Managing Change and Complexity

The reality of software development

THE UNIVERSITY of York

Chris Kimble
January 2008

Overview

- Some more Philosophy
  - Reality, representations and descriptions
- Some more history
  - Managing complexity
  - Managing change
- Some more conclusions
  - A preview of what is to come
  - An introduction to systems development methodologies

THE UNIVERSITY of York

Chris Kimble
January 2008

Review

PENGUINS ARE BLACK AND WHITE.
SOME OLD TV SHOWS ARE BLACK AND WHITE.
THEREFORE, SOME PENGUINS ARE OLD TV SHOWS.
GLASBERGEN

Logic: another thing that penguins aren't very good at.

THE UNIVERSITY *of York*

Chris Kimble
January 2008

Review

Last week …

- Philosophy
  - Descriptions of complete and closed systems
  - Notions of equivalence

- Programs and software
  - Programs
    - A description of the sequence of operations a machine can perform automatically
    - Closed and complete
  - Software
    - A description of the relationships between human communities, their environment and the machines associated with them
    - Almost never closed and complete

THE UNIVERSITY *of York*

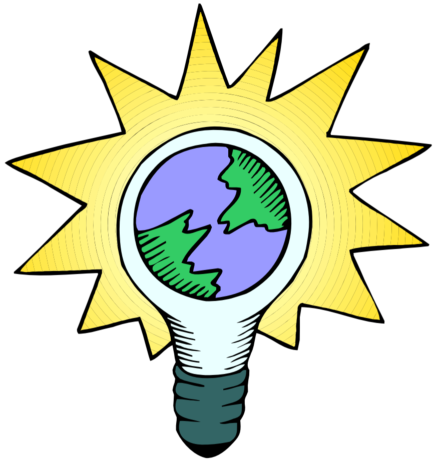
More Philosophy

- Previously we used the term “description” but what does this mean and what is being described?
- To answer this we need to look at three important concepts: reality, representations and descriptions
- As a starting point we will, arbitrarily, define these three concepts by reference to the Oxford English dictionary (Oxford University Press, 2nd edition, 1989).

Reality

- Reality is "... that which underlies and is the truth of appearances or phenomena." (OED)
- It is 'unperceived reality': what actually exists, rather than what is perceived by an individual.
- Reality is "out there" and exists whether we know about it or not.

Representation

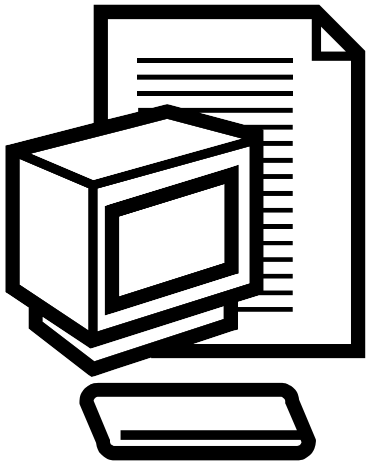


- A representation is "... the operation of the mind in forming a clear image or concept.” (OED)
- It is an *internal* 'perceived reality': the mental model an individual uses to comprehend reality.
- A representation only exists in the mind of the individual and can not be shared directly.

THE UNIVERSITY *of York*

Chris Kimble
January 2008

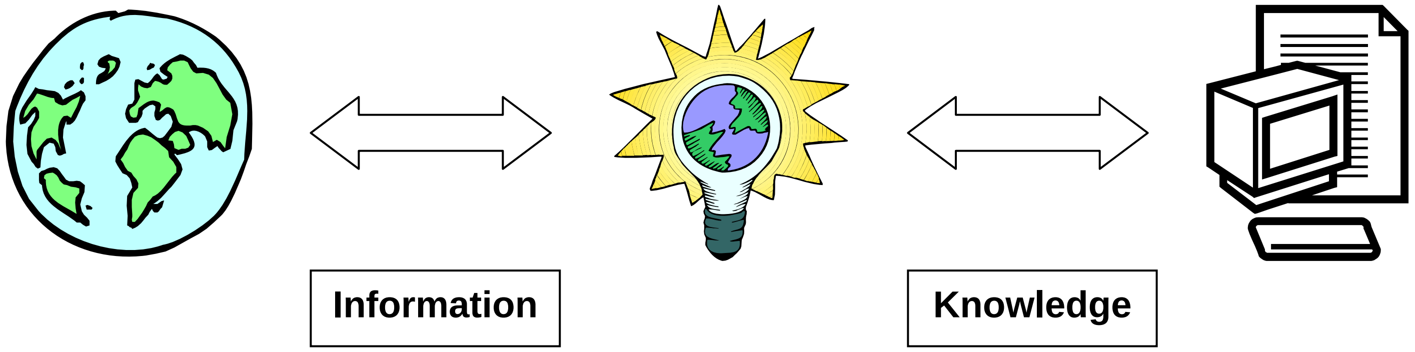
Description



- A description is "... a statement which describes, sets forth, or portrays''. (OED)
- It is an *externalisation* of an individual's internal model of reality made in order to communicate it to others.
- A description exists independently of an individual and can be shared between a number of individuals.

THE UNIVERSITY *of York*

Putting them all together



- The representation is created as the result of the reception of stimuli from reality.
- We refer to this relationship as information.
- In philosophy, the properties of the information relationship is the concern of *Ontology*.

- The description is knowingly created in order to communicate their thoughts to others.
- We refer to this relationship as knowledge.
- In philosophy, the properties of the knowledge relationship is the concern of *Epistemology*.

Chris Kimble
January 2008

More history

- 1950's
  - High level languages are still new and poorly understood but progress is being made
  - Remaining task is how to *characterise* the problem the computer needs to solve
- 1960's
  - Too many problems characterising the user domain, solve them one at a time
  - Develop an ad hoc approach

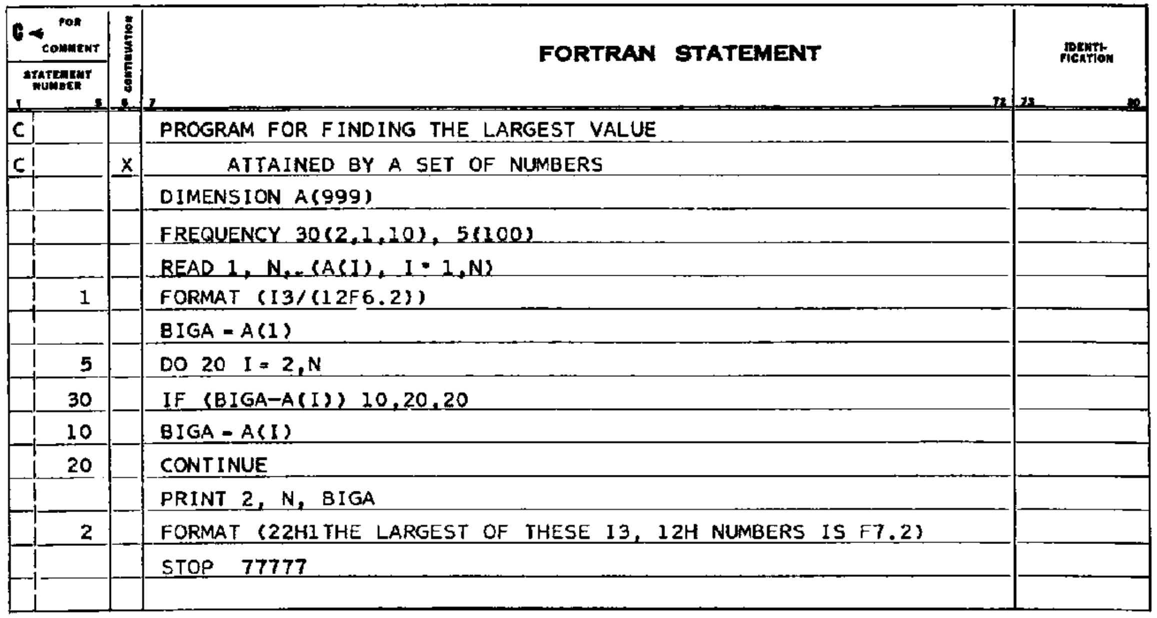
Some history

- mid 1970's
  - Growing computer use leads to increasing complex applications, a problem that the ad hoc the approach can no longer deal with
  - Look for ways to break down tasks and take a *structured* approach
- 1990's
  - Growing pace of change means that structured approaches can no longer keep up
  - Look for hierarchies and take an *object oriented* approach

Very early languages

- FORTRAN I (FORmula TRANslator) was first developed as a scientific language in 1957. It was designed for accuracy and could perform a single repetitive task using a simple instruction set and loops
  - Designed to run on one machine: The IBM 704 EDPM (Electronic Data Processing Machine) as people expected each computer to have its own language and application
  - FORTRAN II to IV followed, then FORTRAN 66, 77 and 90
  - FORTRAN 77 is still used today to re-compile legacy code.

