Software Engineering

What is Software Engineering?

Why should I learn about Software Engineering?

Some general aspects of Software Engineering

• Modelling
• Problem-Solving
• Knowledge Acquisition
• A Rationale Driven Approach

Why learn Software Engineering?

A structured approach is needed for large/complex systems

• Would you use the same approach for building a sky-scraper as you would for a garden shed?

Building large systems involves extensive group work

• A method for decomposing the problem into manageable portions is important.
• A common understanding of the task and proper communication is essential.
• Each member of the group needs to understand their task and how it interfaces with other tasks.
• Groups and individuals need to communicate with a commonly agreed design language.

The term software engineering first appeared in the 1968 NATO Software Engineering Conference [to provoke thought regarding the perceived “software crisis”].

“An engineering discipline that is concerned with all the aspects of software production.” [Sommerville]

“A profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.” [Wikipedia]

“A systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software.” [Laplante]

“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches: that is, the application of engineering to software.” [IEEE Computer Society’s Software Engineering Body of Knowledge]
Why learn Software Engineering?

An understanding of user requirements is vital

• With a small problem, the user requirements are usually obvious (e.g. Calculate the average of n integers).
• With a large problem, not all the users requirements will have been stated and some may be contradictory.
• Discussing the problem with a client will often lead to a change in requirements.
• It is very important that the major requirements are captured early on.
• Early feedback and prototype analysis with the client are crucial.

Adopting a Software Engineering methodology provides a solid framework that will enable you to solve large/complex problems, it describes

• the major stages to be undertaken and
guidelines on how to progress through them.

SE & Modelling

Software Engineering involves modelling a problem.

“\textit{A model is an abstract representation of a system that enables us to answer questions about the system.}” \textsuperscript{[Bruegge & Dutoit, p6]}

• Problems with many components are usually complex. 

Complexity can often be dealt with through simplification of a system into its main component parts.

• There are many different ways, different levels of abstraction, different assumptions on how to model a problem.

• No two people usually model a large problem in exactly the same way -- experience, expertise and creativity.

• Models allow us to visualise and prototype a system.

Building a model involves:

The Real World

• Environment within which the system must operate, e.g. trading rules within a stock-trading system.

The Problem Domain

• Concepts from the real-world that are relevant to the users requirements, e.g. stock components, traders.

The Solution Domain

• The components of the model, e.g. objects from the problem domain and objects that enable problem domain objects to interact, possibly according to rules of the problem domain.

SE & Problem Solving

Software Engineering is about Problem Solving:

• Formulate the problem.
• Analyse the problem.
• Search for solutions.
• Decide on appropriate solution.
• Specify the solution.

There is not always a standard algorithm to follow …

• You often have to try a number of paths (solutions) and then select the one that appears most appropriate.
• Previous experience is very useful in selecting a solution.
• You can also benefit from other peoples experience (more later).
SE & Knowledge Acquisition

As you design, or model, a system, you acquire more information:

- You will very rarely have all the facts at the start.
- New knowledge does not always add to prior knowledge.
- Sometimes newly acquired information may invalidate your previous ideas.
- You need to carefully manage the knowledge acquisition process to ensure you do not miss out crucial information.

SE principles and guidelines help avoid these pitfalls,

e.g. build flexibility into your design!

- Keep component dependencies to a minimum.
- If a major design revision is necessary due to new information, minimal dependencies will reduce the need to start again from scratch.

SE & Rationale Management - 1

The reason behind a given design decision is called the rationale of a system

- We need to know why a system does what it does and what parts of this rationale are relevant to the problem and the solution.
- When your problem concerns the addition of a new component (e.g. database access) to a previous system, it is important to understand the rationale behind the design of the system you are about to modify.
- If the rationale is not taken into account, your new component may conflict with the original aim of the system.
- A problem of documentation.

SE & Rationale Management - 2

Example: Y2K or the Millennium Bug

- 1960 - 1970's, efficient use of memory important therefore years were represented as 2 digits, i.e. '65' instead of 1965.

  Great idea (rationale) - reduce memory storage requirements.

- Assumption was that the program would only be used for a few years.

- Use of 2 digits for a year became standard practice even though memory use was no longer a problem (programmers forgot the original rationale behind the decision to truncate dates).

- The Result - A big headache at the turn of the century.

SE & Rationale Management - 3

It is not easy to specify the complete rationale behind a system design

- Developers make many decisions based on prior experience.
- They do not always state the rationale behind these design decisions in documentation.
- Good software engineering practice tries to manage this process.

  For example: always ensure that you state assumptions when making a design decision. The assumptions often encompass the rationale behind a design and can be a useful check list to discuss with a client.
SE & Rationale Management - 4

Issue: What issue needs to be addressed in order to proceed?
E.g., how should years be represented?

Alternatives: What are the alternative ways of addressing the issue?
E.g.,
   4 digits - complete but increases storage requirements.
   2 digits - compact, in 1970 the year 2000 seemed a long way off.

Criteria: Constraints that the solution should satisfy - not always consistent
E.g., memory efficient, future proof.

Decision: The resolution of an issue:
   pick the solution from the list of alternatives that best fits the criteria.