System Design Methodologies

Back to Basics: Programs and Software

THE UNIVERSITY *of York*

Chris Kimble
January 2008

Overview

- Some philosophy (or is it mathematics?)
  - Complete and Closed Systems, Equivalence and Descriptions
- Some history (or is it mathematics again?)
  - Programs and Formal Languages
  - Software and Formal Languages
- Some conclusions (i.e. the basics)
  - Validation and Verification
  - The difference between Programs and Software

THE UNIVERSITY *of York*

Chris Kimble
January 2008

A puzzle

- A proposition is an idea with which the label “true” or “false” can be associated.

- Consider the following proposition:
  - Fido is a dog
  - Dogs are animals
  - Therefore, Fido is an animal

- Under what conditions it this proposition true?

Answer

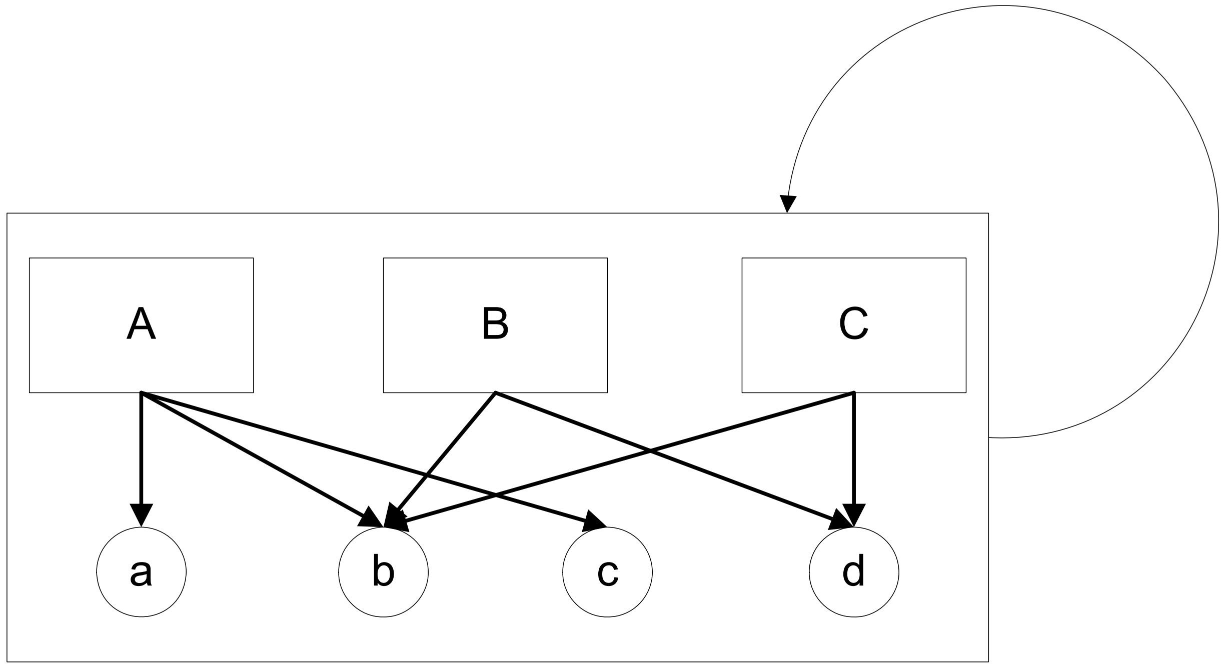
	1	2	3	4
Fido is always a dog	True	True	False	False
All dogs are animals	True	False	True	False
Fido is an animal	True	False	False	False

Closure

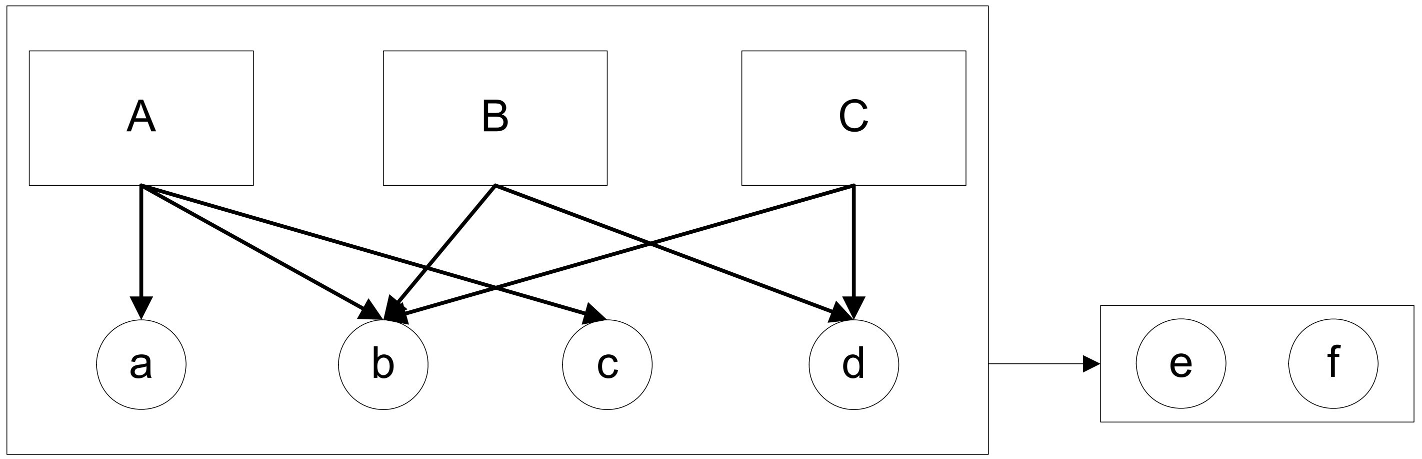
- In set theory, a description is said to be closed if the result of the operation on any member(s) of the set is also a member of the set

- Closed systems are sometimes called self-describing systems because they not only describe a truth, but also describe what truth is and how it can be recognised.

A Closed Description



An Open Description



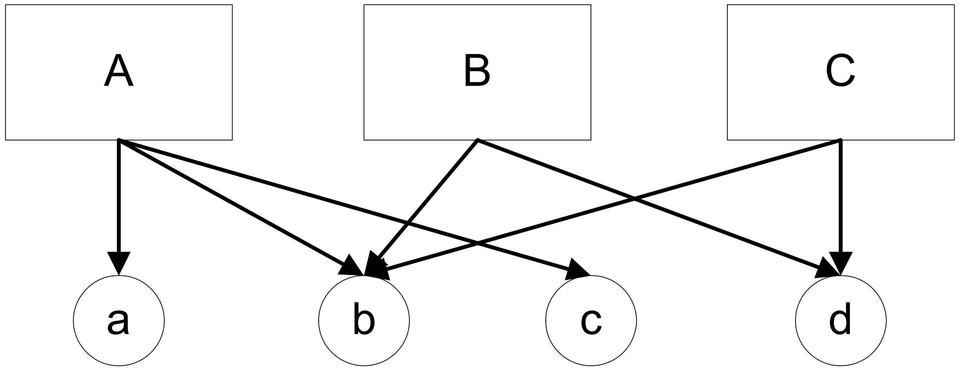
Completeness

- In logic, a description is said to be complete if all of its propositions can be derived only from the axioms of the system.

- Complete systems allow the truth of any statement to be verified by recourse to the basic axioms of the system

THE UNIVERSITY of York

A Complete Description



$$\begin{tikzcd}
\boxed{A} \arrow[d] \arrow[drr] \arrow[drrrr] & & & \boxed{B} \arrow[dl] \arrow[drrr] & & & \boxed{C} \arrow[dllll] \arrow[d] \\
a & & b & & c & & d
\end{tikzcd}$$

An Incomplete Description

$$\begin{tikzcd}
A \arrow[d] \arrow[dr] \arrow[drr] & B \arrow[d] \arrow[drr] & & C \arrow[d] \arrow[dll] & & \\
a & b & c & d & e & f
\end{tikzcd}$$

THE UNIVERSITY *of York*

Chris Kimble
January 2008

Putting them all together

Complete and Closed	Complete and Open
Incomplete and Closed	Incomplete and Open

Complete and closed

- Both the scope of the system and all of the possible behaviours that could occur inside the system are known.
- If any new behaviour is found, or if the boundaries of the system change, the truth of the whole description becomes invalid.
  - If we can demonstrate that the propositions “Fido is a dog” and “dogs are animals” are both closed and complete then it follows that the two terms are *equivalent*, i.e. you could substitute “animal” for “Fido” in any proposition and it would still be true.

Complete and open

- The behaviour inside the system is fixed but the boundaries of the system are not.
- The boundaries of the system are open to change, but if any new behaviours are discovered, the truth of the whole description becomes invalid.
  - For example, if “Fido is a dog” were complete, but “dogs are animals” was open (e.g because there are other animals than dogs) then the terms “animal” for “Fido” are no longer equivalent. Consequently, if you wanted to build something for a Camel, you could not use a dog as a substitute.