FORTRAN



C ← FOR COMMENT	STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT	IDENTIFICATION
1	5	6	7 72	73 80
C			PROGRAM FOR FINDING THE LARGEST VALUE	
C		X	ATTAINED BY A SET OF NUMBERS	
			DIMENSION A(999)	
			FREQUENCY 30(2,1,10), 5(100)	
			READ 1, N, (A(I), I = 1,N)	
	1		FORMAT (I3/(12F6.2))	
			BIGA = A(1)	
	5		DO 20 I = 2,N	
	30		IF (BIGA-A(I)) 10,20,20	
	10		BIGA = A(I)	
	20		CONTINUE	
			PRINT 2, N, BIGA	
	2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
			STOP 77777	

Very early languages

- COBOL (Common Business Oriented Language) was first developed as a business language in 1959. It was designed for easy readability and machine independence
  - COBOL statements have a simple "natural language" like grammar and are arranged in "modules"
  - Its only data types were numbers and strings which could be put into arrays and records so that data could be sorted
  - COBOL 68, 74 and 85 followed. Y2k problem lead to a resurgence of interest in COBOL and a Tiny COBOL (for linux) is now available

THE UNIVERSITY *of York*

“The COBOL Team”

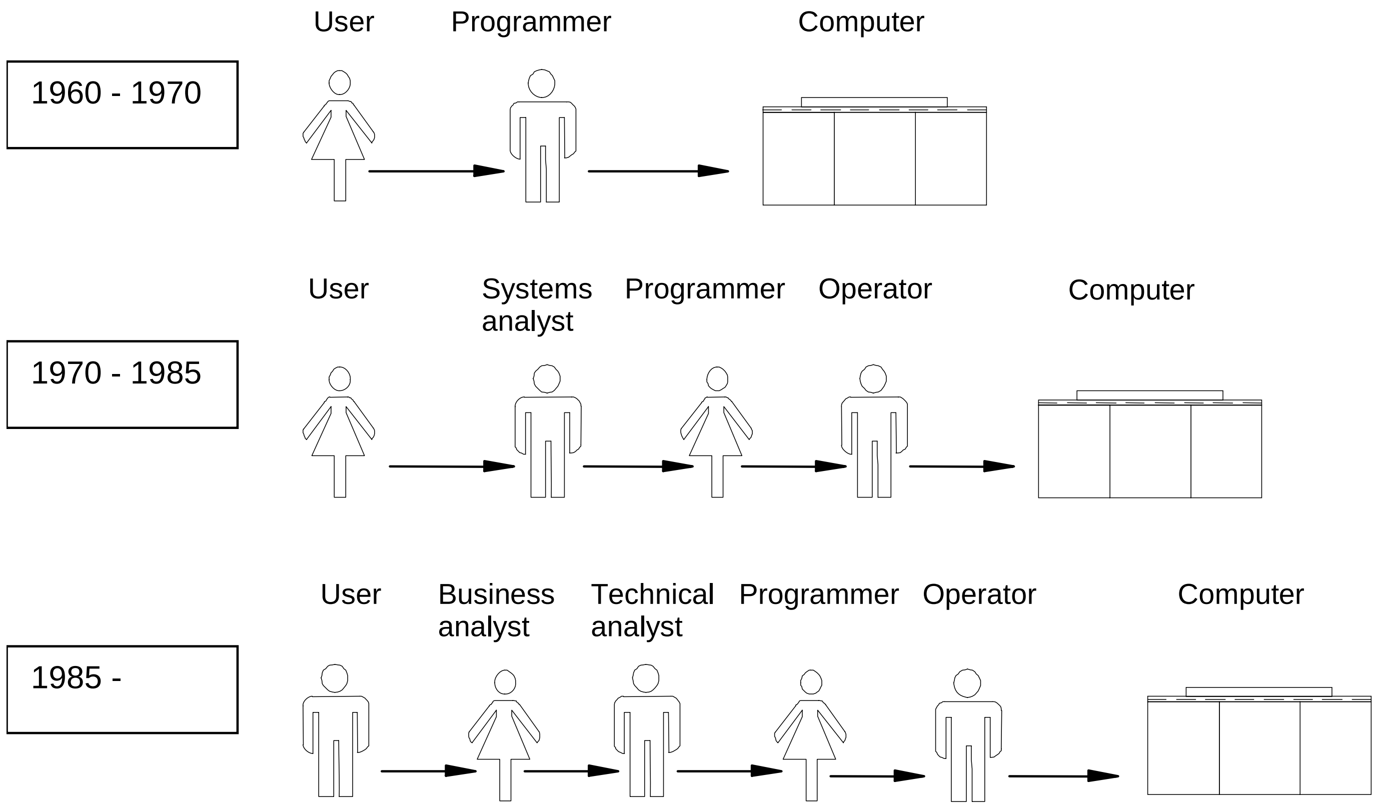
THE UNIVERSITY *of York*

Chris Kimble
January 2008

Early Methods

- Based on program design methods
- Top down structured design based on decomposition
  - The stepwise elaboration from an abstract conceptual model to implementation level details (early 70's)
  - The decomposition of a complex problem into simpler sub problems (early 70's)
  - Data flow diagrams (mid 70's)
  - Structured analysis (mid 70's)
- Leads to the growth of the systems analyst as an intermediary between programmer and user

