
HSC 2010 Software Design and Development

Summary Notes

Jamie Kennedy

CONTENTS

Development And Impact Of Software Solutions	3
Social and ethical issues.....	3
Rights And Responsibilities Of Software Developers.....	3
Software Piracy And Copyright	4
The Software Market.....	6
Significant Social And Ethical Issues	7
Application of Software Development Approaches	8
Software Development Approaches	8
Software Development Cycle.....	15
Defining and understanding the problem	15
Defining the problem.....	15
Design Specifications	16
Modelling.....	19
Communication Issues, Including:	20
Planning and design of software solutions	21
Standard Algorithms For Searching And Sorting	21
Custom-Designed Logic Used In Software Solutions	24
Standard Modules (Library Routines) Used In Software Solutions	25
Customisation of existing software solutions.....	25
Documentation Of The Overall Software Solution	26
Selection Of Language To Be Used.....	27
Implementation of software solution	29
Interface Design In Software Solutions.....	29
Language Syntax Required For Software Solutions	29
The Role Of The CPU In The Operation Of Software	31
Translation Methods In Software Solutions.....	32
Program Development Techniques In Software Solutions	34
Documentation Of A Software Solution.....	38
Hardware Environment To Enable Implementation Of The Software Solution.....	39
Emerging Technologies	39
Testing and evaluation of software solutions.....	41

Testing The Software Solution.....	41
Reporting On The Testing Process.....	43
Maintenance of software solutions	45
Modification of code to meet changed requirements	45
Documentation Of Changes	45
Developing a Solution Package.....	46
Designing and developing a software solution to a complex problem	46
Defining the problem and its solution, including:	46
Systems implementation.....	46
The Software Developer’s View of the Hardware	48
Representation Of Data Within The Computer	48
Electronic Circuits To Perform Standard Software Operations	50
Programming Of Hardware Devices	53

DEVELOPMENT AND IMPACT OF SOFTWARE SOLUTIONS

SOCIAL AND ETHICAL ISSUES

Software developers and users have rights and responsibilities including:

- The right to good quality software for users
- The right to protection for the software creators
- The Responsibility to develop and use software in a social and legally ethical manner

RIGHTS AND RESPONSIBILITIES OF SOFTWARE DEVELOPERS

Software developers invest time and money into the development of their product. This investment brings responsibilities to the developer but also gives them rights over the product they develop.

Electronic material is easy to reproduce and distribute but not easy to develop. Many people are involved in the software design process and all have rights and responsibilities.

Authorship

Right	Responsibility
Protection of their product against theft and modification without their permission	Acknowledge the authors and sources used in development, particularly where there is a development team

Reliability

Right	Responsibility
Protection of their product against operating system problems and other hardware and programs which may make their product unusable	Check the product works with the hardware and operating system they specify.
	Ensure the product has no runtime errors when installed and run as directed

Quality

Right	Responsibility
Codes to ensure that others develop programs that follow the same high standard	Use thorough testing procedures and error checking code
	Meet the user's expectations as much as is possible

Response to Problems

Right	Responsibility
Not to be harassed with trivial problems that could have been solved by reading the documentation	Provide troubleshooting manuals and online help
	Provide customer support
	Quick response to major or critical problems
	Make “bug fixes” freely available to users

Code of Conduct

Right	Responsibility
All developers follow the same ethical standard	Adhere to the standards set by members of the professional association to which they belong

Viruses

Right	Responsibility
Protection of the developer’s products by users by the user of good current AV products	Ensure that they do not distribute viruses with their products or as part of their customer contact such as via email

SOFTWARE PIRACY AND COPYRIGHT

Software piracy is the theft of computer programs. This could involve copying the program or using a product that is installed illegally on a machine without the developer’s permission. Software piracy results in the increase in the cost of software for those who follow ethical standards and reduce user options as software developers have reduced incentive to develop new ideas

Concepts Associated With Piracy and Copyright, Including:

INTELLECTUAL PROPERTY

Intellectual property is personal ownership of the creative ideas that develop from an individual’s mind or intellect. It could include: patents, trademarks, designs, trade secrets and confidential business information.

PLAGIARISM

Plagiarism is the theft of the ideas and expressions of another person. Often code in a project is accumulated from a range of other sources. If the code or design is the work of any other developer then it must be acknowledged. Writing software using the code developed by others is plagiarism when the code is now acknowledged as coming from another source

SHAREWARE

Shareware is software that is distributed for trial use before purchase. Copies can be made and distributed by users but modifications are not allowed. If the user wishes to continue to use the software after the trial period, the shareware cost must be paid.

FREWARE

Freeware can be copied and distributed freely and no license fee needs to be paid however the product cannot be sold or modified and is covered by copyright.

PUBLIC DOMAIN

Public domain software is programs that are freely available for copying and modification. The copyright owner has surrendered copyright or the product is no longer covered by copyright.

OWNERSHIP VERSUS LICENSING

Purchasing a media that contains computer software does not mean that you own the software. The software distribution medium is your property however you have only been sold the right to use the software under certain conditions i.e. you have purchased a license

COPYRIGHT LAWS

Copyright is the legal protection of computer programs against illegal copying. The Copyright Act 1968 and a series of court decisions govern copyright in Australia. The creator of the program owns copyright; this could include any other people associated with the development of the program,

REVERSE/BACKWARDS ENGINEERING

Reverse engineering is the process of reading source code and translating it into an algorithm. The algorithm can then be modified and recoded in the same or another programming language. Reverse engineering is legal when the program is owned by the developer carrying out the reverse engineering however it is illegal if someone else does.

DECOMPILATION

Decompilation is the process of translating object code (machine code) into code that can more easily be studied by the programmer. It is legal if the developer owns the program but is it not if they don't

LICENCE CONDITIONS

License conditions determine what can be done with software. Many developers include a compulsory reading and acceptance of the EULA before installing can continue. This leaves users no excuse for failure to understand the developer's wishes.

NETWORK USE

Software developers have recognised the increasing popularity of networked computers. Programs are now available for network use. They could be either a) centralised software in which software is available as a single copy on a central server or b) distributed software which is available on individual machines. Regardless, each machine on the network or using the software requires a separate license.

Various National Perspectives to Software Piracy and Copyright Laws

Australia is a signatory to international agreements that recognise international copyright laws however, although the distribution of cracked or warez programs is illegal in Australia, it is often not the case in which these countries these programs originate from.

The Relationship between Copyright Laws and Software License Agreements

Software license agreements are contracts that protect the developer's ownership of the software they have created. Different license agreements determine the way some software may be used such as: single use license allows the installation of one copy of the software on one machine. A multi-use license provides one copy of the software but allows the software to be installed on a specific number of machines. Network licenses are for networks... obviously.

THE SOFTWARE MARKET

Maintaining Market Position

The social and ethical standard of a company is important in this area. The product to be sold must be able to meet the needs of users. Some factors that will help software developers to maintain market position include:

- Selling price
- Customer support for the current and earlier version of the product
- Reputation as an ethical developer
- Wide distribution network
- Bug-free software

The Effect on the Marketplace

All software developers want to have an effect on the marketplace. Advertising and other forms of promotion are often used to gain audience attention and build sales. Packaging can also be important for shelf presence as well the new move towards the distribution of content of the internet.

SIGNIFICANT SOCIAL AND ETHICAL ISSUES

National and International Legal Action Resulting From Software Development

Sega vs. Accolade: Accolade wanted to enter the market to produce games that worked on Sega hardware. To do so required certain secret code created by Sega. Sega denied access to the code and Accolade reverse engineered the Sega code. Sega took Accolade to court but the court ruled that Accolade had the right to reverse engineer as it has not stolen Sega creative works and only needed the code to be competitive in the marketplace.

Whelan vs. Jaslow: Whelan developed a dental program that took a considerable amount of time to build and market. A few years later, Jaslow released a very similar product onto the market. Whelan sued and won because the court ruled that Jaslow had used very similar code to Whelan.

Public Issues, Including:

THE Y2K PROBLEM

Early computers had limited memory and RAM was expensive. To save memory, programmers developing operating systems reduced the storage of dates to a two digit integer reducing storage needs by half. As the year 2000 approached, there was great concern about what would happen to software that governments, banks and large businesses depended on. Large amounts of money were spent on changing the dates to a four digit format.

COMPUTER VIRUSES

A virus is a program that alters the functionality of a computer without the permission of the user. Viruses are a far larger issue than the Y2K bug due to their persistent and developing nature. Both software developers and end users have a responsibility to combat viruses. Install and maintain a reliable AV program that is kept up to date with the latest patches.

RELIANCE ON SOFTWARE

As shown by the Y2K bug, programmers need to consider all aspects of the software they develop particularly the needs in the future. Software reliability becomes crucial when society relies heavily on software for many of its basic services

APPLICATION OF SOFTWARE DEVELOPMENT APPROACHES

SOFTWARE DEVELOPMENT APPROACHES

Approaches Used In Commercial Systems, Including:

Commercial software development falls into two main broad categories.

Off the shelf or retail programs (e.g. Microsoft Office) are sold and distributed to any user who wishes to use them. Sometimes referred to as COTS and in many cases can be customised to suit the need of an organisation (e.g. Google Apps for the DET)

Custom programs are commissioned by a particular organisation or individual and are written specifically to suit the defined needs to those requiring the program.

THE STRUCTURED APPROACH

Advantages	Disadvantages
Thoroughly tested	Costly
Should meet exact requirements of users	Time consuming
Uses a range of experts	Requires a range of different skills

The structured (or waterfall) approach to software development follows the software development cycle. It considers the whole problem and divides it into steps that can be systematically followed to arrive at a solution. Because of the complexity of this approach, many people can be involved.

A systems analyst is a person with the skills and knowledge to see the problem as a whole and divide it into elements. They manage the project and are the first level of communication between development and management teams. They are often given control of the whole project.

Define the problem: Understand and define the problem. This is important so you know what you are solving. It is much easier and cheaper to fix mistakes here than in any other stage of development.

Planning the solution involves a further understanding of the needs of the users and a choice of a method or methods to solve the problem. Data needs to be collected to provide the basis for decisions. Planning involves designing algorithms, planning a UI, data and program structures needed, scheduling the project, choosing a programming language. As well, dataflow diagrams, IPO charts, Gantt charts, screen designs and storyboards are all likely tools that will be used here.

Building the solution requires the full involvement of the team. A large and complex project is usually broken down into smaller modules; this process is called stepwise refinement. By breaking parts down into modules, you allow for greater efficiency, easier debugging, less

contamination of the whole project if there is badly written code and you can reuse the modules that you create.

Checking the solution is a continuous part of the development cycle. It involves using real data and may include beta testers. This is to make sure that the program meets the needs and requirements determined in the defining stage. If the project passes management approval, it can be implemented.

Sometimes it is necessary to **make modifications** after a program has been implemented so that it works more effectively. The program may need to be updated to keep abreast of hardware or software changes or may need to be expanded to cover new tasks.

PROTOTYPING

Advantages	Disadvantages
Relatively fast development	May be difficult to implement as a fully working solution
Models a larger project, thus allowing easier modification and visualisation of the end product	

Prototyping involves building a working model that is then evaluated by users. The model is then usually modified and evaluated further until it becomes the solution. The prototyping approach usually only involves a small team of programmers and one or more users. The prototyping approach is particularly useful in the development of interactive systems and AI but not as much where complexity and large mathematical calculations are required.

There are two type of prototyping used. **Information gathering prototypes** are developed to gain information that can be used in another program. The prototype is never intended to become the fully working solution. They are often developed in 5GL languages and use reusable code modules that require only linking to operate.

Evolutionary prototypes become the full working program. The prototype is the first step in the development of the final product. This prototype becomes the short term solution and allows the product to be demonstrated before the fully working program is produced.

Prototyping focuses largely on the UI including menus, windows and IO processes. There is a great amount of user involvement in the development process and there is reduced amounts of documentation compared to the structured approach however this means that the system is harder to maintain.

RAPID APPLICATIONS DEVELOPMENT

Advantages	Disadvantages
(RAD) Fast development	Many not meet exact program requirements
Relatively cheap	Many involve copyright and intellectual property of others
Uses considerable amount of existing reusable code	

The **rapid applications software development** is any method of software design that uses tools to quickly generate a program for a user. It uses existing modules and may reuse code, CASE tools and templates. The developer is involved directly with the user.

END USER DEVELOPMENT

Advantages	Disadvantages
(EUD) Should meet exact user requirements	Limited to simple projects and limitations of application programs
Quick and cheap	

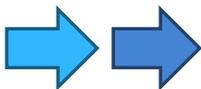
End user development is where the user adapts existing software tools to suit their needs or to obtain a solution to a problem. EUD is very informal and used for very small solutions. It is usually created in a 4GL programming environment such as in a spread sheet or database.

COMBINATIONS OF ANY OF THE ABOVE

Using combinations of any of the four approaches can make it easier to develop a program. If different parts require different approaches then using the right approach for each part will make the process more effective.

Methods of Implementation

DIRECT CUT OVER

Description	Diagram	Advantages	Disadvantages
The new program immediately replaces the old program		Reduced implementation costs compared to other methods	Data transfer from old to new has to be done very quickly
		Less pressure on users as only one program is in place	Testing of new program in operation leaves no fall back to old program if there are problems
			User stress as training

Summary Notes

	needs to be done before new program is in place while old program is still operating
--	--

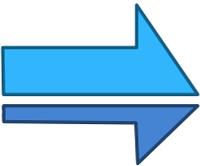
PARALLEL

Description	Diagram	Advantages	Disadvantages
The old and the new work together over a period of time		Two programs can be compared and any problems can be fixed to take account of differences	User stress as both old and new programs have to be operated for conversion time
		Testing and fixing of problems in new program is simpler as old program is still available for use in an emergency	Confusion between programs if close records are not kept
			If there are large changes in operating procedures between the old and new program there can be problems for users
			More expensive than other methods due to dual costs

PHASED

Description	Diagram	Advantages	Disadvantages
One or more tasks of the new program are gradually implemented until the new program takes over all tasks of the old program		Each task can be individually tested as it is implemented	Difficult for users to separate the old and the new programs and operate different tasks in each system
		Training users is simpler as only one new task has to be learnt	Longer time frame leads to high implementation costs

PILOT

Description	Diagram	Advantages	Disadvantages
One section of the organisation use the new program and all other sections continue with the old program until a decision is made to put the new program into place across the whole system		Risk is confined to one section of an organisation	Large organisations usually use this method particularly multiple sites doing the same thing such as one bank amount a series of branches
		Testing and fixing of problems in the new program is simpler as the old program is still available for use in other parts of the organisation	

Current Trends in Software Development, For Example:

OUTSOURCING

Outsourcing is the process where a company needing software services hires an outside organisation to handle all or part of these services. This could include development, implementation, and maintenance. Outsourcing is an effective use of scarce technological assets and costs savings due to economies of scale however some outsourcing companies many not understand the ethos of the company employing it and the organisation may feel as if it is losing control of the system.

POPULAR APPROACHES

Software of the popular approaches to software development at the moment includes:

- Increasing facilities for EUD
- The availability of editors to allow non-programmers to generate simple programs such as WYSIWYG editors.
- Greater distribution of code libraries
- Authoring languages that guide the developer through the steps of creating programs including markup languages
- Increasing use of multimedia in programs

POPULAR LANGUAGES

4GL and 5GL languages are increasing becoming human centred to allow non-programmers to develop customised software. SQL, is very powerful language but is limited to the domain

of databases. It can be used to someone who is relatively inexperienced to create powerful database structures

Most of the popular languages have the same basic capabilities such as syntax, constants, variables and operators.

EMPLOYMENT TRENDS

Technology has been a growth area in employment over the last decade and there are still large numbers of jobs in the field despite a recent slowdown. There are increasingly high educational requirements for those entering the industry.

NETWORKED SOFTWARE

With the increasing trend towards networks in organisational environments, there has come a greater demand for networked software. This comes in two forms, network OSes (Windows Server 2008, Snow Leopard Server, Ubuntu Server Edition) and network applications (MYOB, databases).

CUSTOMISED OFF-THE-SHELF PACKAGES

COTS are readily available today and are often released as suites of programs that integrate together do perform a wide range of tasks such as Microsoft Office, Adobe Creative Suite. These programs can reduce costs for the end user and are more reliable than custom software as they have been tested on a wider scale. They are modular and more easily updated and modified to suit the user's needs.

Use of Case Tools and Their Application in Large Systems Development

Software development teams often use a CASE approach to their works. CASE tools allow the software developer to track bugs, data model, generate documentation, reverse engineer and simulation tools. CASE tools help the maintenance team to interact with the development team

SOFTWARE VERSIONS

When developing a full product, there will be many versions such as for different hardware, operating systems. The number of versions leads to the need for version control and CASE tools can be used to take over this process. CASE tools allows for strict control over the versioning process.

DATA DICTIONARY

CASE tools can collect and track data types through multiple modules and develop concise descriptions of each data type in a program. This can be a major time advantage.

TEST DATA

CASE tools can be used to generate test data for complex programs. This can avoid time consuming manual testing or the generation of large test data tables by a slower input method

PRODUCTION OF DOCUMENTATION

There are a number of CASE tools available to help the documentation process with facilities such as word processors and drawing tools. Others can produce Gantt charts, calendars and other project management tools.

