CMP 2101: Software Engineering

Lecture 01: Introduction to Software Engineering

1.1 Introduction

Many nations currently heavily depend on complex computer-based systems. There are a number of systems; electrical, industrial, manufacturing and mechanical, among others that operate using a computer and/or controlling software.

Many companies have taken the bold step and largely invested in computerised systems and expect quality of work and a good return-on-investment.

Although Software is abstract and intangible, Software Engineering as an engineering discipline seeks to attain the **cost effective** development of **high-quality** software systems.

The notion of ***software engineering*** *was* proposed in the late 1960s at a conference held to discuss what was then called the 'software crisis'.

Software development was in crisis because the methods used were not good enough:

- Major projects were sometimes years late and cost much more than originally predicted(over budget),
- Techniques applicable to small systems could not be scaled up to use on larger complex systems. This resulted directly from the introduction of third-generation computer hardware which compared with software systems developed for the second-generation computers, had the following features:
  - model parts of the *real world;*
  - large and complex;
  - abstract;
  - highly dependable;
  - better maintainable: as the real world changes, so too must the software to meet the changing requirements;
  - user friendly, (and thus the *user interface in a software system is an important part.)*
- Software developed was not responsive to the user and/or customer requirements, difficult to use, maintain, and enhance.

Originally defined in terms of productivity, the software crisis evolved to emphasize quality. Some used the term software crisis to refer to their inability to hire enough qualified programmers.

1.2 Software Engineering

Software Engineering is

- the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. [IEEE 1993]
- the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. [1969]
- an engineering discipline concerned with all aspects of software production from the early stages of specification to maintaining the system after it has gone to use.

Software Engineering involves project planning, project management, systematic analysis, design, verification and validation, user training and support and maintenance activities.

The **primary goal** of software engineering is to provide the **quality of software with low cost.**

1.3 Software

Software is

- A collection of **instructions** (computer programs) and **associated documentation** and **configuration data** needed to make these programs operate.
- A collection of computer programs, rules, procedures, associated documentation and data. [IEEE]

Software engineers are concerned with developing software products, i.e., software which can be sold to a customer. There are two fundamental types of software products:

1. ***Generic products:*** These are stand-alone systems that are developed and sold on the open market to any customer who is able to buy them. Examples of this type of product include software for PCs such as databases, word processors, drawing packages and project management tools.

2. ***Custom (or bespoke) products:*** These are systems which are commissioned by a particular customer and the software contractor develops the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process and air traffic control systems.

The difference between the above mentioned software products is that for generic products, the development company controls the specifications while for the custom products, the customer does that.

Software has a wide range of applications e.g. E-commerce, Business, Hospitals, Schools, Defence, communication…where else??

1.4 Why study Software Engineering

1. It provides an opportunity to come up with solutions for real life problems. The knowledge so obtained can be applied to design systems that include both hardware and software and for solving complex problems.
2. It provides an opportunity to design for the future, hence enabling you to be one of the few that change the world!!
3. It gives one an opportunity to create jobs. For example, if one needs to develop software for large complex systems they'll need people to work on different modules.
4. Many jobs.. a multi-faceted opportunities in fact!!
5. You could go on!

1.5 People in Software Engineering History

Right from the 60's when the notion of software engineering was birthed there have been a number of people at the forefront of Software Engineering. Some of these people include:

- 1968: Peter Naur et al coined the term "software engineering" at NATO conference in Garmisch-Partenkirchen and pointed out that software should follow an engineering paradigm, it was the response to the software crisis which provided with too little quality, too late deliver and over budget.
- 1977: D. Teichrow and E. Hershey: paper on prototyping as a tool in the specification of user requirements.
- 1986: G. Booch: introduced OOD(Object-Oriented Design)
- 1986 Bjarne Stroustrup: introduced C++ Programming Language

- 1991: Peter Coad , Edward Yourdon: book on the principles of object-oriented technology
- 1995 James Gosling/Sun Microsystems: introduced Java, a simplified C++ like OOP which expressly designed for use in the distributed environment of the Internet.
- **Grace Hopper (read about...assignment due next lecture!!)**

1.6 Challenges with Large projects

Software projects can be large and complex and this presents with quite a number of challenges. These have to be taken into consideration as one embarks on building large complex software products:

- They can be effort intensive due to the fact that they are large in scope and span and also present with complexity in understanding and developing.
- Large projects mean high cost requirements almost always. High costs which may be due to tools or labour requirements.
- Possibility of long development time and even worse that this time might even stretch longer than initially projected is high.
- There is a high risk of failure in terms of performance, user acceptance, maintainability.
- One has to be ready for users' needs to change quite frequently!!

Therefore, in order to build good systems, there is need for:

1. A well defined development process with clear phases of activities, each of which has an end-product
2. Methods and techniques for conducting the phases of activities and for modelling their products,
3. Tools for generating the products.

1.7 Software Myths

Management myths

- Add more programmers if behind schedule

Customer myths

- A general description of objectives enough to start coding
- Requirements may change as the software is flexible

Practitioner myths

- Task accomplished when the program works
- Quality assessment when the program is running
- Working program the only project deliverable

1.8 Major Software Failures – A History

- **Therac-25 (1985-1987)** : six people overexposed during treatments for cancer
- **Taurus (1993)** : the planned automatic transaction settlement system for London Stock Exchange cancelled after five years of development
- **Ariane 5 (1996)** : rocket exploded soon after its launch due error conversion (16 floating point into 16-bit integer leading to an exception)
- **The Mars Climate Orbiter :** assumed to be lost by NASA officials (1999): different measurement systems (Imperial and metric)

1.9 Software Engineering Vs Computer Science

Computer science is concerned with theory and fundamentals that underlie computers; software engineering is concerned with the practicalities of developing and delivering useful software.

1.10 Software Process

A software process is the set of activities and associated results that produce a software product.
Different types of systems need different development processes. E.g. space systems require different processes from business applications. Therefore the use of the appropriate software process goes a long way in enhancing system quality and usefulness, and reducing development costs.

1.11 The attributes of good software

Other than the services or functions the software provides, software products have a number of other associated attributes that reflect the quality of that software. Although these attributes are not directly concerned with what the software does, they reflect its behaviour while it is executing and the structure and organization of the source program and associated documentation.
The attributes so expected from a software product also depend on its application e.g. a banking system must be secure, an interactive game must be responsive, a particular software's program code must be understandable.
These can be generalized into the set of attributes shown in Table 1, which are the essential characteristics of a well-designed software system.

Product characteristic	**Description**
Maintainability	Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.
Dependability	Software dependability has a range of characteristics, including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, utilization, etc.
Usability	Software must be usable, without undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

1.12 Key Challenges in Software Engineering

Software engineering in the 21st century faces 4 key challenges:

1. Legacy systems - These are old, valuable systems must be maintained and updated
2. Heterogeneity -Systems are distributed and include a mix of hardware and software
3. Delivery - Increasing pressure for faster delivery of software
4. Trust - Developing techniques that demonstrate that software can be trusted by its users

1.13 Professional and Ethical Responsibility

Software engineering involves wider responsibilities than simply the application of technical skills. Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals. Ethical behaviour is more than simply upholding the law.

Issues of Professional Responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients even without a formal confidentiality agreement.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is beyond their competence.
- Intellectual property rights
  - Engineers should be careful to ensure that the intellectual property of employers and clients is protected and know the local laws governing IP.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers.