Week 1

Introduction

Software development

What is software?

The word software denotes instructions for a machine, organised into helping humans use the machine. The physical machine (the hardware) can be anything, but to be controlled by software it must include a microprocessor.

Why not just have the machine and use it? Why software?

Every machine's functionality is embodied in its hardware, but some machines are more limited than others. An old-fashioned hairdryer blows air at two or three heat settings and that is all. Controls can be implemented using fairly simple electrical circuits.

A programmable washing machine is also limited in what it can do but there is scope for variation. To allow users to configure a washing machine, an electronic circuit (pure hardware) could be used, but it this would have to be more complicated than the simple circuits of the hairdryer. The alternative is to have a processor and software. The processor consists of circuits that perform a limited number of primitive operations and software is a recipe for how these primitive operations should be combined to achieve something. For all but very simple functionality, this is **cheaper to produce than a specialised electronic circuit**.

However, the true value of this setup (processor + software) lies in another property it has. While a washing machine can only wash clothes (admittedly in several different ways), less specialised hardware and a more complex user interface can provide user-defined functionality through the **theoretically infinite combinatorial power of sofware as a recipe**. This is what computers and smartphones do. The versatility of their user interfaces allows for ever new functionality to be invented and their internal programmabiliy supports its implementation.

What is software physically?

A unit of software usable for some purpose is called a software program, application, app or product. One of these is like the contents of a book, e.g. Oliver Twist by Charles Dickens. When we call out that name we generally do not mean any particular paper or electronic copy of the story, we mean the words that Dickens wrote and that we read. It is like that with software, which, although contained in a file on disk or in the memory of a device, cannot be equated with where it is stored but consists of abstract instructional content, which can be copied and stored in multiple places at once. **It is not physical.**

How does software development fit into all this?

Putting together a program does not require listing of primitive instructions understood by a processor. These are packed into higher level constructs and ready-made software chunks of a programming language, which is less granular and more user-friendly for humans. Different programming languages present the underlying processor capabilities in different ways, depending on their purpose but also simply on the preferences of their designers.

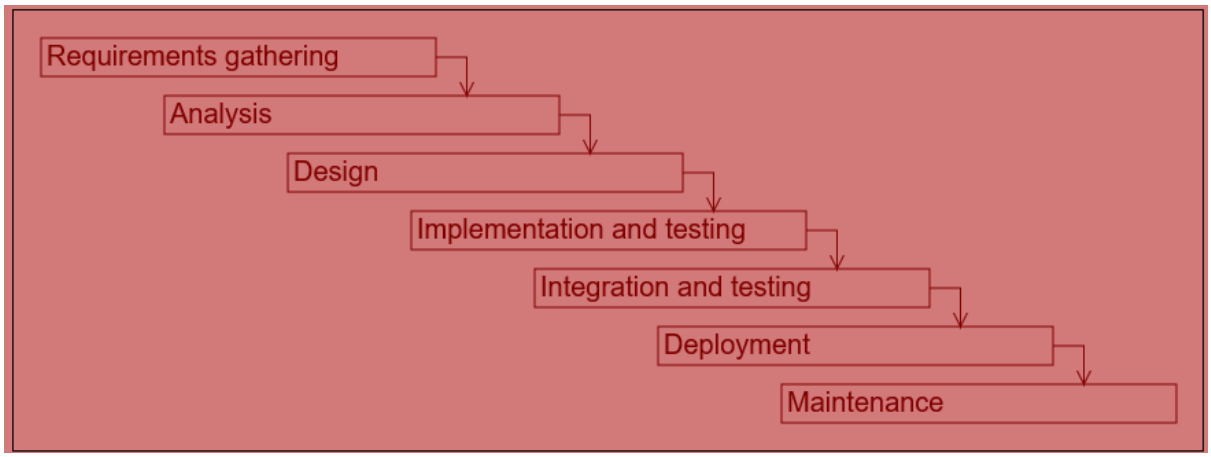
What is the difference between software development, programming and coding?