SOFTWARE DEVELOPMENT CYCLE

DEFINING AND UNDERSTANDING THE PROBLEM

DEFINING THE PROBLEM

The programmer must have a complete understanding of the problem before developing a solution. This will establish what needs to be done.

Identifying the Problem

This involves determining the requirements of the program to be developed. Includes:

NEEDS

It is the programmer's job to determine how to solve a problem or meet the need of a user. Requirements definition: clear statement of the requirements of the system being developed.

OBJECTIVES

The short and long term aims and plans for the software being developed.

BOUNDARIES

The boundary is the limit to a system such as the scope of a project. It is also important to consider any constraints

Determining the Feasibility of the Solution

A feasibility study is carried out usually after requirement definition to determine whether the proposed solution is practical.

IS IT WORTH SOLVING?

An investigation into the problem will allow the systems analyst (or developer) to decide if the problem can be solved or if an alternative route can be taken.

TECHNICAL

For example: what software and hardware is currently being used and if this a solution can be built around it or if new hardware/software needs to be bought (then consider economic)

ECONOMIC/BUDGETARY

Is the solution to the problem affordable? A cost/benefit analysis of the solution will be used to determine initial and recurring costs as well as benefits to the consumer.

SCHEDULING

Is there enough time to allow the solution to be developed? Planning tools such as Gantt charts can be used and the developer must be able to meet these deadlines for the solution to be achievable.

OPERATIONAL

Will the solution be usable by the target consumer? The future user of the software must be able to use and operate the program. If training costs are required then this should be factored into the cost (see economic)

LEGAL

Does the software comply to legal standards? For example, does it calculate GST correctly? Does it acknowledge other author's if their code is used?

SOCIAL/ETHICAL CONSIDERATIONS

The software must be developed in a socially responsible manner and not be socially biased. It should be widely accessible (but this may impact on economic and technical)

POSSIBLE ALTERNATIVES

If the solution is determined to not be feasible, then the developer may choose to explore alternative solutions

DESIGN SPECIFICATIONS

The Developer's Perspective in Consideration Of:

DATA TYPES

Simple data types include:

- Integers: signed whole numbers (+32, -98)
- Real/Floating Point: Numbers with decimals or exponents
- Character: any character on the keyboard
- Boolean: only two possible values (T/F)

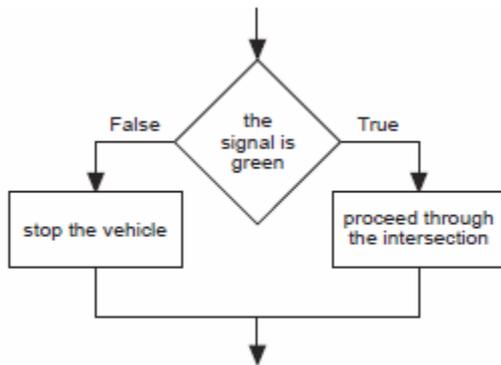
Structured data type:

- String: composed of multiple characters which can be referred to by a single name and manipulated as a group. Also, can be seen as an array of characters.
- One Dimensional Array: A number of data items of the same type referred to by the same name. These items can be accessed by their "index" number. An array can also be considered a list.
- Multidimensional Array: Two or more separate arrays where elements refer to similar data. An example of this could be a spread sheet or grid, this will have two index positions. [Similar to dictionary in python]
- Single Record: A collection of different data types that are related. Consists of a number of fields, each accessed by name. Differ to arrays in that they can contain different data types.
- Array of Records: Single records combined together into an array and can be accessed with an index (similar to a database)
- Files: A data structure that allows data to be stored externally and may consist of any other data type except a file

ALGORITHMS

An algorithm is a series of instructions or steps that will result in the solution to a specific problem. It can be represented in pseudocode or in flowchart form.

Binary Selection:

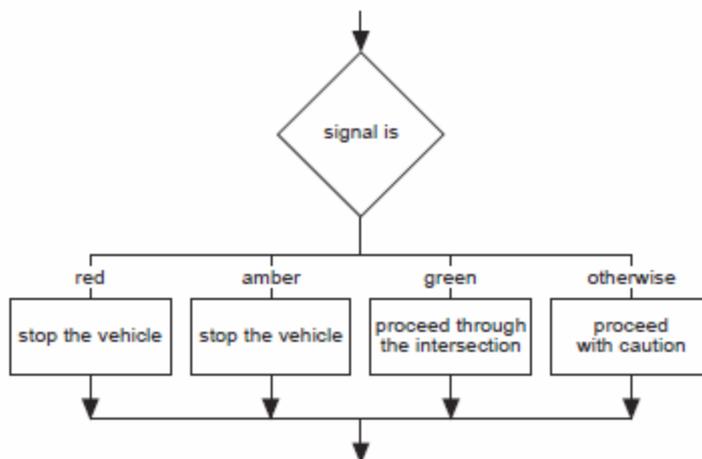


IF the signal is green THEN
 proceed through the intersection

ELSE
 stop the vehicle

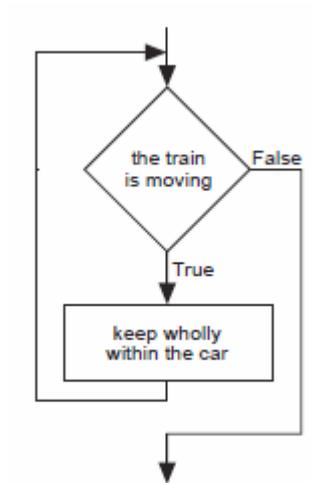
ENDIF

Multi-way selection:



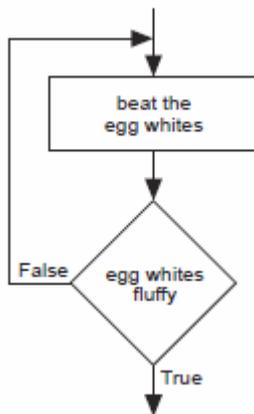
CASEWHERE signal is
 red : stop the vehicle
 amber : stop the vehicle
 green : proceed through the intersection
 OTHERWISE : proceed with caution
ENDCASE

Pre-test (guarded) loop:



WHILE the train is still moving
 keep wholly within the car
END WHILE

Post-test (unguarded) loop



REPEAT
 beat the egg whites
UNTIL fluffy

VARIABLES

A variable is a pointer to an area in memory that is being used by a program to store data. The contents are not fixed and may vary.

The User's Perspective

It is important for software developers not only to analyse the problem but to consult with the user to ensure the task they are solving is clearly defined.

MODELLING

Representing a System Using Diagrams, Including:

System modelling tools allow the system designer to communicate and record the system so that users and other developers can understand the system. There are three attributes that needs to be represented. These are the function, the logic and interfaces.

INPUT PROCESS OUTPUT (IPO) DIAGRAMS

An IPO chart describes the system in terms of its input data, its output data and the processes that are performed on the inputs to transfer them into the outputs.

STORY BOARDS

Storyboards give a general overview of a program. They are used to document the screens used in a system and the flow between them. They are suited to applications with a large number of screens of information which link to other screens. Storyboards can also be useful in planning the flow of information between modules.

CONTEXT DIAGRAMS

Context diagrams are used to represent entire information systems. This system is shown as a single process along with the input and output (external entities) to the system.

DATA FLOW DIAGRAMS

Data flow diagrams represent the flow of data into and out of the system in terms of the processes used. A data flow diagram provides more detail at a lower level than a context diagram. Contains processes, data flows, external entities and data stores.

SYSTEMS FLOWCHARTS

System flow charts are a diagrammatic way of representing both the flow of data and logic through an information system.

SCREEN DESIGNS

The screen is the first point of contact between a user and a program. Sample screens may be developed to provide the customer with a clear idea of the final interface of the program.

CONSIDERATION OF USE OF A LIMITED PROTOTYPE

A prototype is a hands-on model of the computer system that concentrates on the design of input and output screens. Prototypes are often concerned with the final design and can give users a good indication of what the final program will look/feel like.

COMMUNICATION ISSUES, INCLUDING:

The Need to Empower the User

It is important to include the user in the software development process.

The Need to Acknowledge the User's Perspective

Software developers and users view the solution from different perspectives and it is important to understand the user's views. Users want operational software that does what they want and it is up to the developers to provide this.

Enabling and Accepting Feedback

The developer must communicate regularly with the user to ensure that the solution will meet the needs of the user. The cost to fix a program increases exponentially as the system is developed and it is much cheaper to fix any errors earlier on than later.