Incomplete and closed

- The boundaries of the system are fixed, but the behaviour inside the boundaries is not fully known.
- New behaviours can be found in the system, but if the boundaries of the system change, the truth of the whole description becomes invalid.
  - For example, if "Fido is a dog" were incomplete (e.g. because, when there is a full moon, Fido becomes a Hell Hound) but "dogs are animals" is closed then the terms "animal" for "Fido" are not equivalent because we don't know if a "Hell Hound" is an animal.

Incomplete and open

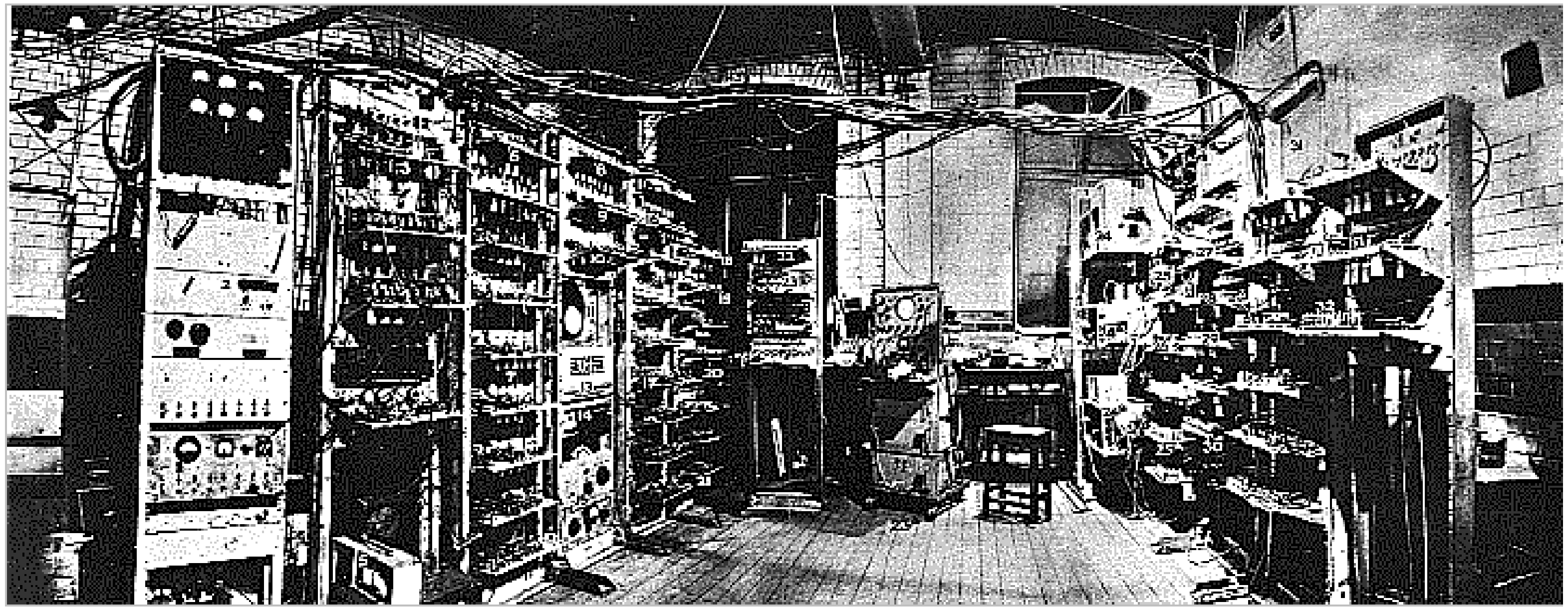
- Neither the scope of the system, nor all of the possible behaviours that could occur inside the system are known.
- Both the boundaries of the system and the behaviour of system can change without challenging the validity of the description.
  - Clearly, if we cannot demonstrate that the propositions “Fido is a dog” and “dogs are animals” are either closed or complete then the two terms cannot possibly be equivalent.

