LAU
Lebanese American University
Chartered in the State of New York
www.lau.edu.lb

SOE
School of Engineering
Department of Electrical & Computer Engineering
www.lau.edu.lb/ece

BYBLOS CAMPUS	BEIRUT CAMPUS	NEW YORK HEADQUARTERS
P.O. Box: 36	P.O. Box: 13-5053 Chouran	211 E. 46th Street
Byblos, Lebanon	Beirut 1102 2801, Lebanon	New York, NY 10017-2935, USA
Tel: +961 9 547 262	Tel: + 961 1 786 456	Tel: +1 212 203 4333
Fax: +961 9 546 262	Fax: +961 1 867 098	Fax: +1 212 784 6597

Software Engineering

COE 416

Chapter 1 - Introduction

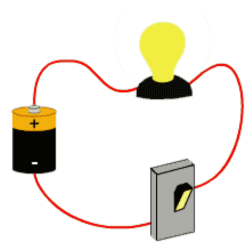
Dany Ishak, Ph.D.
Software Engineering

Ch. 1 - Introduction

Chapter Contents

1. Introduction

- The modern world cannot run without software!
  - Infrastructure and utilities are increasingly controlled by computer-based systems



  - Industrial manufacturing is completely computerized
  - Entertainment: music, cinema, gaming, TV, etc. is becoming software intensive

Not to mention the Web…

1. Introduction

- Software systems are **abstract** and **intangible**
  - Not constrained by the laws of physics and properties of materials

This should inherently simplify Software Engineering ?!

⇨ No limits to the potentials of software

"A world without borders or boundaries,
where anything is possible" – The Matrix (movie, 1999)

- Yet because of the **lack of constraints**:
  - Software systems can become **extremely complex**!

⇨ Difficult to understand, to manipulate, and expensive to change

Which is where SE comes to play

1. Introduction

- There is no unified recipe to solve all SE problems!

  ⇨ **Diversity:** different types of software require different SE approaches

  ⇨ From embedded systems, to world wide information systems

  ⇨ **Increasing demands:** differ with user, application, context, & constraints

  Could induce software failures

  ⇨ Due to lack of adapted SE solutions

- **Couldn't we simply program without using SE?**
  - **Seems easier with small projects**

    ⇨ **Lack of SE complicates big project development**

    – In addition to producing **unreliable solutions**

Ch. 1 - Introduction

Chapter Contents

1. Introduction
2. Basic Concepts
3. Software Engineering Ethics
4. Case Studies

D. Ishak – dany.ishak@lau.edu.lb

2. Basic Concepts

*2.1. What is **Software**?*

- It is a computer program with associated documentation & configuration data

- Lack of proper documentation/configuration data
  - System documentation, user documentation, support websites, etc.

    ⇨ Major difference between **professional** and **amateur** software

- Also, **professional software** are developed for a **particular customer/market**

    ⇨ In contrast with **personal use** programs

    - Such as spreadsheet or formula solving programs for engineers/scientists
      - No need for documentation since the only user is the code developer
        - For instance, simple programs on (MS) Excel or (MathWorks) Matlab

2. Basic Concepts

*2.2. Job of a **Software Engineer***

- The job of **software engineers** is to **develop professional software** products
- We distinguish two kinds of software products:
  - **Generic products:** Developed for the general market
    - For example: software for PCs, mobile applications
    - **Vertical applications**: designed for specific purposes (databases, accounting, etc.)

      ⮡ Software developing organization controls software specs
  - **Customized (bespoke) products:** developed for a specific user
    - Commissioned by a particular customer
      - For a particular business process, or computerized system (cars, aircrafts, etc.)

        ⮡ Customer usually controls software specs, rights, and privileges

2. Basic Concepts

*2.3. **Quality** of Professional software*

- **Functional** attributes: *what the software does!*
- **Non-functional** attributes:
  - **Efficiency:** Time and space performance, latency, throughput, etc.
  - **Maintainability:** software should be easy to update and/or extend
    - Adapting to changing needs
  - **Acceptability:** by the type of user for whom it was designed
    - Understandable , usable (documentation), and compatible (configuration)
  - **Dependability and Security**: Reliability, security, and safety
    - Preventing physical/economic damage, while protecting user information/interests

Attributes of a good software depend on its application

2. Basic Concepts

*2.4. What is **Software Engineering**?*

- It is a discipline concerned with all aspects of **software production**
  - Mainly concerns **medium** to **large-scale** software production
  - **Engineering:** Making things work, applying (creating new) theories, methods, and tools where/when appropriate.
    - Given specific organizational/financial constraints

      An engineer cannot be a perfectionist!
  - **Software production:** Technical aspects of software development, as well as project management and theory/method/tool development
    - Adopting a systematic and organized approach

2. Basic Concepts

*2.5. Software **Process***

- The systematic approach used in SE is called: **software process**
  - Sequence of activities leading to software production
- Software processes consists of four main activities:
  - **Software specification**: defining the functional/non-functional constraints of the software to be produced
    - Identified by the engineer and the customer
  - **Software development**: design and programming
  - **Software validation:** checking and testing to verify functional/non-functional requirements
  - **Software evolution:** maintenance and extensibility to satisfy customer needs

2. Basic Concepts

2.5. Software ***Process***

- Different systems have **different needs**
  - ⇨ Require **different software processes**, for instance:
    - A automobile's computerized board should be completely designed before development
      - Very expensive to recall a car and update its computer software
    - E-commerce or Web applications can be easily re-designed during/after development
      - Developed through a series of prototypes

It is up to the software engineer to identify the process most adapted to the software project

⇨ Covered in more detail in *Chapter 2*

2. Basic Concepts

2.6. Different Kinds of Software

- **Stand-alone applications: Running on a local computer (office apps, CAD, etc.)**
  - Functionality among key requirements
- **Interactive transaction-based: Executing on a remote computer (Web applications, Web services, e-commerce, Cloud utilities, etc.)**
  - Availability among key requirements
- **Embedded control systems: Control hardware devices (mobile phones, car/aircraft computer systems, etc.)**
  - Responsiveness, power management, and safety are among key requirements
- **Batch processing systems: Business systems processing data in batches (periodic billing, monitoring systems, etc.)**
  - Reliability and efficiency are among key requirements

2. Basic Concepts

2.6. Different Kinds of Software

- **Entertainment systems: for personal use and entertainment (games)**
  - User-system Interaction is among key requirements
- **Modelling and simulation: developed by scientists/engineers to model physical complex situations with different parameters**
  - High-performance is among key requirements
- **Data collection: Collecting data from environment using sensors**
  - Coping with limited bandwidth and processing capabilities
- **Systems of Systems: Composed of other software systems**
  - Component integration and coherence is among key requirements

The boundaries between these software can become blurry!

2. Basic Concepts

*2.7. **Software Engineering** versus **Computer Science***

- **Computer Science:** focuses on theory and fundamentals
- **Software Engineering:** Practices of developing and delivering useful software

CS knowledge is useful for SE, such as physics is useful for ELE

- **Software Engineering** versus **Systems Engineering**?
  - **Systems Engineering:** covers hardware and software aspects of computer systems
    - Designing overall system architecture and then integrating different parts
      - Including: policy, process design, and system deployment
  - **Software Engineering:** part of system engineering concerned with software

2. Basic Concepts

2.8. Rising challenges to software Engineering

- **Heterogeneity:** Distributed systems, across networks, diverse devices (from PCs to tablets and smart mobile phones)
- **Business and social change:** Changing software to adapt to emerging socio-economic trends
  - Smart phone applications, web services, cloud computing, etc.
- **Security and trust:** Software is becoming intertwined with human's lives
  - Need to trust that the software will do the job
    - Especially remote software (such as Web services or Cloud)
  - Need protection from malicious code/users

2. Basic Concepts

2.9. Impact of the Web

- The **Web** went from a universal data store in the 1990s
  - To an ever increasing software services repository around 2000
    - **Web applications** (client/server side)
    - **Web services**
      - Self-contained, modular applications that can be described, published and invoked over the Internet
      - Executed on the remote system where it is hosted
    - **Cloud computing**
      - On-demand provisioning of (unlimited) computational resources as a utility
        - Infrastructure as a Service : Raw processing capabilities
        - Platform as a Service: A software platform (OS, programming environment)
        - Software as a Service: Turnkey applications

2. Basic Concepts

2.9. Impact of the Web

- **Web-based development** instead of **special-purpose user interface**
  - Run on browsers instead of PCs
    ⇨ **Cheaper** to **change** and **update** software
- **Web services**: invoking and using software as needed
  - **Composition**: combining simple services to perform sophisticated tasks
    ⇨ More emphasis on **software reuse**
- **Software as a service:** Cloud service-based utilities
  - No need to develop and deploy local specific software solutions
    ⇨ Highly **distributed** pay as you go **systems**

2. Basic Concepts

2.9. Impact of the Web

- **Impact on software processes:**
  - Software reuse has become a dominant approach
    - Thinking about **assembling**/integrating existing software components
      - Before designing a new one from scratch
  - Impractical to specify **all** system requirements in advance
    - Web-based solutions can be developed and delivered **incrementally**
  - User interfaces are constrained to the functionalities of Web browsers
    - **Poorer capabilities** in comparison with personalized PC software
      - Although it is evolving rapidly

3. SE Ethics

- SE involves wider responsibilities than application of technical skills

- A software engineer:
  - Should hold normal standards of honesty and integrity
  - Should respect SE codes of ethics and professional practice
    - *ACM/IEEE*, *Lebanese Order of Engineering*, etc.

      - **Confidentiality:** of employer and clients
      - **Competence:** should not misrepresent one's levels of competence
      - **Intellectual property rights:** of employer and clients
      - **Computer misuse:** from game playing to virus dissemination

4. Case Studies

- Three different case studies are considered in this course
  - Describing different types of systems and application scenarios

1. Embedded system

- Software controls a hardware device and is embedded in the hardware
  - ⇨ Issues: physical size, responsiveness, power management, etc.

2. Information system

- Software to manage and handle access to a data repository (e.g., database)
  - ⇨ Issues: security, privacy, data integrity, etc.

3. Sensor-based data collection system

- Software to collect and process data from a set of sensors
  - ⇨ Issues: reliability, maintainability, limited resources, etc.

4. Case Studies

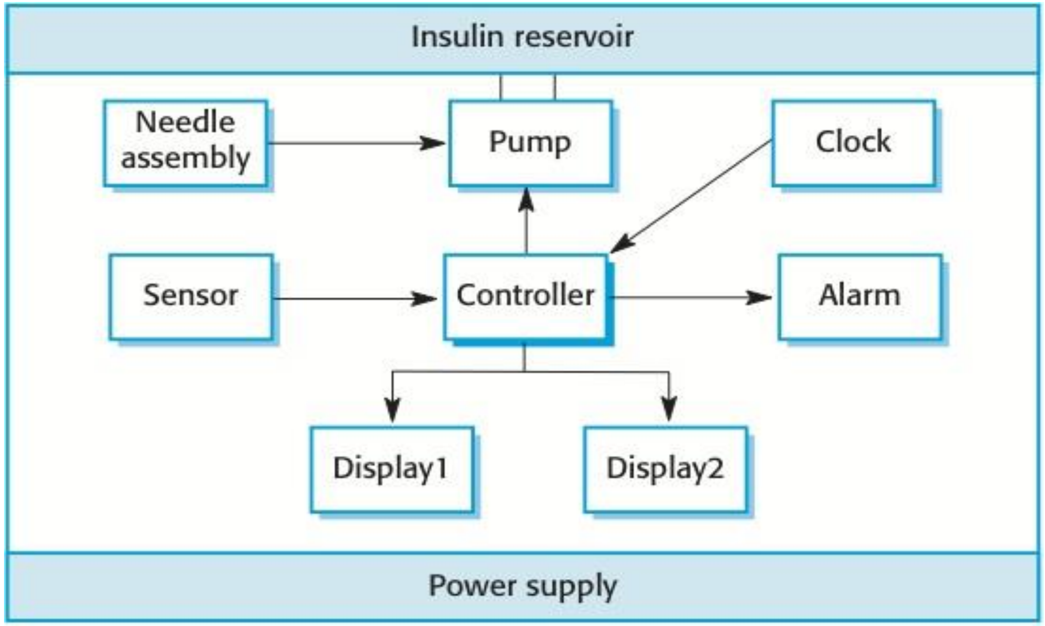
4.1. Insulin Pump Control System

- **Insulin:** hormone produced by the pancreas metabolizing glucose in the blood
- **Insulin pump system:** simulating the operation of the pancreas
  - Collects information from the patient's blood using sensors:
    - Blood glucose level
    - Time of last injection
  - Delivers a controlled dose of insulin to the patient
    - Pump delivers one unit of insulin in response to a pulse from a controller
- **Main non-functional requirements:**
  - **Reliability:** the system should deliver the right amount of insulin
  - **Safety:** the system should deliver the correct amount of insulin
  - **Availability:** the system should always be ready to deliver

4. Case Studies

4.1. Insulin Pump Control System

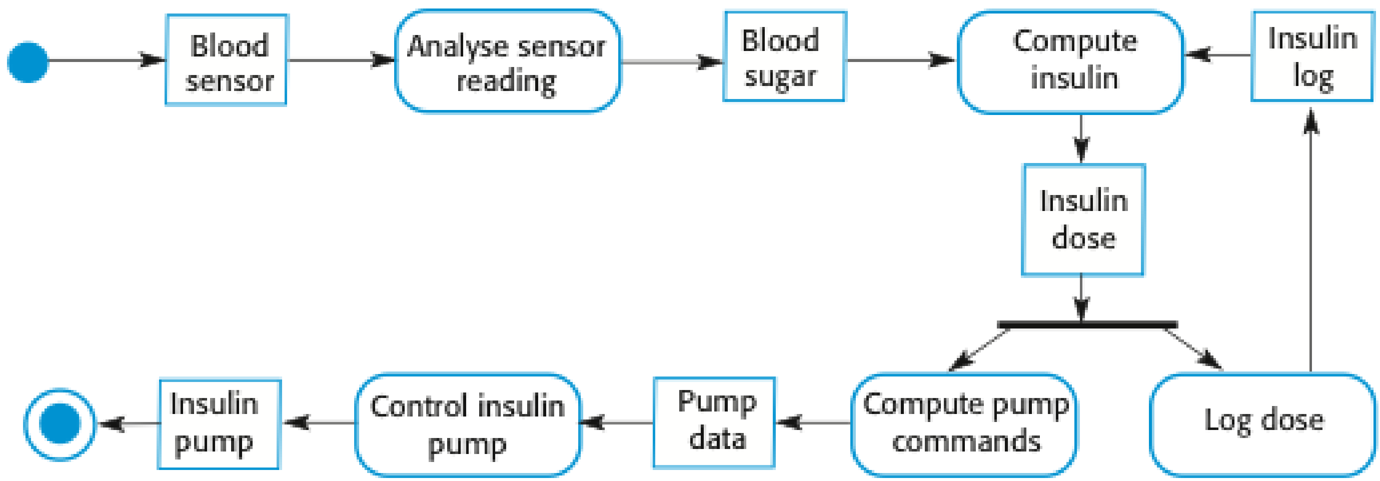
- Hardware components:



4. Case Studies

4.1. Insulin Pump Control System

- Activity model:



4. Case Studies

4.1. Insulin Pump Control System

- **Safety-critical system**
  - System failure to inject the correct amount of insulin can have serious health consequences
    - ⇨ May result in user (patient)'s injury, or loss of life
- **Dependability is central**
  - Safety, reliability, and availability
- **Formal requirements specification** is essential
  - Since computing the amount of insulin required is complex and difficult to specify using natural language
    - – Procedure should be thoroughly described and formalized in advance

4. Case Studies

4.2. IS for Mental Health Records

- IS that maintains data about patients with mental health problems

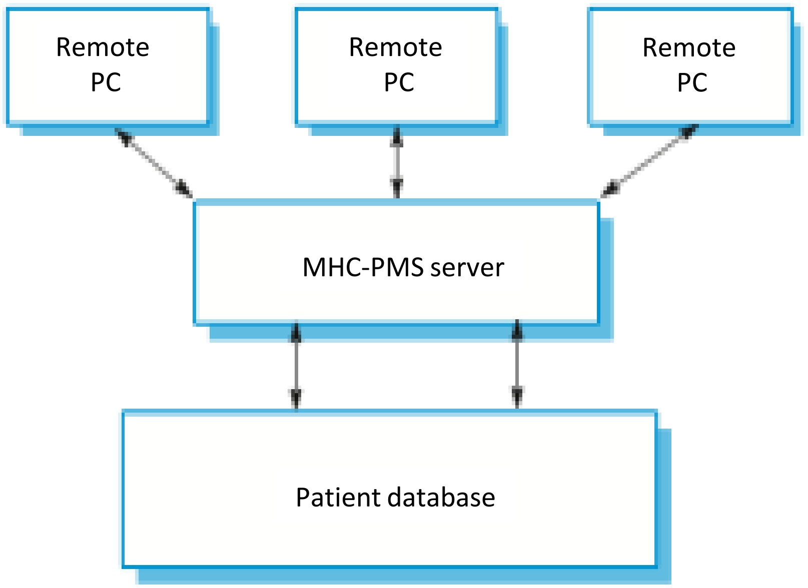
- **MHC-PMS** (Mental Health Care – Patient Management System)
  - Centralized database (DB) storing mental patients' data
  - Access from remote computers (in mental health clinics)
  - Remote computers can download data from the DB (for use when disconnected)
  - System may interact with other medical IS (to acquire complete medical data)