PLANNING AND DESIGN OF SOFTWARE SOLUTIONS

STANDARD ALGORITHMS FOR SEARCHING AND SORTING

Standard Logic Used In Software Solutions, Namely:

FINDING MAXIMUM AND MINIMUM VALUES IN ARRAYS

Max and min values can be found by moving through each array element in a sequential manner and comparing the current element with the value currently thought to be the max or min. If the new value is a max or min, it now becomes the one that others are compared against

```
BEGIN MAINPROGRAM
  Set Max to TheArray[1]
  Set Min to TheArray[1]
  Index = 2
  WHILE index < 100 THEN
    IF TheArray[Index] > Max THEN
      Max = TheArray[Index]
    END IF
    IF TheArray[Index] < Min THEN
      Min = TheArray[Index]
    END IF
    Increment Index
  END WHILE
  Print "The highest number found was:" Max
  Print "The lowest number found was:" Min
END MAINPROGRAM
```

PROCESSING STRINGS (EXTRACTING, INSERTING, DELETING)

A string is an array of characters and this means that they can be manipulated. Strings can be joined together (concatenated), characters to be inserted and deleted, portions of strings to be extracted and for the length to be calculated

FILE PROCESSING, INCLUDING SENTINEL VALUE

A file is a collection of data stored on a secondary storage device. They must be opened and closed to be read and written to. Files often include a sentinel value which is a special symbol that terminates the file. This symbol is usually an EOF (End of File) symbol.

LINEAR SEARCH

A linear search is used to search through a set of data sequentially. Each value is compared to the target and if the value is not the same, the next value is compared until the target has been found or the end of the array has been reached.

```
BEGIN MAINPROGRAM
  Set Found to False
  Ask user to enter target
  Next = 1
  Last = 100
  WHILE Next <= Last AND Found = False
    IF List[Next] = Target THEN
      Found = True
      FoundPosition = Next
    END IF
    Increment Next
  END WHILE
  IF Found = True THEN
    Print "Target was found at position" FoundPosition
  ELSE
    Print "Target was not found"
  END IF
END MAINPROGRAM
```

BINARY SEARCH

A binary search is used to search data that has been sorted. It divides the data into two parts and determines which part the data lies in with the other part discarded. The remained is split in two and the process is repeated again until the target is found or there are no more items left to divide.

```
BEGIN MAINPROGRAM
  Set Lower to 1
  Set Upper to 12
  Set Found to False
  Get the Target value
  REPEAT
    Set middle value to integer of (Upper + Lower) / 2
    IF Target = NumArray[Middle] THEN
      Set Found to True
      Set FoundPosition to Middle
    ELSE
      IF Target < NumArray[Middle] THEN
        Set Upper to Middle - 1
      ELSE
        Set Upper to Middle + 1
      END IF
    END IF
  UNTIL Found OR Lower > Upper
  IF Found THEN print "found" ELSE print "not found" END MAINPROGRAM
```

BUBBLE SORT

```
BEGIN MAINPROGRAM
  Set End to ArrayLength
  WHILE End > 1
    Set Current to 1
    WHILE Current < End
      IF TheArray[Current] > TheArray[Current + 1] THEN
        Swap(Current, Current + 1)
      END IF
      Increment count
    END WHILE
    Decrement count
  END MAINPROGRAM

BEGIN SUBPROGRAM Swap(Position1, Position2)
  Temp = TheArray[Position1]
  TheArray[Position1] = TheArray[Position2]
  TheArray[Position2] = Temp
END SUBPROGRAM Swap
```

INSERTION SORT

```
BEGIN MAINPROGRAM
  Set First to 1
  Set Last to ArrayLength
  Set PositionOfNext to Last
  WHILE PositionOfNext >= First
    Next = TheArray[PositionOfNext]
    Current = PositionOfNext
    WHILE (Current < Last) AND (Next > TheArray[Current + 1])
      {Shuffle sorted part along}
      Increment Count
      TheArray[Current - 1] = TheArray[Current]
    ENDWHILE
    TheArray[Current] = Next
    Decrement PositionOfNext
  ENDWHILE
END MAINPROGRAM
```

SELECTION SORT

```
BEGIN MAINPROGRAM
  Set EndUnsorted to ArrayLength
  WHILE EndUnsorted > 1
    Set Current to 1
    Set Largest to TheArray[Current]
    Set PositionOfLargest to Current
    WHILE Current < EndUnsorted
      Increment Count
      IF TheArray[Current] > Largest THEN
        Set Largest to TheArray[Current]
        Set PositionOfLargest to Current
      END IF
    END WHILE
    Swap (PositionOfLargest, EndUnsorted)
    Decrement EndUnsorted
  END WHILE
END MAINPROGRAM

BEGIN SUBPROGRAM Swap(Position1, Position2)
  Temp = TheArray[Position1]
  TheArray[Position1] = TheArray[Position2]
  TheArray[Position2] = Temp
END SUBPROGRAM Swap
```

CUSTOM-DESIGNED LOGIC USED IN SOFTWARE SOLUTIONS

Requirements To Generate These Include:

IDENTIFICATION OF INPUTS, PROCESSES AND OUTPUTS

Here the software developer must fully describe the data and its format. It is helpful to develop an IPO diagram to show the relationship between inputs, processes and outputs

REPRESENTATION AS AN ALGORITHM

Writing an algorithm helps the programmer to understand the logic of the problem making it easy to convert it straight into code

DEFINITION OF REQUIRED DATA STRUCTURES

Data structures are a set of rules for storing and manipulating data. Developing a data dictionary enables the programmer to determine the composition of data that is to be used in the program.

USE OF DATA STRUCTURES, INCLUDING MULTI-DIMENSIONAL ARRAYS, ARRAYS OF RECORDS, FILES (SEQUENTIAL AND RELATIVE/RANDOM)

The choice of data structure is important as it will have a significant impact on the algorithm used. There is often a trade-off between complexity of the algorithm and the data structure used.

USE OF RANDOM NUMBERS

Random numbers are often used to allow access to unpredictable data.

THOROUGH TESTING

The amount of time spent on each of the other phases will reduce the amount of time that is required to test. A well designed solution will require little testing time and produce an easier to implement program. Algorithms should be desk checked before use.

STANDARD MODULES (LIBRARY ROUTINES) USED IN SOFTWARE SOLUTIONS

Most programming languages have library routines that can be accessed for use in a program such as the “print” statement in Python.

Requirements For Generating Or Subsequent Use Include:

Structured programming results in modular programs, each module carrying our specific tasks

IDENTIFICATION OF APPROPRIATE MODULES

Programmers will only use modules or libraries that are necessary to perform the task required. It would be a waste of resources to include other redundant libraries that are not accessed by the program.

CONSIDERATION OF LOCAL AND GLOBAL VARIABLES

Local variables are those which can only be accessed from within that particular subprogram and will not be recognised outside of it.

Global variables are those that can be used from anywhere within the program.

APPROPRIATE USE OF PARAMETERS (ARGUMENTS)

A subprogram often requires arguments of parameters to be included in the subprogram call. Parameters are data items that can be passed from one part of the program to another where the values can be used.

APPROPRIATE TESTING USING DRIVERS

A driver is a special instance of a stub. It is a temporary section of code that is created to test an individual procedure or module by calling it up and executing it.

THOROUGH DOCUMENTATION

Any modules of code developed should be thoroughly documented and should include parameters, inputs, processes and output. This will allow programmers to reuse modules of code and understand the processes being carried out in the future.

CUSTOMISATION OF EXISTING SOFTWARE SOLUTIONS

It is much easier to develop a new program from a known working program instead of starting from scratch.

Identification of Relevant Products

Many languages are purchased with sample programs and some programming languages include portions of code that can be used by programmers to develop their own solutions

Customisation

Customising software involves determining the portions of the program which are relevant to the solution that you are developing.

Cost Effectiveness

Customising software solutions often results in faster development time and lower cost but if the original program performs insufficiently than in the long run, the newer program may end up costing more.

DOCUMENTATION OF THE OVERALL SOFTWARE SOLUTION

Tools For Representing A Complex Software Solution Include:

ALGORITHM DESCRIPTIONS

Algorithms provide a detailed description of the logic that is carried out in a program.

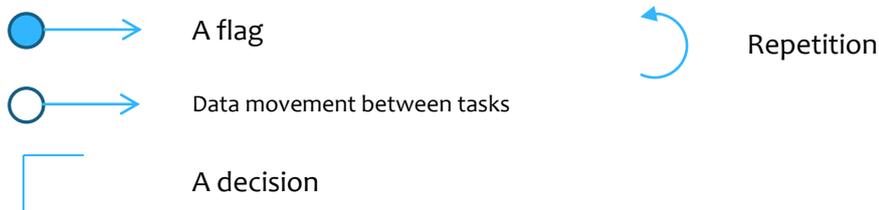
SYSTEM FLOWCHARTS

System flowcharts are a graphical representation of the logic of a computer system. They demonstrate the source and destination of data which is used by the system.

