

# Documentation

## Lecture Plan:

- About Documentation
- Self-Documentation
- Literate Programming

## Why Bother with Documentation?

Software engineers build big, complex systems for other people in a professional manner

A computer-based system is of little value unless it can be used and maintained by others - this requires good documentation

# About Documentation

In a list of software quality factors, one might expect to find the presence of good documentation as one of the requirements. But this is not a separate quality factor; instead the need for documentation is a consequence of the other quality factors

*Object-Oriented Software Construction*

Bertrand Meyer

aka “Bertrand’s Bible”

## Levels of Documentation

- *external* documentation, which enables users to understand the power of a system and use it conveniently, is a consequence of ease of use.
- *internal* documentation, which enables software developers to understand the structure and implementation of a system, is a consequence of the extendibility requirement.
- *module interface* documentation, enabling software developers to understand the functions provided by a module without having to understand its implementation, is a consequence of the reusability requirement. It also follows from extendibility, as module interface documentation makes it possible to determine whether a certain change need affect a certain module.

*Object-Oriented Software Construction*

Bertrand Meyer

# Software Self-Documentation

Rather than treating documentation as a product separate from the software proper, it is preferable to make the software as self-documenting as possible.

This applies to all three kinds of documentation:

- By including on-line “help” facilities and adhering to clear and consistent user interface conventions...
- A good implementation language will remove much of the need for internal documentation ...
- It is ... possible to use tools to produce module interface documentation automatically ...

*Object-Oriented Software Construction*

Bertrand Meyer

# Module Interface Documentation

Interface documentation describes the contract a class offers those who use it.  
It must describe the following:

- The services the class can render.
  - The queries supported by the class.
  - The commands supported by the class.
  - *Their post-conditions.*
- The obligations users of the class must meet.
  - *Their pre-conditions.*
- *And, nothing else!*  
Recall the principle of abstraction.

# Code Self-Documentation

Documentation should be kept together with the thing it documents.

- Helps ensure that code and its documentation remain consistent.
- Often not practical for external documentation.
- Interface documentation should be extracted from the code:
  - The Eiffel 'short' tool.
  - The Java 'javadoc' tool.
- Contrast with Modula-2 and Ada with separate interface documentation.

# Carol's Rule for Documentation

Good documentation describes

**What is being done** ✓

rather than

**How it is being done** ✗

$i := i + 1$  — increment loop counter ✓

$i := i + 1$  — advance array pointer ✓

$i := i + 1$  — add 1 to  $i$  ✗

# Literate Programming

Another way to keep code and documentation together.

- Typically used to create *extremely* readable internal documentation.
- Reverses the balance of power between code and comments.
- You can document your code with your favourite mark-up language.
- You can therefore document not just with words, but with pictures, diagrams and equations . . .
- You can present the code in the order in which it is best explained.  
This is often not the same as the order in which it must be compiled.
- You can cross-reference all definitions and their uses in your code.  
(These can be hyper-linked when using HTML.)

# Literate Programming History

In 1983, Donald Knuth introduced literate programming in the form of Web, his tool for writing literate Pascal programs. Web lets authors interleave source code and descriptive text in a single document. It also frees authors to arrange the parts of the program in an order that helps explain how the program functions, not necessarily the order required by the compiler.

... noweb strips literate programming to its essentials. Programs are composed of named chunks of code written in any order, with documentation interleaved.

”Literate Programming Simplified”

Norman Ramsey

# Literate Programming Advantages and Disadvantages

## Advantages of literate programming

- Literate programs are easier to understand and maintain.
- Literate programming can weave a collection of classes, their design documentation, and their test data into readable documents.

## Disadvantages of literate programming

- Writing literate programs is time consuming.
- To be a good literate programmer you must
  - be good at writing programs
  - be good at writing prose

## Suggested Reading

From the COMP1110 Reading Brick:

- **Software Quality**

Meyer, Bertrand (1997) *Object-Oriented Software Construction*, 2nd Edition,  
Prentice-Hall, New Jersey  
Chapter 1 “Software quality” pp 3–20

- **Literate Programming Simplified**

Ramsey, Norman “Literate Programming Simplified”  
*IEEE Software*, Vol 11(5) September 1994, pp 97–105,