SENG1110/SENG6110
Object Oriented Programming

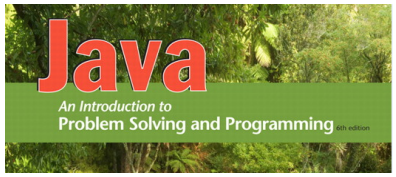
THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

Lecture 1a

Introduction



Outline

- Elements of a Computer System
- Hardware
  - Hardware Components
- Software
  - Languages
  - The Software Life Cycle

Hardware and Software

- Computer systems consist of *hardware* and *software.*
  - Hardware includes the *tangible* parts of computer systems.
  - Software includes *programs* - sets of instructions for the computer to follow.
- Familiarity with hardware basics helps us understand software.

Hardware and Memory

- Most modern computers have similar components including
  - Input devices (keyboard, mouse, etc.)
  - Output devices (display screen, printer, etc.)
  - A processor
  - Two kinds of memory (main memory and auxiliary memory).

The Processor

- Also called the *CPU* (central processing unit)
- The processor **processes** a program's instructions.
- It can process only very simple instructions.
- The power of computing comes from speed and program intricacy.

Memory

- Memory holds
  - programs
  - data for the computer to process
  - the results of intermediate processing.
- Two kinds of memory
  - main memory
    - RAM is short for random access memory
  - auxiliary memory
    - Also called secondary memory
    - Disk drives, CDs, DVDs, flash drives, etc

Files

- Large groups of bytes in auxiliary memory are called *files.*
- Files have names.
- Files are organized into groups called *directories* or *folders.*
- Java programs are stored in files.
- Programs files are copied from auxiliary memory to main memory in order to be run.

Programs

- A *program* is a set of instructions for a computer to follow.
- We use programs almost daily (email, word processors, video games, bank ATMs, etc.).
- Following the instructions is called *running* or *executing* the program.

Software

- Software consists of programs written to perform specific tasks
  - **System software** – System programs control the computer.
    - operating systems
    - communications software
    - compilers/interpreters
  - **Application software** – allows specialized tasks. Includes
    - word processors
    - Spreadsheets
    - web browsers
    - Database systems
    - other programs we write

Operating systems

- The *operating system* is a supervisory program that oversees the operation of the computer.
- The operating system retrieves and starts program for you.
- Well-known operating systems including: Microsoft Windows, Apple's Mac OS, Linux, and UNIX.

The Software Life Cycle - 1

- Large systems must be
  - planned before coding
  - thoroughly tested and optimized before release
  - maintained after release

The Software Life Cycle - 2

- Requirements
- Analysis
- Design
- Implementation (sometimes called Development)
- Integration
- Maintenance

Requirements Phase

- A broad statement of a problem that could possibly be solved using a computerized solution
- It could be as simple as:
  - Calculate the area of a circle given its radius
- Or as complicated as:
  - Computerize the taxation system of the US Government
- US President John F Kennedy told NASA to
  - “put a man on the moon inside 10 years”.

Analysis Phase

- Also called the system **specifications phase**
- Spells out what the system will do to solve the problem posed by the customer
- **Does not describe how the system will do** what it does
- It involves the start of another very important process - MODELLING.
- What we need to do very early in this course is to realise that what we are doing when we develop software is to build a MODEL of some real-world system. If the model is correct then the other phases of development will proceed smoothly and the resulting system will do what it’s supposed to. If not … … ?

Design Phase

- Describes in detail **how the system will do** what our analysis says it will do
- Makes use of pseudocode rather than program code

Implementation Phase

- Also called the coding phase
- Describes the design using a programming language
- This is the phase where the programs are actually written (in our case in Java).

Integration Phase

- Making the new system run.
- Making sure that the data required for the new system is available when and where it is needed and that it supplies correct and timely information to other systems that require it.

Maintenance Phase

- Programs might have a lifetime of 5-20 years
- Requirements might change, and minor or major changes could be necessary
- Maintenance is keeping the system running, by
  - exposing/finding/identifying errors and fixing them, and by,
  - extending the system to provide new functionality within the original overall design framework.
- Our early programs will be so small that Integration and Maintenance will be trivial phases (but programs don't have to get much bigger than this before these phases start becoming real issues).

