**
  - These process data in large batches. They process large numbers of individual inputs to create corresponding outputs. For example, when you send money to other banking systems (EFT), all these are handled in the evening as batch.
- **Modeling and simulation**
  - There are systems developed to model physical processes or situations. Simulink can be an example.

# Software types

- **Data collection systems**
  - These systems collect data via sensors and send to other systems for processing. Wireless Sensor Networks are good examples to it.
    - In a big forest, in order to understand if there is a fire happening inside, you can put sensors. These sensors communicate with the Gateway by hopping through every sensor. They don't calculate anything but the gateway does.
- **Systems of systems**

# Fundamentals

- Writing software is hard and time consuming.
  - That is why, we should **reuse** software that has already developed if possible.
- Software specifications and **requirements** are very important.
  - What the software should do? It must be clear.
- This is generally a problem in daily life. Business makes decisions about the product and software and they are not technical.
- Technical people must have good communication skills so that they can tell what is what to business people.

# Ethics

- Software engineering is not just the application of technical skills.
- We must behave in an honest and ethically responsible way.
- Ethical behaviour is more than just obeying the law.
  - It is also following a set of principles that are morally correct.
  - `The Social Dilemma` is a good watch about these topics.
- Usually you sign a confidentiality agreement but we should not even need it. We should respect the privacy of the employees and companies.

# Case Study

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.

# Insulin Pump Control System

- It collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- This calculation is based on the rate of change of blood sugar levels.
- It sends signals to a *micro-pump* to deliver the correct dose of insulin.
- **Safety-critical system** because it can be life-threatening.
    - These systems must be designed and programmed carefully.

# Activity Model

- We are going to see and learn examples of these kind of UML diagrams.