  Main objectives:
  - Providing clinical staff with timely information to support treatment
  - Generate data reports that allow health service managers to monitor patients

4. Case Studies

4.2. IS for Mental Health Records

- **Key functional features:**
  - Individual care
    - Patient records and history
  - Patient monitoring
    - Issuing warning when necessary
  - Report generation
    - Monthly management reports
- **Key non-functional features:**
  - Data privacy and security
  - Safety (e.g., warning of suicidal cases)
  - Availability (data should always be accessible)



MHC-PMS organization

4. Case Studies

4.2. IS for Mental Health Records

- **Secondary Safety-critical system**
  - System failure can lead to decisions that compromise the safety of the patient or the medical staff caring for the patient
    - ⇨ It indirectly affects user (patient)'s safety

- Both **Security & Reliability** are central non-functional requirements

- **Requirements specification** is important
  - Can be expressed formally and/or in natural language
  - Scenario particularly useful for highlighting **requirements conflicts**
    - Between patient privacy and maintaining the safety of the patient and their carers

4. Case Studies

4.3. Wilderness Weather Station

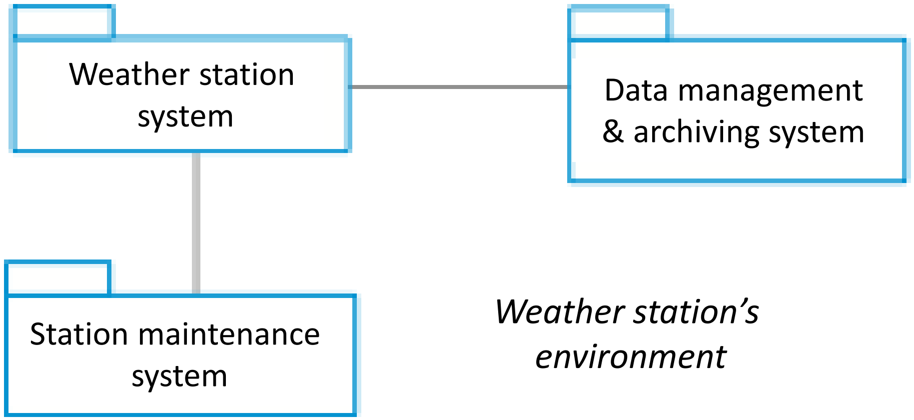
- Monitoring climate change and improving the accuracy of weather forecast
  - Deploying hundreds of remote weather stations
  - Collecting data from sensors: temperature, pressure, sunshine, wind, etc.

- **Wilderness Weather System:**
  - **Remote weather station:** Carries initial processing before transmitting to data management system
  - **Data management system:** collects, analyzes, and archives data from stations
    - For use by other systems (such as larger weather systems)
  - **Station maintenance system:** Communication (by satellite) with remote stations
    - Monitor and control stations, report problems, update embedded software

4. Case Studies

4.3. Wilderness Weather Station

- Main constraints:
  - Limited bandwidth
  - Limited power supply
  - Hazardous environment



Main Non-functional features

- **Reliability:** Instrument monitoring, managing power system, batteries, etc.
- **Maintainability:** dynamic reconfiguration of software, backup instruments, etc.

4. Case Studies

4.3. Wilderness Weather Station

- **Self-contained system**
  - Need for **power management** in situations where available power is restricted
  - Need for **remote dynamic reconfiguration**, where the system software is changed by remote interaction

- **Reliability & Maintainability** are central non-functional requirements

- **Illustrates object-oriented development**
  - Instruments in the system can be represented as objects
  - As can the data that is collected and uploaded to the IS
  - Different interactions between different objects:
    - Remote stations, data management system, and station maintenance system

Ch. 1 - Introduction

Support Material and Exercises

- Available in the textbook - Chapter 1
- Available with solution keys on **Blackboard!**