Chapter 2

Principles of Software Engineering

Objectives:

- The Software Life Cycle
- Software Reliability
- The Software Life Cycle and Engineering Problem-Solving
- Case study

2. Principles of Software Engineering

- Engineers from all disciplines are often heavily involved in the development of large and complex software systems.
  - Flight simulators, chemical plant process control, electrical circuit simulators,…
- There are huge problems associated with the development of large software systems.
  - The problems encountered are similar to those found in any large engineering project:
    - Product design, human resources, management, cost control, selection of tools, quality control…
- In the past the lack of formal software development methodologies had proven fatal for the development of large software systems.
  - Projects were late and over-budget and the resulting software was unreliable, under-performing, and difficult to maintain.
- The discipline of software engineering appeared near the end of the 1960's during the so-called "software crisis" in response to the problems described above.
  - This is an emerging discipline dedicated essentially to the development software products and large software systems.

- Many definitions of software engineering have been proposed. Most definitions include the following points.
  - Discipline dedicated to the construction of large software systems where usually a team or teams of software designers and programmers are involved in the project.
  - Based on sound engineering principles.
  - Requires the interaction of many disciplines that can be technical or non-technical.

- A software engineer must:
  - have a sound background in computer science,
  - be able to communicate effectively (oral and written - just like any other engineer),
  - be able to interact reasonably well with other human beings.

- By software we include:
  - programs,
  - internal documentation (comments in the code),
  - external documentation (software report, users manual and help files).

  - ➢ In a large software system, the documentation is <u>absolutely</u> essential to the maintenance of the software and directly affects its longevity.
  - ➢ In a large software system, the effort required for the preparation of the documentation can be as large as the effort required for the software design and programming.

2.1 The Software Life Cycle

- It is generally agreed upon that the following 5 stages are present in the "life cycle" of a software system designed based on sound software engineering principles.

    1. Requirements analysis and definition.

        – The services to be provided by the system, its goals, and the constraints under which the system must operate are established in consultation with the system users or the client(s).

        – The requirements are then defined in a manner that is understandable by both the users and the system development staff.

        – In the end, a requirements document that is understandable by the users (possibly non-technical) and the development staff results.

2. System and software design.

- Based on the requirements definition, the system is partitioned into hardware and software components… this is commonly termed the system design.
- Basically, software design consists in representing the tasks to be performed by the software system in a manner that can be easily coded into a set of programs and/or sub-programs (in C, a sub-program is a function).
- Generally, algorithms are designed and selected at this point.
- Test plans are also developed at this stage.

3. Software implementation and unit testing.

- At this stage, the software system designed in step 2 is coded into a number of programs and/or sub-programs using a programming language that is found to be suitable and that can generate code that is executable on the targeted hardware platform(s).
- Each program and sub-program is tested individually on the intended hardware platform(s) to ensure that it meets its specifications.

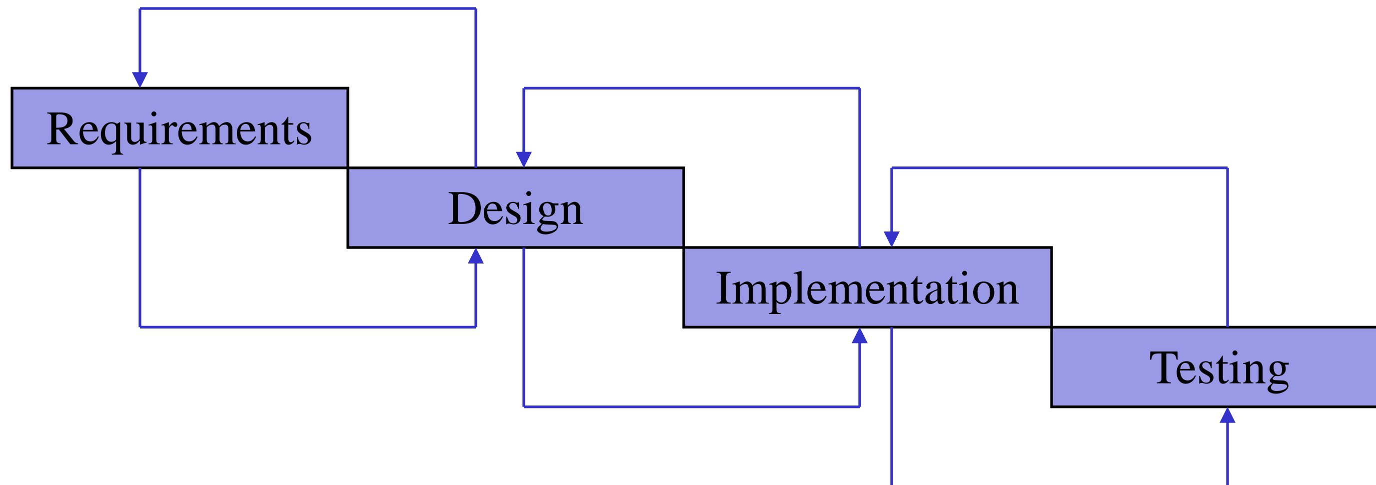
4. System integration and testing.