Programs

- Program - “a sequence of operations that a machine can be set to perform automatically”

- ‘Programs’ are not new
  - Punch Cards
    - Jacquard Loom (programmable weaving machine) - 1804
  - The difference engine
    - Babbage / Lovelace (a design for a calculating engine) - 1833

Early Computers (Manchester mark 1, 1948)



THE UNIVERSITY *of York*

Chris Kimble
January 2008

Early Computers

- Early computers required the operator to change its physical layout of the machine in order to change the program
- “Reprogramming” the computer was a long drawn out a manual process:
  - paper notes
  - flow charts
  - detailed engineering designs
  - re-wiring
  - testing
  - …

THE UNIVERSITY *of York*

Re-programming the Mark 1

THE UNIVERSITY *of York*

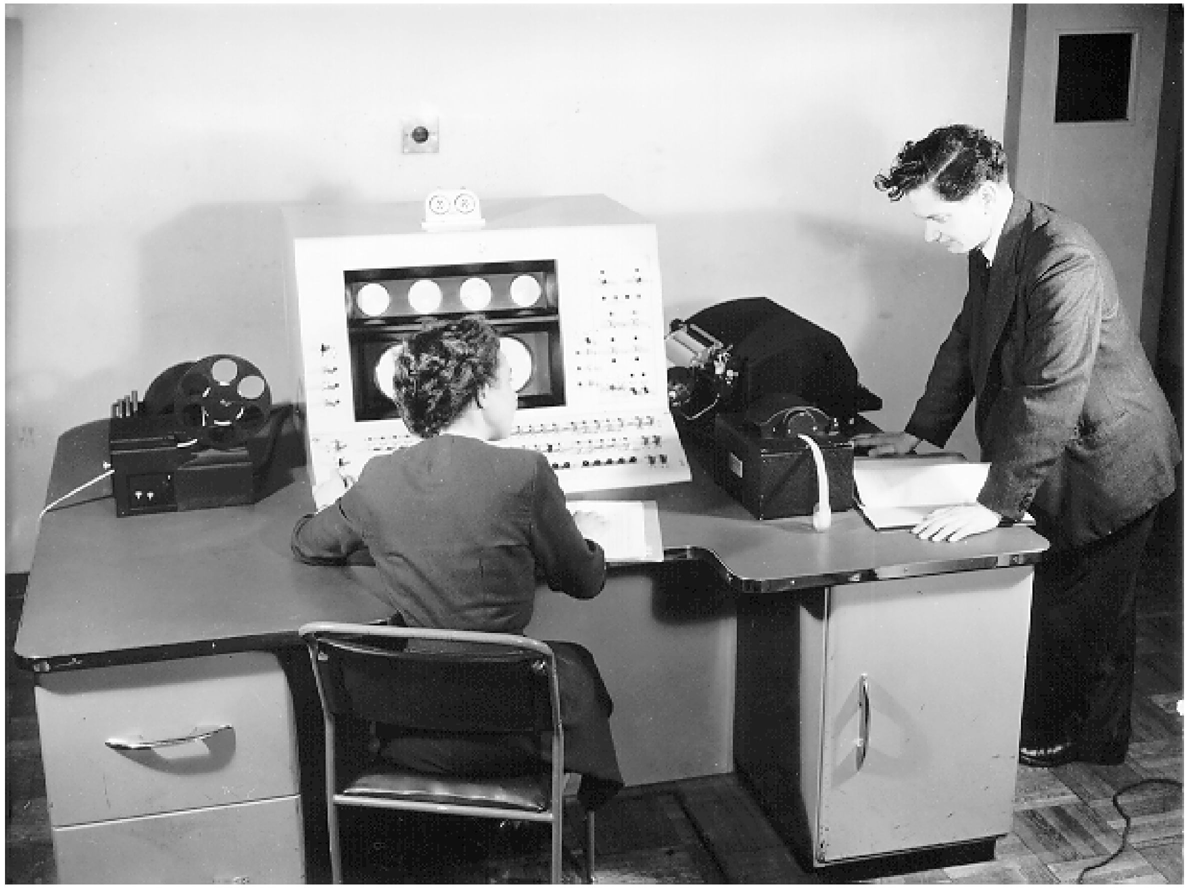
Chris Kimble
January 2008

Programs

- The von Neumann architecture or von Neumann machine
  - a design for a computer that uses a single storage structure to hold both instructions and data
  - Now a language is used to *describe* the operations to the computer will carry out
  - There is no need to change the physical layout of the machine
  - Computers become more flexible …

THE UNIVERSITY *of York*

Early computers (Ferranti Mark 1, 1951)



THE UNIVERSITY of York

Chris Kimble
January 2008

Programs

- Von Neumann did not use natural language but borrowed from the mathematical theory developed by Alan Turing, Emile Post, Alonzo Church in 1936.
- The languages proposed by these new theories relied heavily on formal logic, consequently the languages themselves became known as formal languages.