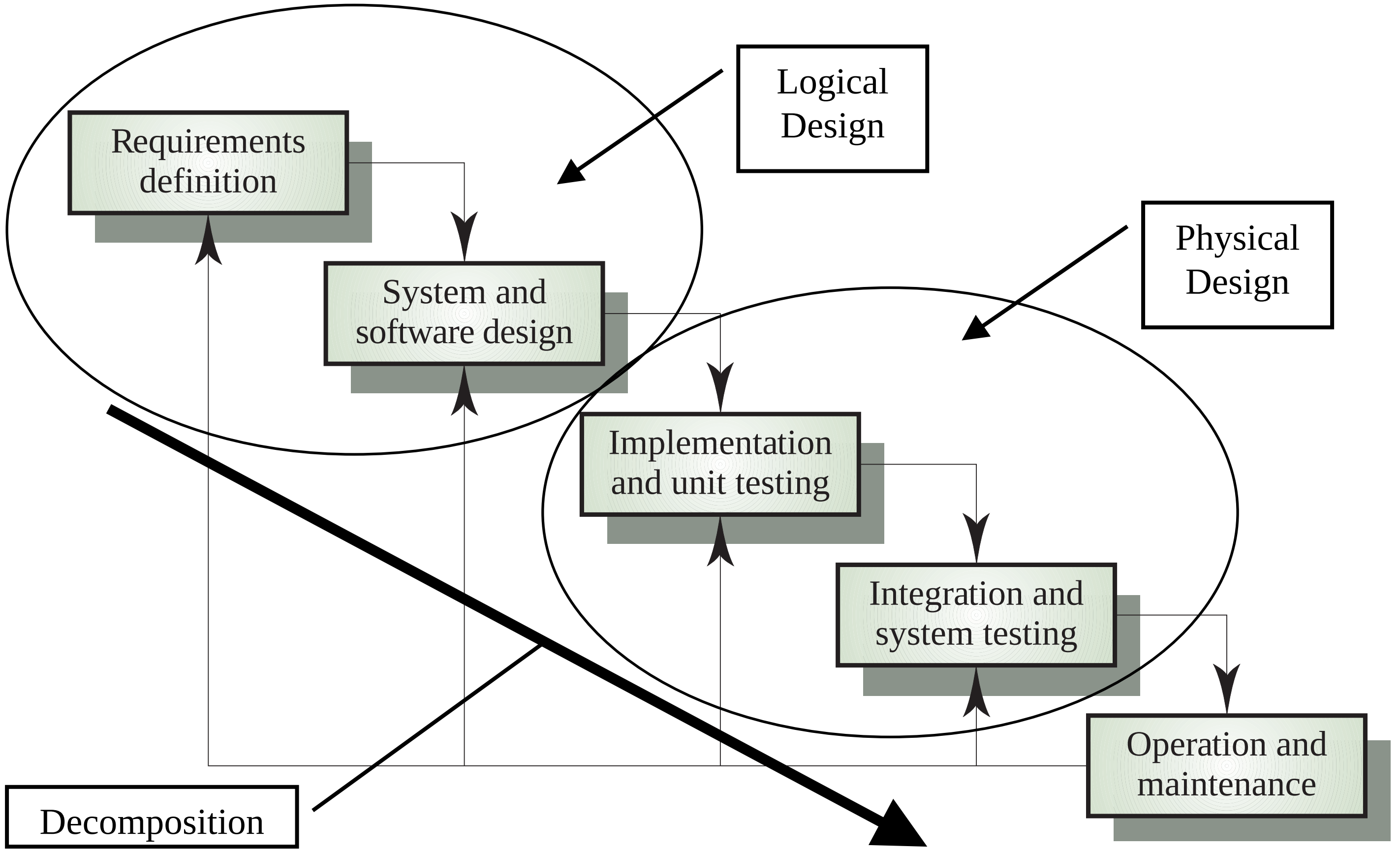
Analysts


User
Programmer
Computer
1960 - 1970
User
Systems analyst
Programmer
Operator
Computer
1970 - 1985
User
Business analyst
Technical analyst
Programmer
Operator
Computer
1985 -

The Systems Development Life Cycle

- The System Development Life Cycle (SDLC) is the classic example of the underlying *process* used to develop the software in the 1970's
- The concept was introduced by Royce in 1970 as an iterative approach to software development
- It:
  - Separates logical and physical design
  - Separates different functions and activities
  - Provides an *abstract description* of the development process

The ‘Classic’ Waterfall model


Logical Design
Requirements definition
System and software design
Physical Design
Implementation and unit testing
Integration and system testing
Operation and maintenance
Decomposition