- At this stage, the programs and sub-programs are integrated to create the complete software system.
- The software system is tested and verified on the intended hardware platform(s) to ensure that it meets all of the requirements and definitions established in step 1.
- After testing and verification, the system is delivered to the users or client(s).

5. Operation and software maintenance.

- **This is often the longest stage in the software life cycle.**
- The system is installed and put into practical use.
- Software maintenance involves:
  - correcting errors (and bugs) which may have remained undetected,
  - improving the software implementation (new and faster algorithms),
  - adding functionality and services as new requirements are perceived.

- In reality, the software development cycle does not consist of a linear progression through steps 1 to 4. There is significant information exchange throughout the development cycle and the information can flow forward and backward as represented schematically below:



- During the operation phase of the software life cycle, maintenance activities may include:
  - the definition of new requirements that have been perceived;
  - modifications to the software design (due to advances in hardware for example);
  - a re-coding of certain or all program and sub-programs due to advances in programming languages (say migrate from COBOL to C++) or algorithms;
  - additional testing and verification.
  - ➢ The maintainability of software is obviously very important since it will directly affect the software's longevity.
  - ➢ Improving the maintainability of a software product is a goal of software engineering.

2.2 Software Reliability

- As software applications began to pervade everyday life, it became apparent that <u>software reliability</u> was a very important issue.
  - In many cases, it is the most important issue after safety.

- The reliability of a software application depends entirely on its design and implementation.

- A generally accepted and realistic definition of software reliability is given below.
  - Software is considered reliable if:
    - it satisfies all requirements, specifications and constraints,
    - it never produces an incorrect output irrespective of the input,
    - it can never be corrupted or modified by the user,
    - it takes useful and intelligent actions in unexpected circumstances,
    - it fails completely only when further progress is impossible.
  - Software reliability is thus a <u>measure</u> of its ability to provide the required services.

- Often times a single software failure may be sufficient to declare the application unreliable.
  - E.g. Software that controls the landing gear of an aircraft.