Different people and even different dictionaries will give you different definitions, with the distinction between the terms varying from huge to none. However, when a distinction is made,

software development is thought of as broader than programming, which in turn is treated as broader than coding. So, coding is simply writing the instructions (or code) in the language; programming involves a bit more e.g. design and testing but is still performed by one person; software development is the entire process that starts with an idea and ends in a product and could involve many people and organisations. Software development can also be used as a term for the entire industry involved in producing software.

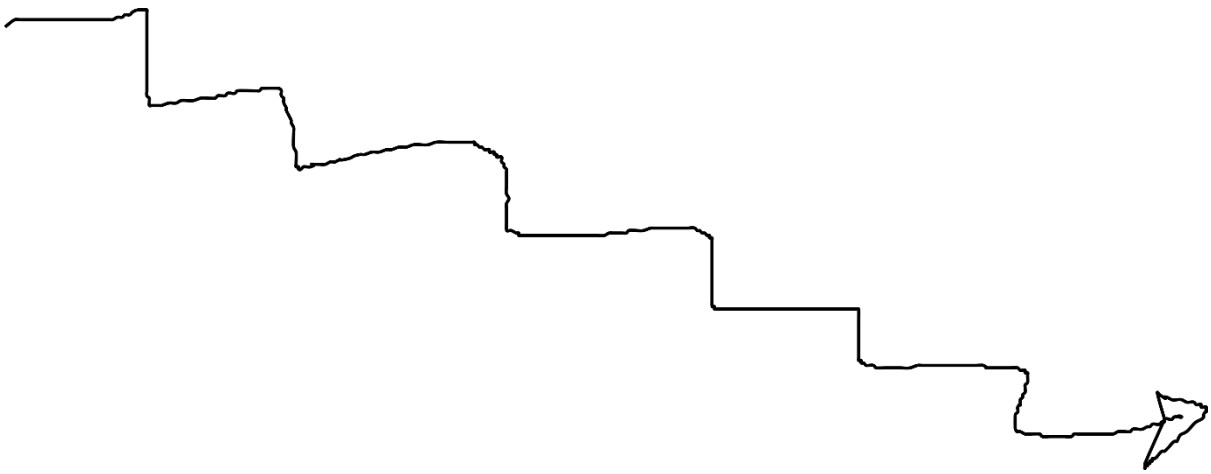
Methodologies

Developing software is a complex undertaking that includes several different types of activity within its lifecycle (the software development lifecycle - SDLC):



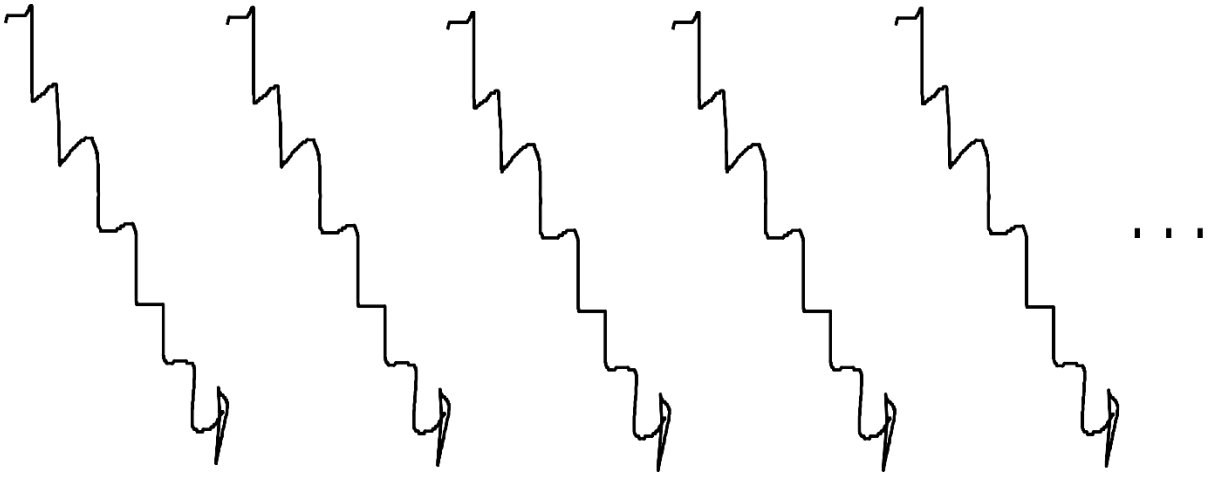
Depending on the context, the way these activities are interconnected and carried out as well as the degree of formality employed will differ greatly. However a **methodology** will generally be adhered to in one form or another.