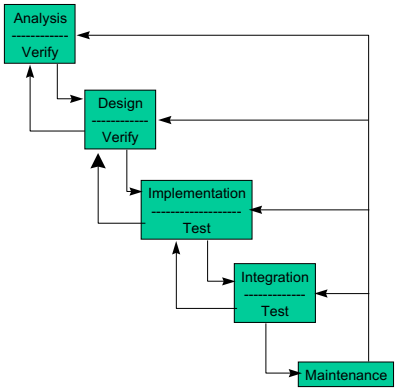
Programming vs Software Engineering

- All we've done is write a very simple pseudo-code.
- The **Customer Request** and **Analysis** phases are trivial in this case
- The **Implementation** phase is interesting, but only if you haven't seen much about programming.
- The **Integration** phase involves nothing more than running the compiler, Java 7.0 or 8.0 installed on our computer
- Hopefully, the **Maintenance** phase will be virtually zero.
- But, WHAT HAPPENS IN TODAY'S REAL SYSTEMS

Software Engineering

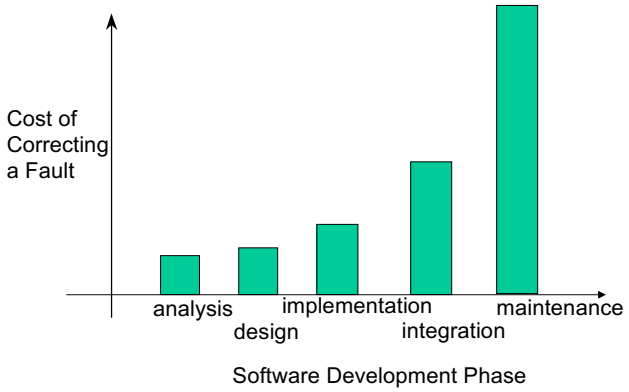
- The reality of today is that software systems are extremely large, much too large for a single person to develop on their own
- Another reality is that the "usual size" of a Software System that you may develop in 5 years time will be twice as big as a "usual" system of today
- Software Engineering is more than writing programs:
  - It involves applying processes and methodologies that will allow a team to produce a (potentially very large) system which
    - is correct, on time and on budget
    - will be easily and affordably upgraded at a later date
  - The processes must scale up to much larger systems

The Life Cycle as a Waterfall



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

The Cost of Correcting an Error



THE UNIVERSITY OF NEWCASTLE AUSTRALIA

More about errors - examples

- Problems for Individuals
  - The auto insurance rate of a 101-year-old man suddenly tripled. The program was written to handle ages only up to 100.
- System Failures
  - American Express Company's credit card system failed during the Christmas shopping in 1999.

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

More about errors - Example: The Therac-25

- The Therac-25 was a software-controlled radiation-therapy machine used to treat people with cancer
- Between 1985 and 1987, this machine at 4 medical centers gave massive overdoses of radiation in 6 patients.
- The patients received 13,000-25,000 rads. The right was 100-200 rads.
- Three patients died.
- **What went wrong?**

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

More about errors - Example: The Therac-25

- Design problems
  - Therac-25 followed earlier machines (Therac 6 and 20, these no fully computer controlled)
    - Reuse of the software from earlier models without testing/verify.
  - Weakness in the design of the operator interface.
    - Many errors/codes and go to manual.
- Bugs
  - I will tell you in some weeks…

More about errors - Contributing Factors

- Complexity of real-time, multitasking systems
- Failing to plan for unexpected inputs or circumstances
- Interactions with physical devices that do not work as expected
- Inadequate management
- **Insufficient testing**
- Pressure to get a product out quickly
- Inadequate response when problems reported
- Inadequate attention to safety risks
- Data-entry errors
- Inadequate training of users
- Errors in interpreting results or output
- Overconfidence in software
- Lack of incentive to do a better job

Your task

- Read
  - Chapter 1 of the text book
  - Read the first chapter or the first presentation of the book Software Engineering by Ian Sommerville:http://www.softwareengineering-9.com