STRUCTURE DIAGRAMS

Structure diagrams are a method of representing the elements of a system in a hierarchical form. Modules are represented by a box. They are suited to top-down methods of solving the problem.

Rectangles are used to represent tasks with lines used to show the connections between tasks



DATA FLOW DIAGRAMS

Data flow diagrams are useful for tracking the movement of data through a system. They show graphically the processing which occurs in a system and indicate where data is stored

DATA DICTIONARY

Data dictionaries help the programmer and users of the system to understand the data that will be used in a particular program

SELECTION OF LANGUAGE TO BE USED

Event-Driven Software

Event-driven software is executed in an order dependent on the events detected.

DRIVEN BY THE USER

The user determines the sequence in which the programming tasks are carried out by the processor. Events may occur in any order and event driven software typically involves the use of menus and buttons.

PROGRAM LOGIC

During program execution, the computer's operating system polls the hardware to generate a stream of events. An event driven program will act upon the events detected during this polling processes

Sequential Approach

A sequential program will execute instructions in a linear fashion one after the other until the end of the program is reached.

DEFINED BY THE PROGRAMMER

Sequential programs must follow a predetermined order of events executing the first statement and subsequent statements until the program is complete. While the program is executing a statement, it cannot react to anything else. Each task is carried out in the order as specified by the programmer.

Relevant Language Features

The programmer must carefully consider the nature of the problem before choosing a programming language. One particular language may be more suited to a problem than another (such as functional vs object oriented)

Hardware Ramifications

The language chosen depends on the hardware architecture. Certain programs may be more suited to different hardware types. As well, the speed of the hardware the required execution time would determine the final language used. For example, if execution time was important, a fast language such as C would be used

Graphical User Interface (GUI)

Graphical User Interfaces are usually made up of Windows, Icons, Menus and Pointers (WIMPs) and they display information on the screen in the way it will be printed (WYSIWYG)

IMPLEMENTATION OF SOFTWARE SOLUTION

INTERFACE DESIGN IN SOFTWARE SOLUTIONS

The Design of Individual Screens, Including:

As the screen is the first point of the contact between the user and the software, it is important that the interface used does not frustrate the user, prevent the efficient entry of data and lead to physical fatigue.

IDENTIFICATION OF DATA REQUIRED

A well laid out screen should focus the reader's attention on the important areas and main interest. Screens should be simple with enough white space around them. There should be variety on the screen as well as consistency in the overall program. Graphic elements are also important. Legibility refers to the ease with which a screen can be read, a program should be legible. The positioning of text on the screen should be taken into account as to not overwhelm the user. As well, the type of font (serif, sans serif), alignment and justification, upper and lower case, colour of text, size and background should also be considered.

CURRENT POPULAR APPROACHES

The most popular approach today is the use of the GUI and WIMP interface. This includes other navigational elements such as scroll bars, hypertext (links), radio buttons and dialogue boxes.

DESIGN OF HELP SCREENS

Help screens and error messages should be non-threatening. It is important that the user not be put down or turned off by the language of the program. Error messages should be helpful and provide instruction on what to do. Error messages should be consistent throughout. User prompts should be specific and appropriate.

AUDIENCE IDENTIFICATION

The language that the program uses to communicate with the intended audience should be specifically suited to that target audience. For example the language and layout would differ greatly between a program targeted at children compared to an engineer or a visually impaired person.

CONSISTENCY IN APPROACH

For ease of use and to aid in navigation, it is important that the programmer maintains consistency in screen design throughout including consistent placement of repeated elements (such as menus), user interface (such as logical shortcuts) and fonts (same heading font).

LANGUAGE SYNTAX REQUIRED FOR SOFTWARE SOLUTIONS

The syntax of the language refers to the rules that determine whether a statement is legal and so can be executed.

Use of BNF, EBNF and Railroad Diagrams to Describe the Syntax of New Statements in the Chosen Language

The use of a metalanguage such as BNF to describe the syntax of a language allows users to interpret definitions and check the legality of constructions.

BNF:

```
word ::= <letter><word>|<letter>
letter ::= a|b|c|d|e|...|y|z
```

Repetition is described through recursion. ::= stands for “is defined as” and non-terminal elements are enclosed with <> brackets.

EBNF:

```
word = <letter>{<letter>}
letter = a|b|c|d|e|...|y|z
```

Symbols similar to BNF with the addition of {} for repetition, [] encloses optional elements, () groups elements together.

Railroad diagrams are also known as syntax diagrams or structure diagrams. They are a method of graphically representing the syntax by tracing a one-way path or railroad track from left to right. Branching is only permitted in the direction of motion. Terminals are represented by circles or rounded rectangles. Rectangles represent previously defined elements and repetition is shown by a looping structure.

Commands Incorporating the Definition and Use Of:

MULTI-DIMENSIONAL ARRAYS

ARRAYS OF RECORDS

FILES (SEQUENTIAL AND RELATIVE/RANDOM)

Files allow for data to be stored on secondary storage devices. Traditionally stored and accessed using sequential techniques (accessed from start to finish such as a VHS).

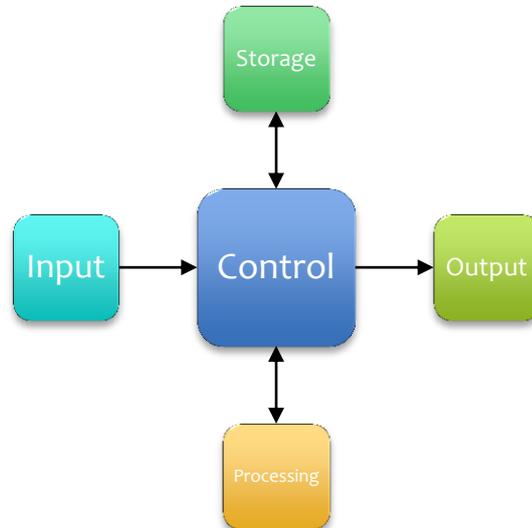
Random (direct) access uses indexes to identify separate items of data (such as a dictionary in Python – hashtables) which means that it can be accessed directly and quickly.

RANDOM NUMBER GENERATORS

Random numbers are generated by picking a starting number (seed) and performing a variety of mathematical operations on it. The seed is often the current time to avoid the same random numbers being generated each time.

THE ROLE OF THE CPU IN THE OPERATION OF SOFTWARE

All modern computers are based on the ideas of input, output, memory and a central controller.



Machine Code and CPU Operation

Computers are run by instructions stored in binary form called machine code. Instructions are fetched one at a time from the computer's memory (RAM) and interpreted by the control unit. Arithmetic and logic operations are performed by the ALU. Machine language is the only language that the computer can understand without translation.

INSTRUCTION FORMAT

An instruction is a command given to the CPU by a program. The CPU uses a fetch-execute cycle. Op codes are the machine language command for a single operation of the CPU such as ADD or STORE. The CPU uses these to move numbers from a memory location to the accumulator for example. Each memory location in RAM has an address and a storage location that contains data or instructions.

USE OF REGISTERS AND ACCUMULATORS

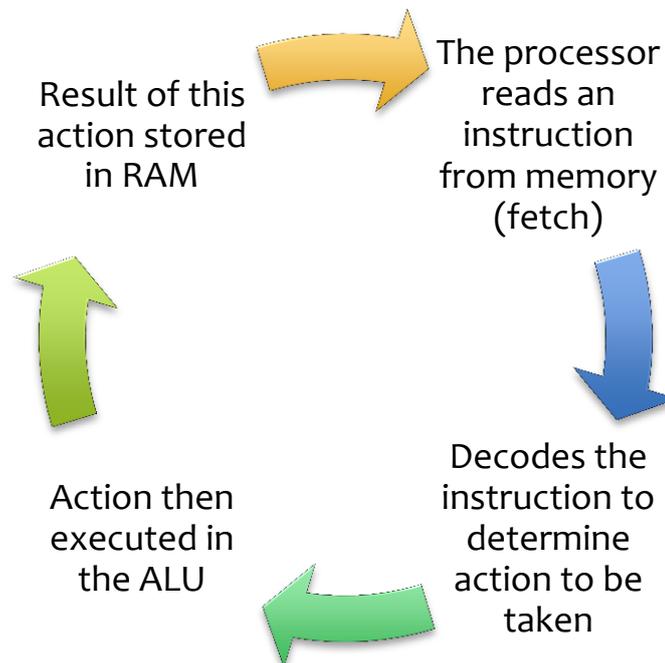
A register is a temporary storage location that is able to hold one instruction. Everything must pass through the buffer register.

The accumulator is a specific register in the CPU. It stores data that is about to be computed or the results of a computation. Data must be moved to the accumulator before a mathematical or logical operation can be performed. Data is then passed back to the accumulator before being stored.

USE OF PROGRAM COUNTER AND FETCH-EXECUTE CYCLE

The program counter is a register that is part of the computer's CPU. The program stores the address of the next instruction to be executed. The value in the program counter is automatically incremented when an instruction is executed.

The fetch-execute cycle is performed continually by the CPU.



ADDRESSES OF CALLED ROUTINES

The address of a command to be executed is stored with the op code in RAM,

LINKING, INCLUDING USE OF DLL'S

Linking allows for machine code to be combined with another machine code program. The use of a linker allows only the modules being used at any one time to be loaded into RAM. This saves on resources.

A DLL is a collection of programs in machine code which can be accessed by other programs to perform specific tasks. DLL files are often called device drivers because they control a specific device.

TRANSLATION METHODS IN SOFTWARE SOLUTIONS

Computers can only execute code that is in machine language. A translator converts statements from one programming language to the format of another (eg: machine code). This has many advantages as certain complex structures are only available in high level languages and is far more human readable than machine language.

Different Methods Include:

COMPILATION

Compiling takes the whole source of a program and translates it producing a complete object program that can be executed at a later stage. Once compiled, it is much more difficult to change the program which can help protect against plagiarism and malicious damage. The compilation process translates the program as whole and removes

redundancies resulting in more efficient code. The disadvantage is that compiling can take some time and is testing is required at each stage during the development process than this may waste time. As well, a program will not compile if an error has been found.

INCREMENTAL COMPILATION

An incremental compiler compiles each line of the source code as it is entered and adds it to the object file. This has the advantage of an interpreter in error checking and the speed of a compiler in execution.

INTERPRETATION

When the program is translated line by line it is called interpreting. Interpretation ensures that there is no memory or resource conflicts and allows syntax errors to be identified immediately. Examples include the Python Interactive Interpreter, but this method is the slowest.

The Translation Process

The translation process involves four stages:

1. **Lexical Analysis:** uses the delimiters (symbols such as colons or CR that indicates the end of a syntactic unit) in the source code to break the program into single commands. Each character in the line is scanned and redundant characters removed. Different types of elements are labelled with a token. If there is an error in the syntax then translation cannot continue.
2. **Syntactic Analysis:** examines and tests the validity of the relationships between the elements. It is an examination of whether the identified elements of a statement are combined together in a way that is legal according to the syntax of the language.

Advantages and Disadvantages of Each Method

	Advantages	Disadvantages
Compiling	<ul style="list-style-type: none"> Fast execution of object code Code is transportable Security of code Allows optimisation of object code Permits sharing of resources and reduces redundancy by using run-time libraries 	<ul style="list-style-type: none"> Modification of the problem means it has to be completely recompiled Compiling stops when a syntax error is encountered Object code duplicates space
Interpreting	<ul style="list-style-type: none"> Simple, rapid production of object code Immediate detection of syntax errors No stored object code, so no duplication of resources Run-time errors can be detected and 	<ul style="list-style-type: none"> Slow execution speed Poor efficiency – the same lines will be reinterpreted each pass of a repetition No reusable object code is produced

	corrected quickly Interactive programming	
--	--	--

PROGRAM DEVELOPMENT TECHNIQUES IN SOFTWARE SOLUTIONS

Structured Approach to a Complex Solution, Including:

The structured approach to software development involves planning a solution by stepwise refinement – that is solving an overall problem by dividing it into smaller, more manageable parts. The use of modules is the preferred method of achieving a solution. There are two methods that use modules: top down design and bottom up design.

In top down design, the problem is refined into small sub-problems, which are then broken down again into smaller problems. The program that controls these modules is written and tested first. Code that controls the next level of modules is then created and tested until code for the lowest level is written.

In bottom up design, the problem is solved starting with the modules at the lowest levels. Each module is written as a self-contained unit, which can be run and tested independently. The modules are then put together to form the end program. This approach is usually suited to OOP or event driven programming.

ONE LOGICAL TASK PER SUBROUTINE

When testing a program, it is usual to test the programming code of these modules independently rather than delay testing until the whole program is complete. However, while a module may be a part of the solution, in most cases it is not possible to run the module as a stand-alone program. A driver is a temporary section of code that is created to test an individual procedure or module by calling it up and executing it.

STUBS

The use of a stub is particularly important in top-down program design and testing, as a stub allows a section of code to be tested even though it relies on other modules of the program. A stub is effectively a temporary subprogram which usually contains very little code such as displaying a message to indicate that a certain program has been successfully called or to temporarily assign variables.

FLAGS

A flag is a marker placed in the code to signal a change in status. They are usually Boolean variables that record whether a certain condition has been met and are very useful debugging tools.

ISOLATION OF ERRORS

There are other methods that can be used to determine where errors occur in a program such as program traces, breakpoints and commenting out sections of code.

DEBUGGING OUTPUT STATEMENTS

Debugging output statements are temporary lines of code added to a program or module to display the value of variables at strategic places in the program. This allows the programmer to compare actual values with expected values. Similar to stubs and flags.

ELEGANCE OF SOLUTION

An elegant solution is one that is efficient in its use of resources and data structures such as using a loop to iterate over lists instead of manually hard coding in each input. This also allows for much easier changes if the specifications in the future change.

WRITING FOR SUBSEQUENT MAINTENANCE

A well-structured solution with clear documentation makes the task of maintenance a much simpler process than trying to understand a poorly documented and unstructured program.

The Process of Detecting and Correcting Errors, Including:

SYNTAX ERRORS

A syntax error is an error detected by the compiler when it is unable to match items in the source code to syntactic entries in the language. The syntax analyser takes the lexical units (tokens or words) from the lexical analysis stage and from the constructs hierarchical structures called parse trees.

LOGIC ERRORS

A logical error means that the program will perform an incorrect action or give the wrong output. They are usually caused by an error in the design of the program so that although the program will run, it will produce an incorrect output. Logic errors are detected through careful checking use selected test data (desk check etc).

PEER CHECKING

Peer checking is a process in which programmers not involved with the original design are asked to check the logic of an algorithm or program.

DESK CHECKING

In desk checking, test data is used to compare the actual result with expected results. The programmer uses a structured walkthrough using pen and paper to step through the code to find where the error has occurred.

USE OF EXPECTED OUTPUT

In choosing the test data, the following needs to be considered:

- Should test a range of values
- Values between expected limits (ie: middle values)
- Values on the limit of expected values (ie: boundary values)
- Non-valid values.

RUN TIME ERRORS, INCLUDING:

Run-time testing involves the use of live data – real that is supplied by the client. Run-time errors are problems that appear when a translated program is executed.

ARITHMETIC OVERFLOW

Overflow errors involves the incorrect use of data types and data structures. Arithmetic overflow errors occur when the computer is unable to store or process the number of digits being manipulated. Floating point overflow may occur when performing calculations using large numbers. The result may be either out of the range of values able to be stored or the precision may be inaccurate. Range errors occur when a value is too large to be stored using the number of bits allocated.

DIVISION BY ZERO

Errors may also be caused by performing an undefined arithmetic operation such as dividing a number by 0 (which is arithmetically impossible!)

ACCESSING INAPPROPRIATE MEMORY LOCATIONS

Errors will also occur if the program attempts to access an inappropriate or privileged memory location.

The Use of Software Debugging Tools, Including:

Debugging is the process of removing bugs from your program. Translation will detect syntax errors however other errors need to be removed using a systematic approach. This process requires the use of good test data and desk checking. Many programming languages allow the setting of break points and the tracing of variables, otherwise, the method of using stubs, flags and debugging statements can also be used.

USE OF BREAKPOINTS

Setting breakpoints is the simple technique of stopping the program part way through the execution. This allows the programmer to examine the contents of variables or to only run certain portions of code.

RESETTING VARIABLE CONTENTS

The value stored in a variable can be changed over the duration of the execution of the program. It is helpful to change and print the contents of variables to determine if in fact that a certain portion of code is being executed.

PROGRAM TRACES

When tracing a program, the actual line of the program that is being run will be displayed instead of this output. The values of variables can be seen which the program is running. This is particularly useful when running nested loops or complicated selections.

SINGLE LINE STEPPING

Summary Notes

Single line stepping allows the programmer to step through the code, one line at a time to ensure that each line of the source code is being executed correctly. The values of variables can be traced as well as the flow of control throughout the program.