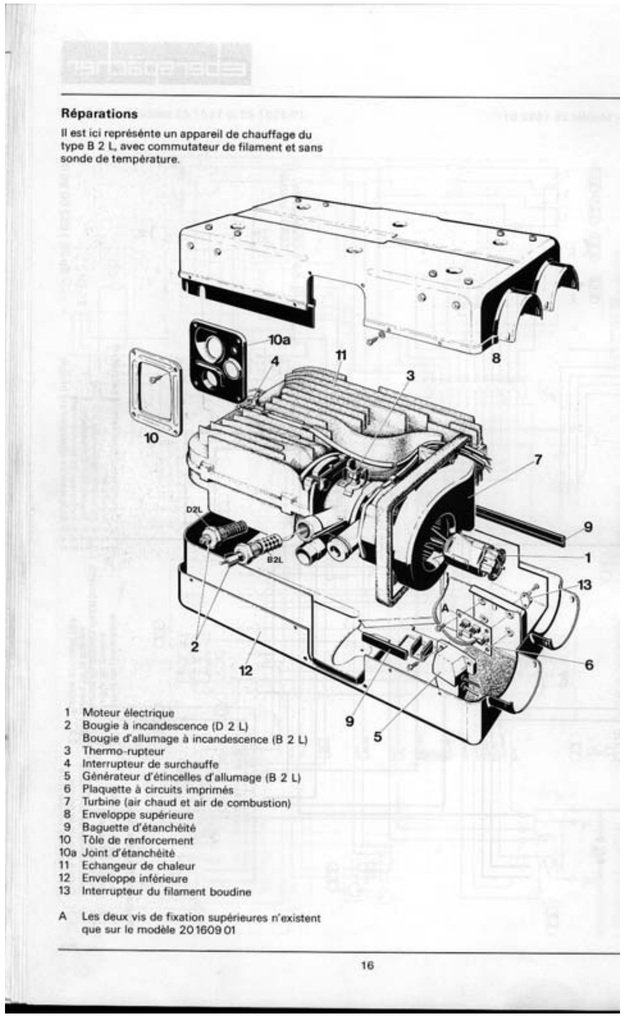
The Systems Development Life Cycle

- There are now many different variations of the basic model:
  - Specification, Design, Validation, Evolution
  - Feasibility Study, Analysis, Design, Implementation, Testing, User Guide, Evaluation
  - Requirements Specification, Design Stage, Coding and Construction, Testing, Installation, Maintenance
  - Preliminary Investigation, Systems Analysis, Systems Design, Systems Development, Systems Implementation, Systems Maintenance
  - Problem/needs identification, Feasibility Study, Systems Analysis, Systems Design, Implementation, Review and Maintenance, De-commissioning
  - etc, etc …

Structured Analysis and Design

- Some other approaches
  - Jackson Structured Programming uses Data Structure Diagrams to model inputs and outputs which are then used to structure the design of the software
  - Structured Analysis uses dataflow diagrams (DFDs) to represent the logical “flow” of data between processes. These are then used to structure the design of the software
  - Structured Design uses structure charts to show the decomposition of functions and their allocation to “units” which are then used to structure the activities associated with writing the software

Exercise



Réparations

Il est ici représénte un appareil de chauffage du type B 2 L, avec commutateur de filament et sans sonde de température.

1	Moteur électrique
2	Bougie à incandescence (D 2 L) Bougie d'allumage à incandescence (B 2 L)
3	Thermo-rupteur
4	Interrupteur de surchauffe
5	Générateur d'étincelles d'allumage (B 2 L)
6	Plaquette à circuits imprimés
7	Turbine (air chaud et air de combustion)
8	Enveloppe supérieure
9	Baguette d'étanchéité
10	Tôle de renforcement
10a	Joint d'étanchéité
11	Echangeur de chaleur
12	Enveloppe inférieure
13	Interrupteur du filament boudine
A	Les deux vis de fixation supérieures n'existent que sur le modèle 20 1609 01

16

- All of the above methods make similar assumptions about the relationship between:

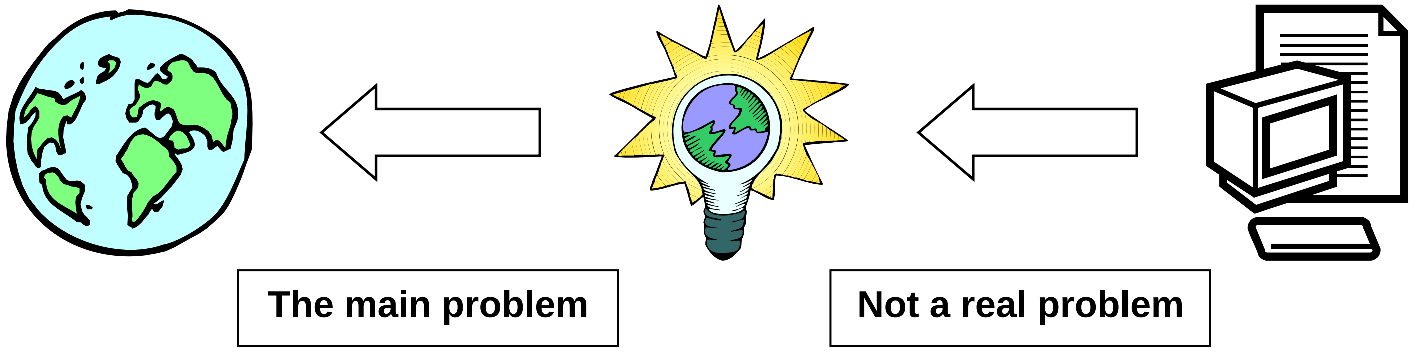
1. Reality and its representation in the head of the designer
2. The designer's idea of reality and its software description