Formal Languages

- Contain a finite number of symbols and rules for combining symbols
- Atomic symbols can not be decomposed into other symbols and form the alphabet of the language
- Non-atomic symbols are created by combining atomic symbols
- Non-atomic symbols must have a set of rules that describe precisely how they are created
- Atomic symbols, non-atomic symbols and the rules combining them are the grammar of the language

Formal Languages

- Some basic assumptions of descriptions using formal languages:
  - The language is complete, i.e. no new symbols or operations can be introduced
  - The system they describe is closed, i.e. the system can not behave in such a way that the behaviour exhibited can not be described by the language

Equivalence and Description

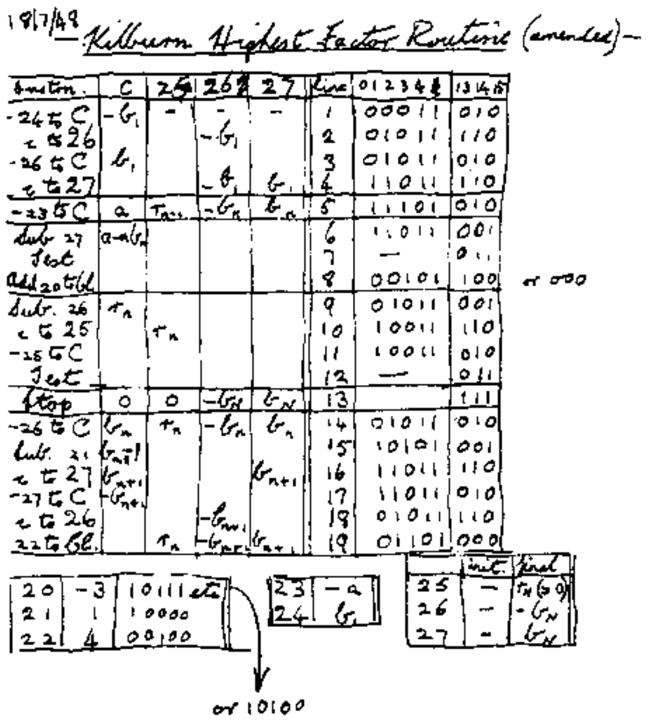
For complete and close systems:

- Given a written *description* (program), it is possible to work out exactly what sequence of actions the machine will produce
- Given a sequence of actions by the machine, it is possible to work out exactly what the written *description* (program) must have been

- Equivalence
  - The actions of the machine and the *description* are *equivalent* (actions ≡ instructions)

THE UNIVERSITY *of York*

Exercise



18/7/48 — Kilburn Highest Factor Routine (amended) —

Instrn.	C	25	26	27	Line	01234	13 14 15
-24 to C	$-b_1$	–	–	–	1	00011	010
c to 26			$-b_1$		2	01011	110
-26 to C	b_1				3	01011	010
c to 27			$-b_1$	b_1	4	11011	110
-23 to C	a	r_{n-1}	$-b_n$	b_n	5	11101	010
Sub 27	$a-nb_n$				6	11011	001
Test					7	—	011
Add 20 to Cl					8	00101	100
Sub. 26	r_n				9	01011	001
c to 25		r_n			10	10011	110
-25 to C					11	10011	010
Test					12	—	011
Stop	0	0	$-b_N$	b_N	13		111
-26 to C	b_n	r_n	$-b_n$	b_n	14	01011	010
Sub. 21	b_{n+1}				15	10101	001
c to 27	b_{n+1}			b_{n+1}	16	11011	110
-27 to C	$-b_{n+1}$				17	11011	010
c to 26			$-b_{n+1}$		18	01011	110
22 to Cl.		r_n	$-b_{n+1}$	b_{n+1}	19	01101	000

or 000

20	-3	10111 etc
21	1	10000
22	4	00100

or 10100

23	-a
24	b_1

	init.	final
25	–	r_N (= 0)
26	–	$-b_N$
27	–	b_N

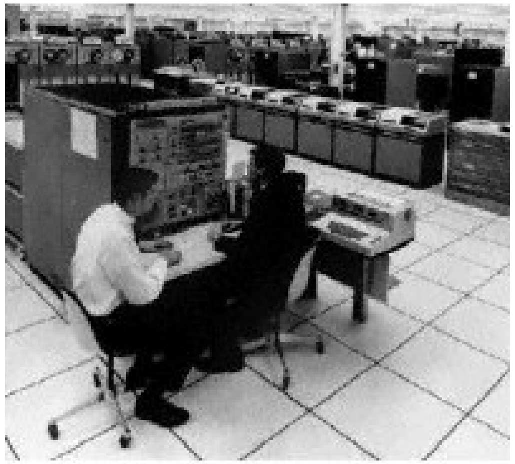
- This is a description of a procedure to find the highest factor of a number
- It was written on 18 June 1948 and run on the Manchester Mark 1 on 21 June 1948
- Is this a program?