DOCUMENTATION OF A SOFTWARE SOLUTION

Forms of Documentation, Including:

PROCESS DIARY

Design documentation specifies the development process and the specifications for the design of the software. A process diary is used to document program development. Programmers and system developers monitor the progress of a new system and document changes in the system as they occur by using the process diary. Also known as a logbook.

USER DOCUMENTATION

The main volume of documentation is user documentation. This should be clear and non-technical. It includes such documentation as: installation guides, troubleshooting guides, online help, user manuals and reference manuals.

SELF-DOCUMENTATION OF THE CODE

Intrinsic documentation refers to the use of a self-explanatory, easy to follow coding style. This includes such facets as clarity of layout using indentation to show repetition and selection structures, clear declaration of variables, use of meaningful identifier names that are self-explanatory.

TECHNICAL DOCUMENTATION, INCLUDING SOURCE CODE, ALGORITHMS, DATA DICTIONARY AND SYSTEMS DOCUMENTATION

Source code is the programming code that makes up a particular program. It is sometimes helpful to print out source code to make it easier to read. This allows programmers a greater depth of understanding as to the how the program works and how to solve problems.

Algorithms are primarily used during program design to provide a clear description of the steps carried out to solve the problem. They allow maintenance programmers to determine the structure of the program.

Data dictionaries describe each data item as fully as possible. It usually includes the data names, length, data type, description and an example and/or default value.

Systems documentation should provide a description of the operation of the program including the use of subprograms and any special features of the code. It should include documentation on how to configure the hardware and software required.

DOCUMENTATION FOR SUBSEQUENT MAINTENANCE OF THE CODE

It is important to document a program in a way that will make it easy to modify if a bug is found or an improvement is required. Internal documentation is documentation that forms part of the programming code. Intrinsic documentation and comments in the code are examples of internal documentation. Changes made during maintenance should be documented for future reference.

Use of Application Software to Assist In the Documentation Process

Application software is used to structure and clearly present documentation. Application software ranges from simple word processors with a template to CASE tools that will automatically generate portions of the documentation

USE OF CASE TOOLS

CASE tools provide programmers with a range of tools to aid in the development process including documentation. CASE tools can sometimes automatically generate some forms of documentation such as data dictionaries or provide a structured way of entering data.

HARDWARE ENVIRONMENT TO ENABLE IMPLEMENTATION OF THE SOFTWARE SOLUTION

Hardware Requirements

During development, it is important to determine the hardware specifications of the computers that will be used to run the program such as Pentium 4+, Intel Core2 Duo, ARM Cortex.

MINIMUM CONFIGURATION

Commercial software products will usually have a minimum configuration on which the software will run reliably. This will usually specify:

- What type of computer: Pentium, Macintosh
- The amount of hard disk space required for the program
- The amount of RAM required for the program to run
- The OS under which the software will run: Windows XP+, OS X 10.5

POSSIBLE ADDITIONAL HARDWARE

As programs become more complex, they require greater hardware requirements. This could come in the form of needing more RAM or a discrete graphics card. Some programs may also require other IO devices such as a barcode scanner or Bluetooth connectivity.

APPROPRIATE DRIVERS OR EXTENSIONS

Extensions and drivers are software programs that enable common tasks to be accessed from many programs. Drivers and extensions act as a interface between software (ie: the OS) and the IO device being accessed. For example, a driver that allows programs to output to a printer would convert the output signals into a format that is understood by the printer. It is sometimes necessary to install additional drivers as the OS only comes with a limited number.

EMERGING TECHNOLOGIES

Hardware

Computer hardware is constantly improving and new hardware devices are being developed that allows users to operate in completely different ways. Processor speed, storage and

communication speeds have increased dramatically and new markets are constantly being created particularly with the advent of internet-enabled mobile devices such as the iPhone and the movement to the cloud. Furthermore, Moore's law states that computer power will double every 18 months and costs for such technology is dropping rapidly as we refine manufacturing techniques.

Software

Wirth's law states that software gets slower faster than hardware gets faster. As we continue to improve computer hardware, applications continue to be developed that require every increasing processing speeds. Compare the CLI programs of 10-15 years ago with today's modern 3D games and photo/video manipulation programs. Current hardware limitations will restrict the type of software that can be developed. There has also been an emerging trend in mobile operating systems with the iPhone and iOS revolutionising the smart phone market as well as the emergence of Android that provides users with an experience ever close to that of a desktop computer (multi-tasking, video conferencing, games, email, word processing etc).

Their Effect On:

HUMAN ENVIRONMENT

Internet-enabled mobile devices allow for greater access and portability to information however with GPS becoming standard in phones and services such as social networking broadcasting your location to your social network, it raises concerns over privacy with people being able to effectively track your location through social networking. Geolocation could also allow advertisers to target you such as sending you a text message if you are close to a restaurant.

DEVELOPMENT PROCESS

With the emergence of new technologies, the types of programs being developed will change. Devices such as the iPad blur the line between mobile, desktop and laptop. They provide users with an experience that is as close to a computer as a mobile device ever has done before. With the Smartphone becoming more and more accessible to the common user, developers have a whole new market than they previously had before when a Smartphone was targeted as a business user.

TESTING AND EVALUATION OF SOFTWARE SOLUTIONS

TESTING THE SOFTWARE SOLUTION

Comparison of the Solution with the Original Design Specifications

It is no use if the program runs but does not meet the original design specifications for which it is meant to solve! Developers must continually ensure that the original specifications are being met in the product being developed.

Generating Relevant Test Data for Complex Solutions

Test data should be generated during analysis and design stages of the software development cycle when the algorithms are created to determine IPO and ensure the correctness of an algorithm or program.

Levels of Testing

Test data must be sufficient to ensure the program is completely operational and free from logic errors. To do so, a set of test data must test:

All parts of a program: Test data must test every part of a program, including the mainline of the program and any modules used by the program.

Each path of execution: Must test every logical pathway in the algorithm. Data must be chosen which will execute every possible alternative in a selection control structure. Should also test the termination and correct exit of repetition control structures.

Boundary conditions: Boundary conditions are the values of variables or expressions that determine the choice of available options to be taken. Values between boundary conditions are called middle values.

UNIT OR MODULE

It is possible to develop computer programs that can be easily tested. In structured programming, each module performs a simple task. Subprograms can be tested as a black box where only the inputs and outputs are checked and the processes ignored or they can be tested as a white box where the algorithm of the subprogram is understood and each path of execution tested appropriately.

A driver program may be developed to test modules in a program, this substitutes for the main program calling the subprogram and supplying the necessary values for any variables.

PROGRAM

To completely test a program, it is a minimum requirement that the test data tests each logical pathway and program branch that can be entered. This may not always be feasible and in such cases test data should be selected which tests scenarios of expected and possible system use.

Once all modules have been tested, it is necessary to ensure that they interface with each other correctly and that the main program from which the modules are called functions as expected. All user interfaces should be tested during program testing.

SYSTEM

During system testing, the program is tested in a variety of operating environments. The effect of hardware, operating systems and other software may create errors that have not been previously detected. The system testers treat the program as a black box.

The Use of Live Test Data to Test the Complete Solution:

Live data is real data supplied by the client that will be used on the new system. It is only by using the new system as it is intended to run that bugs will often be detected. Live test data is used to ensure that a program works under real-life conditions. Live test data will help to test the performance of the software and hardware under expected loads. Stress testing involves increasing the load on a program in an attempt to make it fail.

LARGER FILE SIZES

A program developed to access files should be tested with a range of file types and files of varying sizes – a large file may create errors due to the method of storage for example.

MIX OF TRANSACTION TYPES

A program may process data correctly when it is entered in a consistent and logical way. In a live situation, data may not be processed in the sequence expected by the software developer. A different variety of different transaction types and sequences of data entry should be tested with live data.

RESPONSE TIMES

Live data is used to ensure that the system response times are appropriate. Increasing the frequency at which test data is processed may create problems that were otherwise hidden. Software developed on the most modern computer may respond very quickly to commands from the user however, in real life, it may be slowed down by older computers and slower networks.

VOLUME DATA

It is important to test a system using expected loads however it is also important to test the system under heavy loads as, over time, the loads on the system may increase.

INTERFACES BETWEEN MODULES

The interface of a module must be tested to ensure that data is correctly transferred to the module and also that data and control is transferred correctly back to the part of the program from which the module was called. Interface tests will ensure that the correct numbers of parameters are sent to and from the module and that the format is correct.

COMPARISON WITH PROGRAM TEST DATA

Live data is extremely helpful in detecting errors that will be frequently encountered in a program. The use of live test data also tests how the system responds to the procedures carried out by the users of the program.