- What are they?

THE UNIVERSITY *of York*

Chris Kimble
January 2008

Answer?



- The key problem was with the *validation* of software: are we building the right product? The emphasis was on finding methods to create *closed* systems so that designers could deal with the growing complexity of applications
- Although there were still a number of problems with writing programs, the development of new languages that could be verified formally lead to the issue of writing the program the designer intended to be seen as a minor issue

Problems with the early languages

- Difficult to understand
  - Use specialist notation and emphasis on logic does not have a counterpart in everyday life
- Difficult to maintain, modify or adapt
  - In addition to the above, the links between different parts of a program are not clear (if there is no documentation)
- Difficult to reuse
  - Routines and procedures are produced for one application and are not easily identified or ported to other applications
- etc, etc

Problems with the early methods

- Only appropriate when the requirements are well-understood (an assumption of completeness?)
- The partitioning of the design into distinct stages was inflexible
- It was difficult to respond to changing customer requirements
- They became too large and cumbersome and added to complexity rather than reducing it
- They provide procedures that become an excuse for not thinking about the problem
- etc, etc

THE UNIVERSITY *of York*

Later languages

- Object Orientated (OO) languages are said to have become popular because of the problems with the earlier functional / procedural languages
- OO is an approach to the modelling of systems as a set of objects, and the relationships between between objects them as associations
- OO languages differ radically from their predecessors as the concentrate on the object (the world *outside* the machine) rather than the instructions to the computer (the world *inside* the machine)

Later languages

- It can be argued that object oriented languages first came into existence in 1962 with the creation of SIMULA I, or in 1967 with SIMULA 67, or in 1971 with SMALLTALK 71, or …
- Object Orientated languages began to gain commercial dominance in the the 1980’s
- First language was when C++ and later, in the 1990’s, Java cemented the place of object orientated languages in today's commercial applications

Later methods

- Object orientated methods are based on observation of reality
- However, reality is unimaginably large and complex, thus the designer has to simplify this by looking for commonality:
  - Look for classes of similar or related objects based on certain features known to be of interest
  - Look for hierarchies where the properties from one object are passed on to another
- These principles of Objects, Abstraction, Classification and Inheritance, form the basis of what is now called the Object Orientated approach

Object Oriented approaches

- Object-oriented analysis (OOA) is concerned with developing software requirements and specifications that are expressed as an object model.

- Object-oriented design (OOD) is concerned with developing an object-oriented model of a software system to implement the identified requirements.

THE UNIVERSITY *of York*

Object Oriented approaches

- Unified Modelling Language (UML) is an attempt to develop a single approach to OO analysis and design
- It is a graphical language, or a notation, for modelling system analysis and design concepts in an object-oriented fashion
- It provides a set of rules and semantics that can be used to specify the structure and logic of a system