THE UNIVERSITY *of York*

Chris Kimble
January 2008

Software

- Before the 1960s separate notions of ‘programs’ and ‘software’ did not really exist, but as computer use became more widespread the need for separate terms became apparent
  - In 1962 IBM launched the System/360 (a family of six computers with up to 40 compatible peripherals)
  - Orders for the system reached 1,000 per month within two years



Software

- A mismatch between the requirements of the machine and the needs of the business
  - Example: a payroll system for a business

Machine

Sort the output by employee identity code, following the physical layout of the files storing the payroll data to minimise the time spent moving data to and from the files

Business

Sort the output by employee name, following the way the accounts departments processes are arranged to minimise disruption to the smooth running of the business

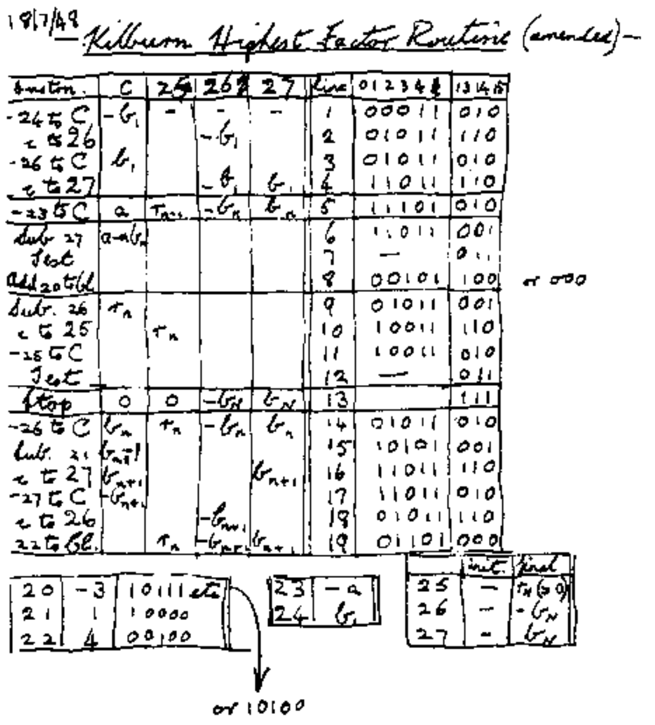
Software

- There can be a mismatch between the needs of the machine and the needs of the business
- The actions of the machine and the instructions are no longer equivalent
- Formal languages are ill-suited to describing sensible and long established business rules
- “Something else” was needed; that something else began to be called “Software”

Software

- Some definitions:
  - *"… the printed materials supplied by a computer manufacturer to its customers"* (1972)
  - "*Computer programs, procedures, rules and any associated documentation concerned with the operation of a data processing system."* (1982)
  - *" ... those components of a computer system that are intangible rather than physical"* (1996)

Exercise



18/7/48 — Kilburn Highest Factor Routine (amended) —

Instn.	C	25	26	27	Line	0 1 2 3 4	13 14 15
-24 to C	$-b_1$	-	-	-	1	00011	010
c to 26			$-b_1$		2	01011	110
-26 to C	b_1				3	01011	010
c to 27			$-b_1$	b_1	4	11011	110
-23 to C	a	r_{n-1}	$-b_n$	b_n	5	11101	010
Sub 27	$a-nb_n$				6	11011	001
Test					7	—	011
Add 20 to Cl					8	00101	100
Sub. 26	r_n				9	01011	001
c to 25		r_n			10	10011	110
-25 to C					11	10011	010
Test					12	—	011
Stop	0	0	$-b_N$	b_N	13		111
-26 to C	b_n	r_n	$-b_n$	b_n	14	01011	010
Sub. 21	b_{n+1}				15	10101	001
c to 27	b_{n+1}			b_{n+1}	16	11011	110
-27 to C	$-b_{n+1}$				17	11011	010
c to 26			$-b_{n+1}$		18	01011	110
22 to Cl.		r_n	$-b_{n+1}$	b_{n+1}	19	01101	000

or 000

20	-3	10111 etc
21	1	10000
22	4	00100

or 10100

23	-a
24	b_1

	init.	final
25	—	r_N (=0)
26	—	$-b_N$
27	—	b_N

- This is a description of a procedure to find the highest factor of a number
- It was written on 18 June 1948 and run on the Manchester Mark 1 on 21 June 1948
- Is this software?