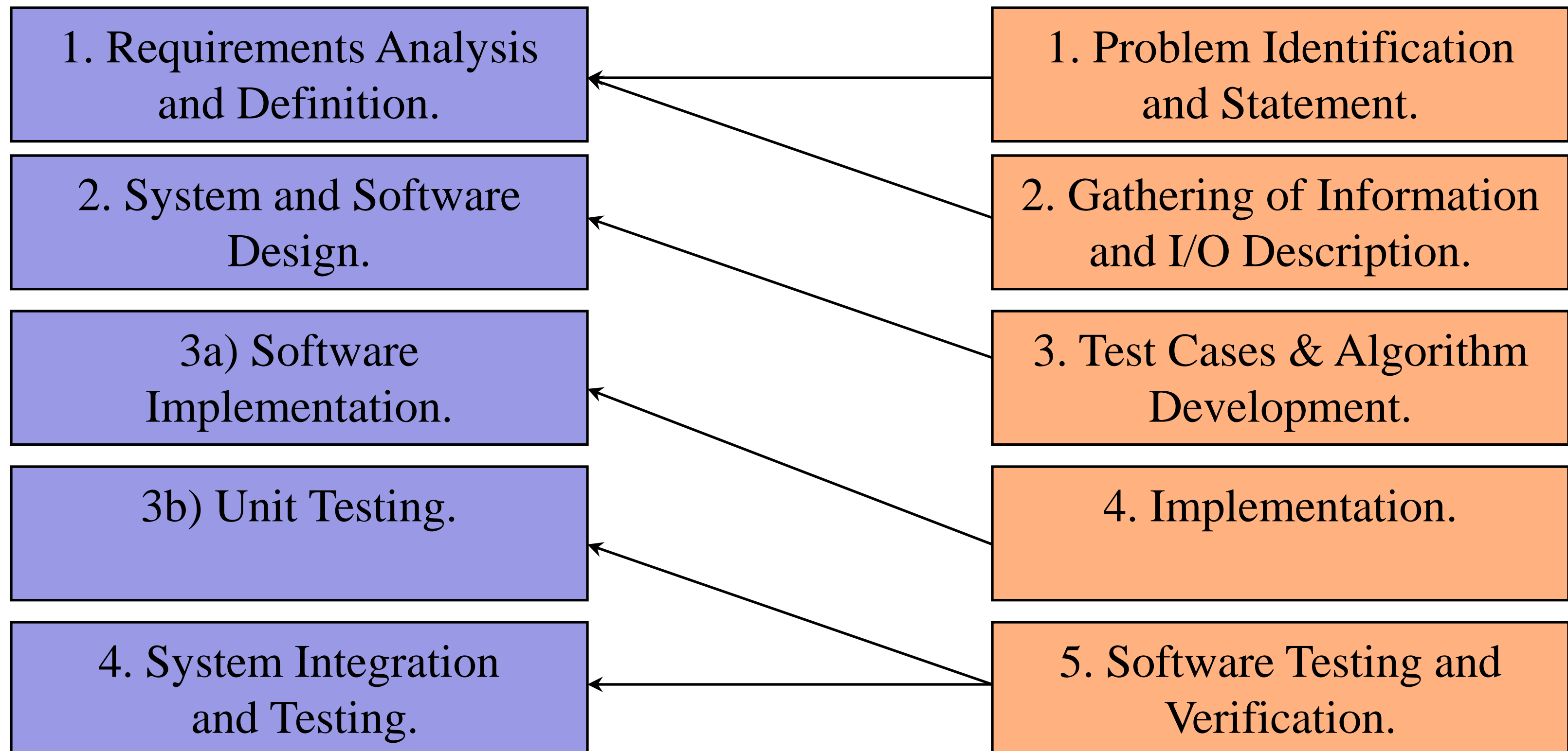
Software Reliability Versus Speed of Execution

- Making software reliable requires a large amount of often redundant code to make verifications such as validating user entries, verifying the integrity of data and checking intermediate results.
  - This extra code usually slows down the execution of the program and increases the amount of memory required to run the program.
  - ➔ Software reliability is traded off against execution time and memory requirements.

- In most applications, the most important criterion that determines the quality of the software is reliability. A few reasons are listed below.
  - Computer hardware evolves rapidly and is becoming faster with time (Moore's law).
  - A software application that is unreliable will be avoided by users and will rapidly become useless; furthermore, unreliable software can cause considerable losses of data.
  - In certain applications, reliability is by far the most important criterion since a software failure can be completely unacceptable:
    - E.g.: Aircraft navigation control system, nuclear reactor control system…
  - Slow software can often be improved by optimizing the code where most of the execution time is spent since this is often in a few very distinct sub-programs.
  - Slow but reliable software is predictable so activities can be planned accordingly.

2.3 The Software Life Cycle and Engineering Problem-Solving.

- Engineering problem-solving using computers usually follows a prescribed methodology that incorporates sound engineering principles, such as the methodology that we have adopted in this course. The software life cycle and our problem solving methodology interact as shown below.



Case Study – Temperature Conversion

The problem is to design a software that prompts the user to enter a temperature in Fahrenheit and outputs the equivalent in Celsius.

