

CSC227

***Formal Specification
of Software***

**John Fitzgerald
Centre for Software Reliability**

Software Today: why we need to model systems

- Challenges in software development
- Modelling Computing Systems
- Formality
- Formal specification languages
- The structure & content of CSC227

Characteristics of Software

*We build computing systems out of software - engineers in other disciplines use physical materials like steel, electronic devices or advanced materials. **What makes software different?***

Software Today: challenges

Technological: you can do more in software than before

Software is often used for critical tasks.

Name some safety- or security-critical applications

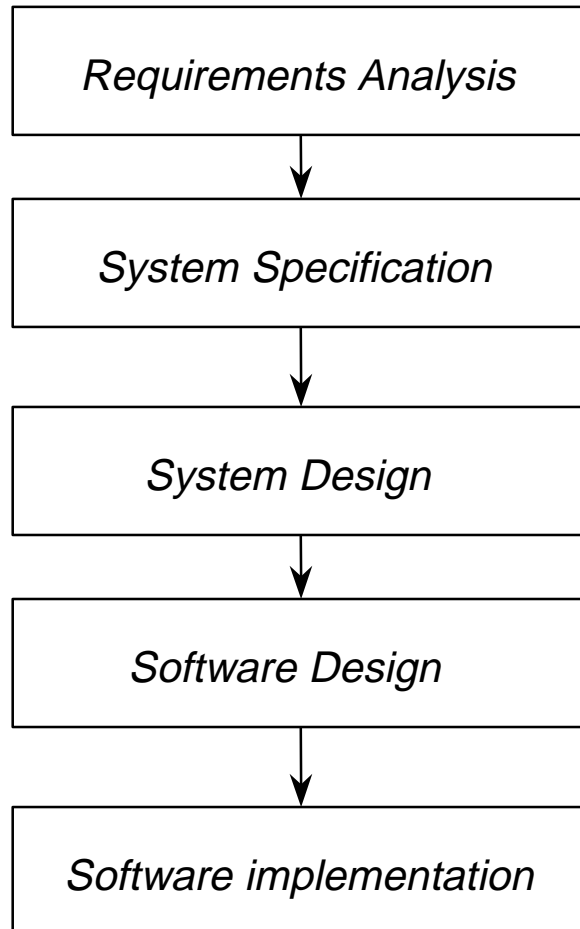
Software Today: challenges

Economic Challenges: the cost of rework

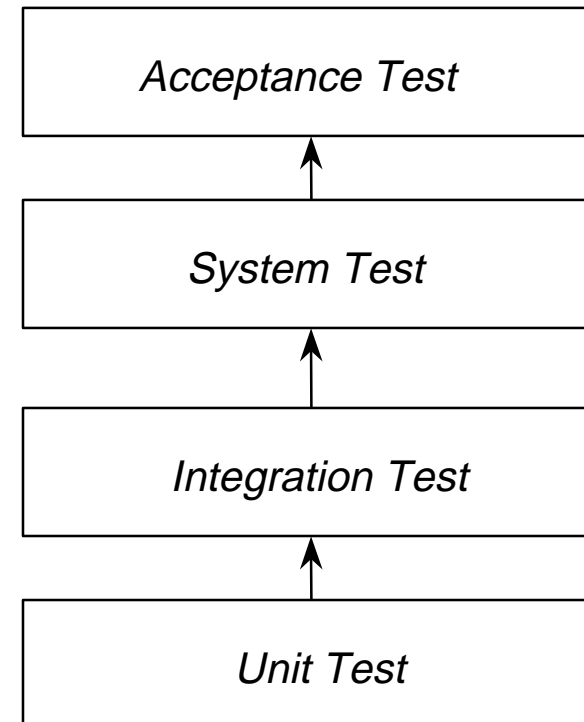
Software development takes place on a huge scale, and often goes wrong!

How much software gets used as delivered?