THE UNIVERSITY *of York*

Software and programs

- The influence of programs on software
  - Program design methods were (and still are) developed before software design methods
    - Structured code led to structured design methods, object-oriented languages led to object-oriented design methods
  - Software design must lead to software implementation (i.e. a program)
    - So called “seamless development” (i.e. applying the same ideas to both software design and program design) has some obvious attractions
  - History and expediency can lead to the assumption that program design methods are always “right”
    - Struggling to make software descriptions fit into the confines of formal languages is sometimes promoted as a virtue

Validation and Verification

- Validation and verification address the problem of demonstrating the suitability of a software description
  - Validation: are we building the right product? Validation is concerned with demonstrating that the software description is based on the right assumptions
  - Verification: are we building the product right? Verification is concerned with demonstrating the internal consistency of the software description

THE UNIVERSITY *of York*

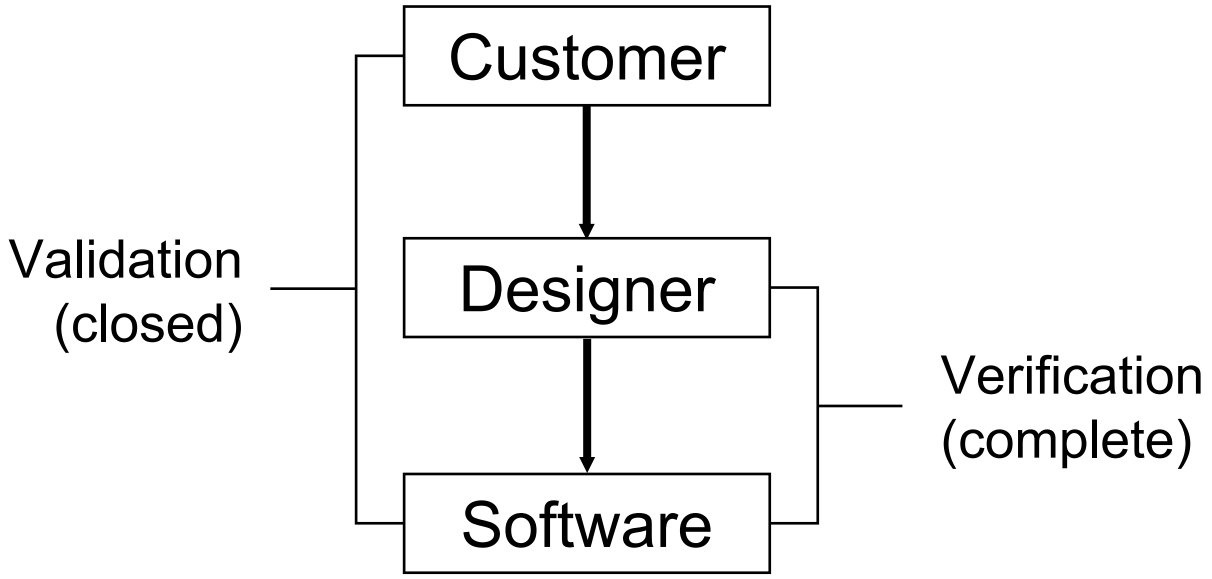
Validation and Verification

- Closed systems are self defining and nothing new can be introduced
  - Software designers test the assumption of closure through *validation*
    - Does the design correspond to the requirements as specified by the customer?
  - If a design is valid then requirement statements can be composed and decomposed without introducing errors
  - This allows the designer to (a) reduce one large problem into several smaller problems (b) structure those problems in such way that they are capable of being described in a formal language

Validation and Verification

- Complete descriptions mean that any part of a description can be traced directly to an axiom
  - Software designers test the assumption of completeness through *verification*
    - Is the design correct in terms of the requirements established at the beginning of the activity?
  - If verified a designer can claim that the relationship between the description and the thing being described is completely and unambiguously defined
  - This allows the designer to ignore differences between conceptual descriptions and the thing it describes within the scope of the system