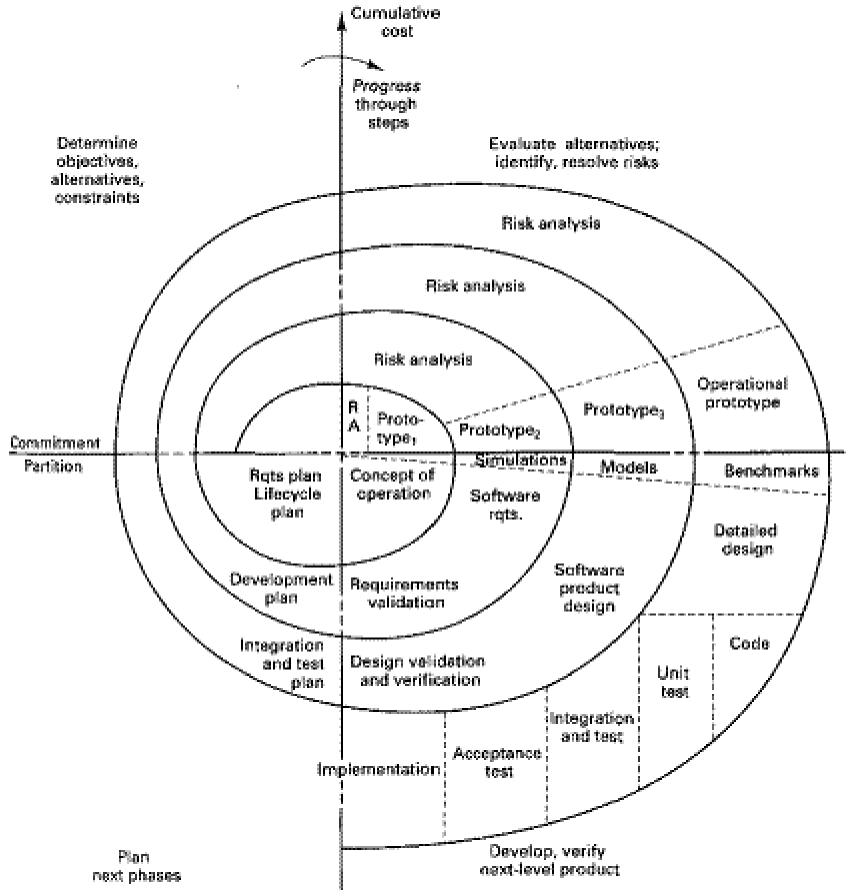
Object Oriented approaches

- The Rational Unified Process (RUP) provides a flexible framework that can be used to describe specific development processes
- the essence of RUP is *iteration*; there are four phases in the software cycle
  - Inception - do you and the customer have the same understanding of the system?
  - Elaboration – can you actually build the system?
  - Construction - are you developing the product?
  - Transition - are you trying to get the customer to take ownership of the system?

Return to the SDLC

- The 'Classic' System Development Life Cycle was a model of the process used to develop the software in the 1970's
- The more iterative / evolutionary approach of OO led to attempts to rethink of the 'Classic' SDLC
- The Boehm spiral model (1988) is the most notable and is based on the idea of a series of incremental releases.
  - The development 'spirals' outward from the centre, with each cycle of the spiral leading to a further refinement of the system.

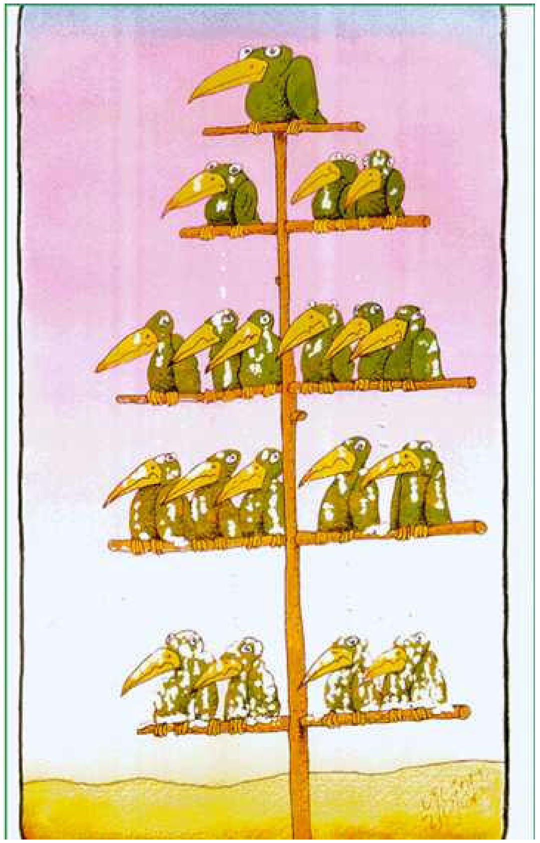
The Spiral Model



THE UNIVERSITY *of York*

Chris Kimble
January 2008

Exercise

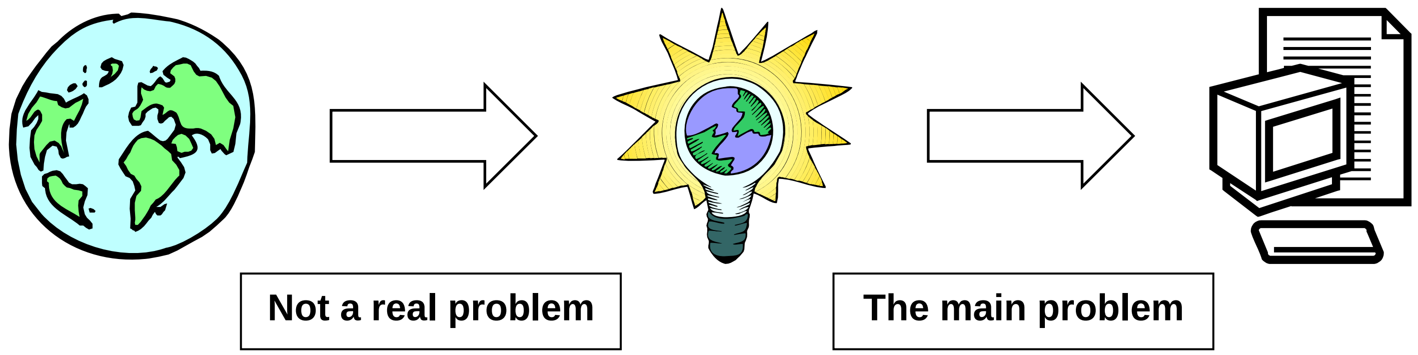


- All of the above methods make similar assumptions about the relationship between:

1. Reality and its representation in the head of the designer
2. The designer's idea of reality and its software description

- What are they?

Answer?



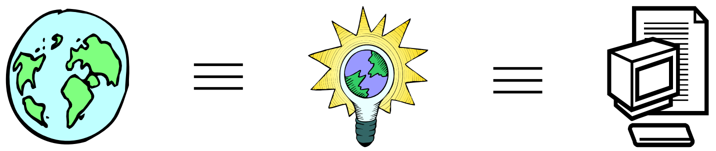
- Although there are a number of problems with interpreting reality, by basing OO approaches on observed reality the proponents of OO can make a strong claim to have valid descriptions
- The key problem is with the *verification* of the software description. The language, provides everything needed to *describe* a problem, but it is difficult to demonstrate that the design conforms to the requirement

Review

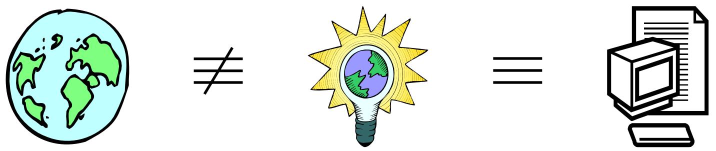
- Programs
  - All relationships are closed and complete, a special case
- Early methods
  - Relationship between description and reality is assumed to be complete but not closed (i.e. needs validation)
- Later methods
  - Relationship between description and representation is assumed to be closed but not complete (i.e. needs verification)
- Other approaches
  - Open and incomplete?

A Preview

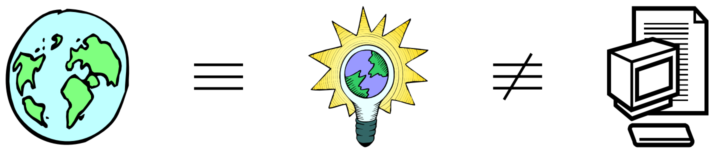
- Using notions of:
  - Reality, representation and description
  - closure and completeness
- We now (potentially) have four ways to think about the problem



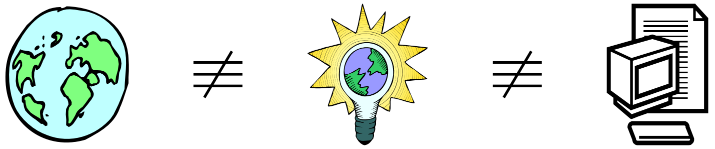
Programs



Old approaches



New approaches



Others?

THE UNIVERSITY *of York*

A Preview

- The story so far has been a general introduction based on the history of the term software and how notions of software development have changed over time
- Later we will look more closely at the philosophical underpinnings of each position and also look at some specific examples of each approach
- But next, we need to ask “what do we mean by a methodology”

Methodologies

- In philosophy a methodology is:
  – The analysis of the methods appropriate for a particular field of study or branch of knowledge.
- In Information systems and software design it is:
  – A collection of procedures, techniques, tools and documentation which will help developers in their efforts to design and implement a new information system.
  – A collection of phases and sub-phases that guide developers toward appropriate techniques for each stage of the project and help them plan, manage, control and evaluate it.
  – A selection of tools techniques and methods unified by a common *philosophy*.

Why have a software design methodology?

1. A better end product
   – People may want a methodology to improve the end product of the development process, that is, they want better information systems.
2. A better development process
   – The benefits of controlling the development process and identifying the outputs: improved management and enhanced productivity.
3. A more standardised process
   – more integrated systems, staff can move from project to project without retraining and easier maintenance of systems.

Why a philosophy?

- Which approach gives the best solution?
  - A system which makes most use of computers is a good solution
  - A system which has accurate documentation is a good solution
  - A system which is the cheapest to run is a good solution
  - A system which is most quickly implemented is a good solution
  - A system which is the most adaptable is a good solution
  - A system that is liked by the users is a good solution.
- What assumptions do we make when we choose a particular approach and are they appropriate?

Next week

- Be prepared …

  to review all of the material we have covered so far
  - Lectures
  - Notes
  - Reading
  - etc