Software Today: challenges



Rework Costs



Software Today: challenges

Rework Costs

- The rework cost to fix a bug is related to “distance” between the commission and the discovery of the error.
- Improved analysis of requirements and designs could reduce the rework costs for some of the most expensive errors.
- CSC227 is about a particular class of techniques which help us to do this kind of analysis.

Modelling Computing Systems

In other engineering disciplines (Mechanical, Electrical, Aeronautical etc.) system models are built to help gain confidence in requirements and designs. For example:

In this course, we will look at how we can build and analyse models of software. There are two characteristics of these models which are crucial to their successful use: **abstraction** and **rigour**.

Modelling: abstraction

Engineering models omit details which are not relevant to the purpose of the model. For example:

The omission of detail not relevant to a model's purpose is called **abstraction**. The choice of which details to omit is a matter of engineering skill.

Modelling: abstraction

Compare these extracts from two descriptions of the same system.

The FlightFinder System is to be used by travel agents and their customers. Details are entered, including point of departure, destination, preferred dates and times. He system will respond with a range of itineraries and fares, along with the relevant restrictions.

The system record locations as nodes in a connected graph structure. Each node struct contains an array of pointers to reachable destinations plus, for each pointer, a timetable of flights stored as a hash table. Each record in the hash table has a flight number (8 character string), departure and arrival times (standard time formats) and operating dates (standard date format). To obtain the optimal route, the graph must be traversed using a shortest path algorithm on a modified adjacency matrix ...

Modelling: rigour

The most important property of a model of a computing system is its suitability for analysis. The analysis must be **objective** (not down to the opinion of the individual engineers performing it). It should also be **repeatable** and susceptible to **machine support**.

The language in which a model is expressed should be **rigorously defined**: little room for disagreement about what a model actually says; analysis tools reach the same conclusion about the properties of models.

Many programming language have non-rigorous definitions. What is the consequence?

Modelling computing systems

How do these concepts of system modelling transfer to software development?

A range of modelling techniques are used in software development:

Models constructed in early development stages are **specifications**; those developed in later stages are **designs**. We will generally be concerned with specifications (because of the importance of modelling in early development stages) but we will tend to use the term **model** to refer to the system descriptions that we develop.

Formality

This course concentrates on **formal** languages for expressing models.

A language is formal if its syntax rules and its semantics (the meaning of every construct in the language) are so precisely defined that there is no room for disagreement about the meaning of a model. Models expressed in a formal language are susceptible to a wide range of analysis techniques including mathematical proof (we can, in principle, prove that a model embodies a property such as safety or indeed prove that a program is correct with respect to a specification).

Formal Specification Languages

A **formal specification language** is a formal language used for expressing models of computing systems. Such languages typically provide support for abstraction and rigour.

General Purpose

VDM-SL

Z

RSL

Act One

Clear

Special Purpose

CCS

CSP

Real-Time Logic

Deontic Logics

Formal Specification Languages: VDM-SL

- Vienna Development Method (VDM)
- Spec Language is VDM-SL
- ISO Standardised: fully formal
- Support Tools are available
- Good record of industrial use
- Support for abstraction of data and functionality

Course Structure

- *Introduction*
- *Guided Tour through a formal model*
- *Logic*
- *Basic abstractions*
- *Principal abstractions: sets, sequences, mappings*
- *State-based and functional styles*
- *Validation*

Course Principle

- ***Formal Methods are part of practical Systems Engineering, not theoretical Computing Science!***
- *All our examples are based on real formal models developed in a commercial context.*
- *We will use examples and exercises extensively to give a strong flavour of the modelling skills used.*

CSC227 Arrangements

- **Lectures:**
- **Coursework:**
 - An individual exercise book (60%)
 - A group-based modelling problem (40%)
- **Text:** *Fitzgerald & Larsen, "Modelling Systems: Practical Tools and Techniques in Software Development", Cambridge Univ. Press 1998, ISBN 0-521-62348-0*
- **Web:** <http://www.csr.ncl.ac.uk/modelling-book/>