The two main methodology styles are:



waterfall

- projects are large
- each activity is a single phase in a large project
- phases last weeks or months
- a project generally corresponds to a single release



agile (e.g. Scrum, XP)

- projects consist of many small iterations (1-2 weeks)
- constant updating of requirements based on feedback
- constant communication between teams and stakeholders (internal and external)
- continuous delivery and even deployment
- goes hand-in-hand with DevOps (also promotes connectedness of activities and roles but is centered around *delivery* as opposed to *development* in agile, see this)

But there are many more that are either variations on these two themes or sit somewhere between the two, for example:

Rapid Application Development (RAD)

- less focus on design and documentation
- prototyping
- multiple iterations

'Sashimi' model - with overlapping phases

Methods and tools

There are methods and tools for every activity in the software development cycle.

requirements gathering and analysis,

- using tools such as Eclipse-based Rational tools or VisualParadigm
  - use cases e.g. as prescribed by Rational Unified Process (RUP) framework
  - user stories in agile
  - source-stimulus-environment-artefact-response-measure for non-functional requirements
  - analysis-level class diagrams

design, using tools such as Eclipse-based Rational tools or VisualParadigm

- component diagrams (describing high-level structure)

- design-level class diagrams (describing structure)
- sequence diagrams (describing behaviour)

implementation

- based around programming in languages such as Python, Java, C++
- supported by many different types of tools:
  - syntax-aware text editors e.g. Visual Studio Code, Notepad++
  - interpreters and compilers for
  - debuggers
  - program profilers and validators e.g. Valgrind
  - version control systems e.g. git implemented by GitHub, Bitbucket etc.
  - integrated development environments (IDEs) e.g. Eclipse, NetBeans, PyCharm; providing integrated access to all of the above listed tools, typically
    - text editor
    - compiler/interpreter
    - debugger
    - profiler
    - version control

integration, deployment, maintenance

- tools for the automation of configuration and deployment such as Puppet, Chef, Ansible

planning and management (orthogonal to all of the above) also require tools for

- defect tracking
- project management (traditional)
- agile tools, which include both of the above

The substance of software and programming languages

Software has two aspects, both equally important:

- **information**, represented in a program by **data structures**
- **behaviour**, represented in a program by **algorithms**

A **programming language** consists of vocabulary, syntax and rules for representing *information* and *behaviour*.

Here are some broad classifications of programming languages:

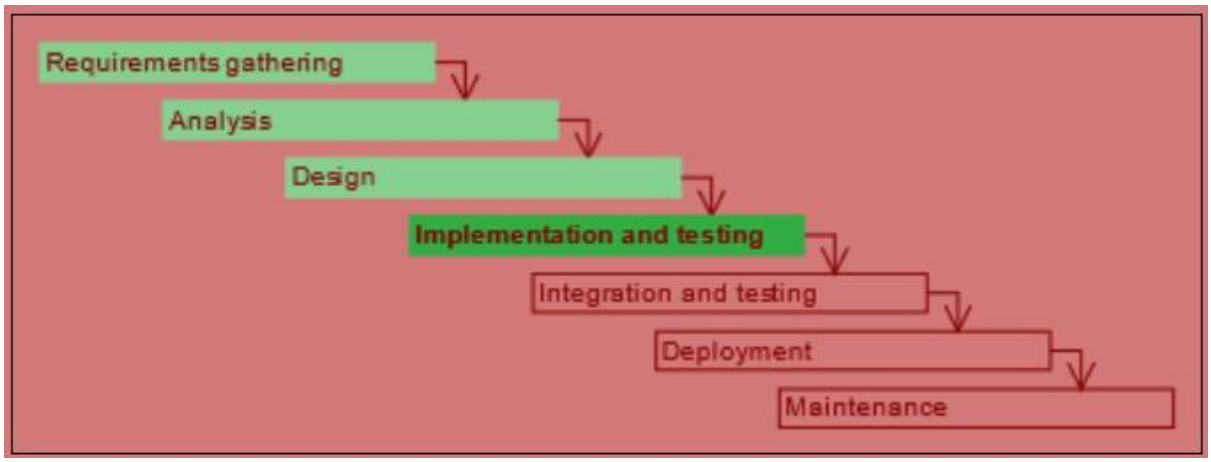
- imperative (Java) vs. declarative (SQL), by focus, on controlling behaviour vs. information