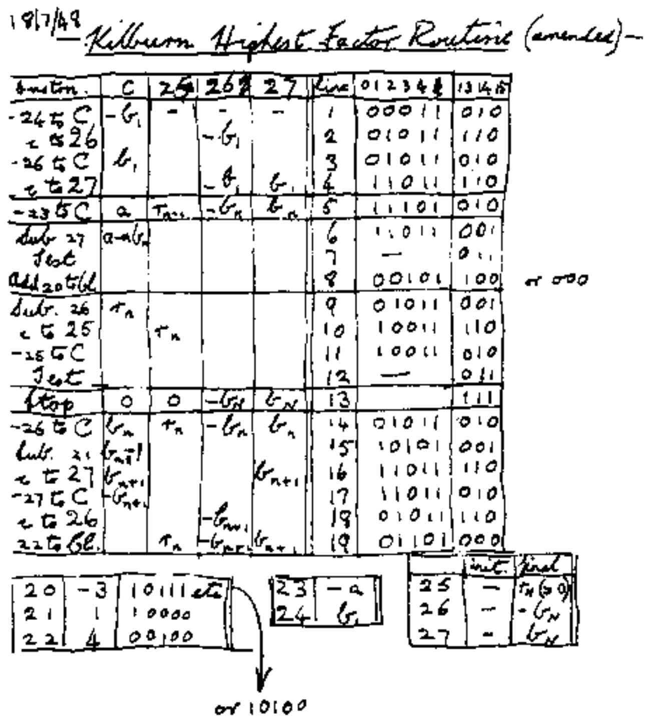
Validation and Verification



Closed and Complete Descriptions

- Closed and complete descriptions mean that a designer can claim the description and the thing being described are equivalent (i.e. they are isomorphic)
- This allows the designer to treat the descriptions and things they describe as being interchangeable
- Because of the unique environment of programs, program designers can do this (at least in theory), but software designers (at least in practice) never can

Exercise



18/7/48 Kilburn Highest Factor Routine (amended) –

Instrn.	C	25	26	27	Line	0 1 2 3 4	13 14 15
-24 to C	$-b_1$	–	–	–	1	00011	010
c to 26			$-b_1$		2	01011	110
-26 to C	b_1				3	01011	010
c to 27			$-b_1$	b_1	4	11011	110
-23 to C	a	r_{n-1}	$-b_n$	b_n	5	11101	010
Sub 27	$a-nb_n$				6	11011	001
Test					7	—	011
Add 20 to Cl					8	00101	100
Sub. 26	r_n				9	01011	001
c to 25		r_n			10	10011	110
-25 to C					11	10011	010
Test					12	—	011
Stop	0	0	$-b_N$	b_N	13		111
-26 to C	b_n	r_n	$-b_n$	b_n	14	01011	010
Sub. 21	b_{n+1}				15	10101	001
c to 27	b_{n+1}			b_{n+1}	16	11011	110
-27 to C	$-b_{n+1}$				17	11011	010
c to 26			$-b_{n+1}$		18	01011	110
22 to Cl.		r_n	$-b_{n+1}$	b_{n+1}	19	01101	000

or 000

20	-3	10111 etc
21	1	10000
22	4	00100

or 10100

23	-a
24	b_1

	init.	final
25	–	r_N (= 0)
26	–	$-b_N$
27	–	b_N

- This is a description of a procedure to find the highest factor of a number
- It was written on 18 June 1948 and run on the Manchester Mark 1 on 21 June 1948

- What is the difference between software and programs?

THE UNIVERSITY of York

Chris Kimble
January 2008

Software and Programs

The problem:

- Program design methods are totally bounded by the completeness and closure assumptions of formal languages.
- Software design methods are only partially bound by the completeness and closure assumptions of formal languages.
- Software designers must go someway towards finding a complete and closed description; otherwise, the program designers cannot design the programs

THE UNIVERSITY of York

Software and Programs

- Software must describe something of the human communities that design, use and form the environment for the programs.

- But some difficulties for software designers are:
  - Where do the boundaries of these communities lie?
  - What is the relationship between the communities and the computers attached to them?
  - How can we form closed or complete descriptions under these conditions?

Software and Programs

- Program descriptions are special cases of the software descriptions
- Software descriptions are special cases of the full complexity of the real relationships and associated communities
- Thus:
  - Software is a description of a special case of the relationships among human communities, the environment of those communities, and machines associated with them.
- By assuming that complete or closed descriptions exist, software design methods may fail to create an adequate description of reality.