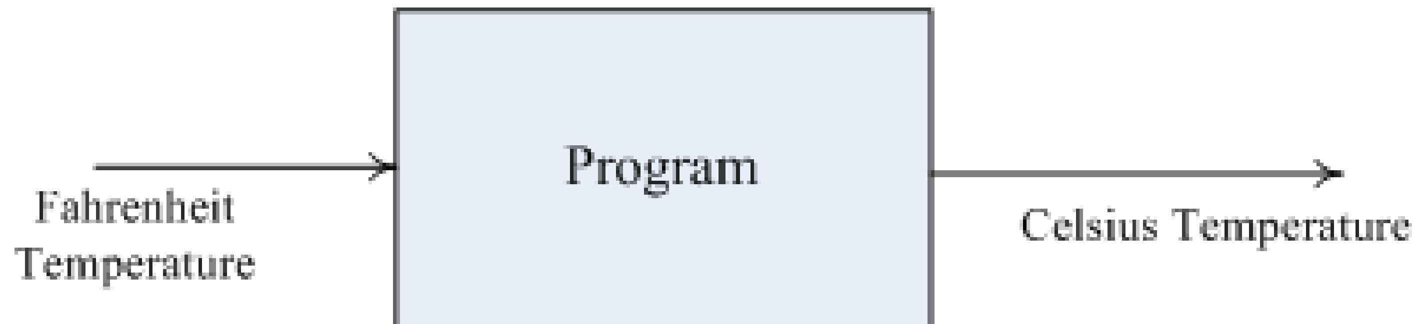
- Using the problem solving methodology
  - Step 1: Problem Identification and Statement
  - Step 2: Gathering of Information and I/O Description
  - Step 3: Test Cases and Algorithm Development
  - Step 4: Implementation
  - Step 5: Software Testing and Verification

Step 1: Problem Identification and Statement

- A software shall be developed to convert the temperature entered by the user from Fahrenheit to Centigrade

Step 2: Gathering of Information and I/O Description

- Find the equation to make the conversion
  - $°C = (°F - 32) / 1.8$
- Input/output description



Step 3: Test Cases and Algorithm Development

- Step 3-a: test cases
  - Test case 1:

    F = 0 -> °C = (0 – 32) / 1.8 = – 17.76

  - Test case 2:

    F = 32 -> °C = (32 – 32) / 1.8 = 0

  - Test case 3:

    F = 193 -> °C = (193 – 32) / 1.8 = 89.45

Step 3: Test Cases and Algorithm Development

- Step 3-b: Algorithm Development

Print "Please enter the temperature in Fahrenheit"

Read a value into F_Temp

Assign (F_Temp - 32)/1.8 to C_Temp

Print "The temperature in Celcuis is", C_temp

Step 4 - Implementation

```
#include <stdio.h>
#include <stdlib.h>

// main function
int main()
{
    float F_Temp, C_Temp;

    printf("Please enter the temperature in Fahrenheit\n");
    scanf("%f", &F_Temp);
    C_Temp = (F_Temp - 32)/1.8;
    printf("The temperature in Celcuis is %f\n", C_Temp);

    system("PAUSE");
    return 0;
}
```


UNIVERSITÉ D'OTTAWA • UNIVERSITY OF OTTAWA
Génie • Engineering

Step 5: testing and verification

- Test case 1:

```
C:\Documents and Settings\MCRLAB\Desktop\Untitled1.exe
Please enter the temperature in Fahrenheit
0
The temperature in Celcuis is -17.777779
Press any key to continue . . .
```

- Test case 2:

```
C:\Documents and Settings\MCRLAB\Desktop\Untitled1.exe
Please enter the temperature in Fahrenheit
32
The temperature in Celcuis is 0.000000
Press any key to continue . . .
```

- Test case 3:

```
C:\Documents and Settings\MCRLAB\Desktop\Untitled1.exe
Please enter the temperature in Fahrenheit
193
The temperature in Celcuis is 89.444443
Press any key to continue . . .
```

Thank You!

Ευχαριστώ

MULTUMESC

ขอบคุณ

Vielen Dank

Teşekkürler

Merci

DMnvwd

شكراً

متشكرم

Gracias

Grazie

Bedankt

Dankie

THANK YOU

Köszönettel

Hvala

Obrigado!

Asante

شكريا

Díky

謝 謝

WAD MAHAD

SAN TAHAY

감사합니다

GADDA GUEY

Urakoze