- object-oriented (C++) vs. procedural (C), by how behaviour and information are organised
- high-level (e.g. C++) vs. low-level (e.g. assembly), by closeness to the processor
- compiled (e.g. C++) vs. interpreted (e.g. Python), by mode of translation, in advance as a whole vs. during execution
- special-purpose (e.g. R for data analysis) vs. general-purpose (e.g. Python)

What does the *Object Oriented Software Development* module cover?

- This module focuses primarily on
    - **implementation and testing**

  but also provides an introduction to
    - requirements gathering
    - analysis
    - design

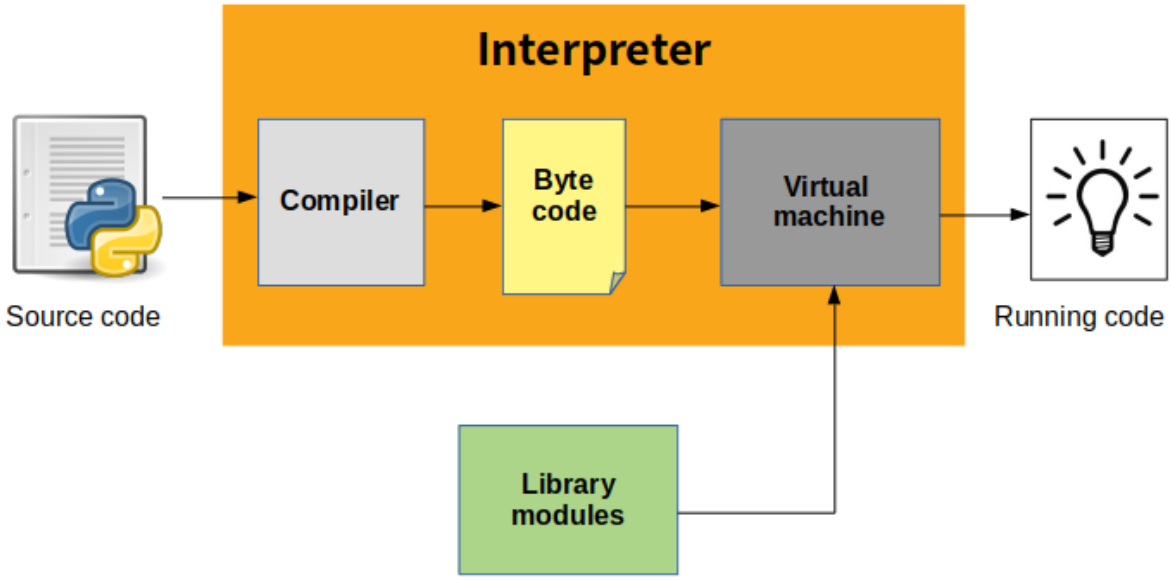


- In the module we employ **problem-based learning (PBL)**, which engages the student in learning through solving open-ended problems. This is particularly suited to software development as it mirrors the real-life (workplace and advanced study) circumstances of software development application.
- The module is taught through **Python**, a high-level interpreted general-purpose programming language.

Python and tools

- The Python programming language can be used
    - interactively on the command line for simple work
    - in one or more interacting scripts that can be kicked off to execute as a unit

- Python is an interpreted language, requiring no explicit compilation step on the part of the developer (traditionally this type of language would have been used for scripting). Python interpreter
(source: indianpythonista.wordpress.com)



- A program in Python (or any other language) is useful only if it interacts with its environment by consuming and providing information Python program inputs and outputs
(source: python-nitol.blogspot.com)