Benchmarking

A benchmark is a standard against which performance of a computer program can be assessed. This standard may be a set of performance criteria that should be met by the program. Benchmarking can be used to determine the “real” improvements when upgrading a system.

Quality Assurance

Quality assurance is a set of procedures used to certify that a generated product meets specified criteria with respect to quality and reliability. Quality control involves periodically performing inspections, reviews and tests on the system being developed. Regression testing is the process of re-testing a program to ensure that the changes made have not affected a previously working part of the program.

After a programmer has tested the program, end users may perform alpha testing in a controlled environment to see if the requirements are met. Following this, beta testing may be performed at the client’s own site under normal working conditions. Once a program passes beta testing it is ready for acceptance by the client.

REPORTING ON THE TESTING PROCESS

A test specification or test plan is often created to describe the methods that will be used to test the program. Software testers can follow the stages outline to ensure that all aspects of the program have been tested. During the testing process, documentation should be created the software developer and others testing the program to describe the tests performed and errors corrected.

Documentation of the Test Data and Output Produced

Test data should be used for a purpose. It is of limited use if test data is randomly selected. A test data table should be created to show the test data to be used and the reason why this item of test data was selected. A desk-check table is used to document the test data used and compare the expected output with the actual output of the algorithm or program.

USE OF CASE TOOLS

Computer aided software engineering tools provide a number of helpful tools to test a computer program as well as tools which simplify the documentation process. Computer aided prototyping tools allow software developers to simulate software performance under real-life situations. Documentation of the testing process is simplified through the use of test management tools.

Communication with Those for Whom the Solution Has Been Developed, Including:

The testing process should never be carried out in isolation. The software developer often approaches testing with the aim of demonstrating how well the program works. This is why it is important to have other people involved in the testing process. It is essential that the end user is provided with the opportunity to evaluate the solution that has been developed.

TEST RESULTS

Results of the testing process should be summaries for the user. This empowers the user and provides an opportunity for them to evaluate and discuss the function of the new system.

COMPARISON WITH THE ORIGINAL DESIGN SPECIFICATIONS

After testing, users are given an opportunity to use the program and test that it meets the initial requirements of the problem. After a customer has approved a program, it can be released to the market.

MAINTENANCE OF SOFTWARE SOLUTIONS

MODIFICATION OF CODE TO MEET CHANGED REQUIREMENTS

Software maintenance involves making changes to meet change requirements or correct problems in the software solution. Is it important to create documentation to enable others to understand the structure of the program and the commands that are executed.

Modification involves: problem definition, planning the solution, building the solution, checking the solution and modifying the solution.

Identification of the Reasons for Change in Code, Macros and Scripts

Maintenance may be carried out because of an error being detected, changes in the program's operating environment or because of increased functionality required by the user.

Location of Section to Be Altered

After an error has been detected or it is determined that a change is to be made, the programmer must determine which section of the source code must be altered to allow for the change. Well structured code and extensive documentation simplifies the process of isolating and correcting errors.

Determining Changes to Be Made

Once the section of code requiring change has been located, the programmer must decide how to change the program in a way that will not create new problems. One technique is known as the software patch. This is a small piece of code that is added to the compiled program that avoids the error in some way. Once all errors have been detected, a new version of the program will be released.

Over time, small patches can make a program quite inefficient and slow. Software re-engineering is the process of using modern technology to rebuild or redesign an out of date program.

Implementing and Testing Solution

Modifications made to an existing program must be tested to ensure that the changes are not going to negatively affect other areas of the program. Regression testing is the process of re-testing a program to ensure that the changes made have not affected a previously working part of the program.

DOCUMENTATION OF CHANGES

Documentation should be created during every stage of the program development cycle.

Source Code, Macro and Script Documentation

Modification of Associated Hard Copy Documentation and Online Help

Use of Case Tools to Monitor Changes and Versions

DEVELOPING A SOLUTION PACKAGE

DESIGNING AND DEVELOPING A SOFTWARE SOLUTION TO A COMPLEX PROBLEM

DEFINING THE PROBLEM AND ITS SOLUTION, INCLUDING:

Defining the Problem

IDENTIFICATION OF THE PROBLEM
IDEA GENERATION
COMMUNICATION WITH OTHERS INVOLVED IN THE PROPOSED SYSTEM

Understanding

INTERFACE DESIGN
COMMUNICATION WITH OTHERS INVOLVED IN THE PROPOSED SYSTEM
REPRESENTING THE SYSTEM USING DIAGRAMS
SELECTION OF APPROPRIATE DATA STRUCTURES
APPLYING PROJECT MANAGEMENT TECHNIQUES
CONSIDERATION OF ALL SOCIAL AND ETHICAL ISSUES

Planning and Design

INTERFACE DESIGN
SELECTION OF SOFTWARE ENVIRONMENT
IDENTIFICATION OF APPROPRIATE HARDWARE
SELECTION OF APPROPRIATE DATA STRUCTURES
PRODUCTION OF DATA DICTIONARY
DEFINITION OF REQUIRED VALIDATION PROCESSES
DEFINITION OF FILES — RECORD LAYOUT AND CREATION
ALGORITHM DESIGN
INCLUSION OF STANDARD OR COMMON ROUTINES
USE OF SOFTWARE TO DOCUMENT DESIGN
IDENTIFICATION OF APPROPRIATE TEST DATA
ENABLING AND INCORPORATING FEEDBACK FROM USERS AT REGULAR INTERVALS
CONSIDERATION OF ALL SOCIAL AND ETHICAL ISSUES
APPLYING PROJECT MANAGEMENT TECHNIQUES

SYSTEMS IMPLEMENTATION

Implementation

PRODUCTION AND MAINTENANCE OF DATA DICTIONARY
INCLUSION OF STANDARD OR COMMON ROUTINES
USE OF SOFTWARE TO DOCUMENT DESIGN
TRANSLATING THE SOLUTION INTO CODE
CREATING ONLINE HELP
PROGRAM TESTING
REPORTING ON THE STATUS OF THE SYSTEM AT REGULAR INTERVALS
APPLYING PROJECT MANAGEMENT TECHNIQUES

Summary Notes

ENABLING AND INCORPORATING FEEDBACK FROM USERS AT REGULAR INTERVALS
COMPLETING ALL USER DOCUMENTATION FOR THE PROJECT
CONSIDERATION OF ALL SOCIAL AND ETHICAL ISSUES
COMPLETING FULL PROGRAM AND SYSTEMS TESTING

Maintenance

MODIFYING THE PROJECT TO ENSURE AN IMPROVED SOLUTION

THE SOFTWARE DEVELOPER'S VIEW OF THE HARDWARE

REPRESENTATION OF DATA WITHIN THE COMPUTER

A computer can only process binary data (0 or 1). Humans work in decimal numbers and hence we need to convert decimal to binary and vice versa. There are limitations on representation of data which the programmer needs to determine.

- The range of acceptable integers (16, 32 or 64 bits)
- Maximum size of a real number and how many decimal places that can be used.
- The disk space that can be used
- The amount of Ram available for data storage.

Character Representation, Namely:

Characters are letters, digits or other symbols from the keyboard. They have to be changed into binary numbers to be stored and processed by the computer. Characters can be grouped together to form strings to allow for easier manipulation and storage (ie: and array of individual characters)

ASCII

ASCII: American Standard Code for information Interchange.

Allows data to be stored in binary form using 7 bits to encode 2^7 (128) different characters. Or EBCDIC which uses 8-bits can be encoded.

HEXADECIMAL

Hexadecimal numbers are base 16 (0-9, A-F). One hexadecimal digit can be used to represent 4 binary digits. One byte (8-bits) can be written using two hexadecimal numbers. You can break a byte in half (nibble) and convert these halves into hexadecimal.

Integer Representation, Including:

Integers are positive or negative whole numbers. They are usually stored as 32-bits.

SIGN AND MODULUS

Sign and modulus: this method uses the left-most bit to represent the sign (0=+, 1=-). The remaining bits give the modulus or magnitude of the number.

ONE'S COMPLEMENT

One's complement: depicts negative numbers by replacing all the ones in the positive form with zeros and all the zeros with ones

TWO'S COMPLEMENT

Two's complement: represents a negative number by adding 1 to the one's complement of the number.

Representation of Fractions, Namely:

Binary fractions are represented as negative powers of 2 such as $2^{-1}=1/2$ etc. The decimal number 0.3125 can be converted to binary by repeatedly multiplying by 2 and removing the digit to the left of the decimal place and then reading off the whole numbers.

FLOATING POINT OR REAL

Real numbers are presented as floating point numbers made up of a sign bit, mantissa and exponent. They are usually stored as 32-bits. Real numbers allow for a greater range of numbers to be represented however the accuracy of large numbers may not be sufficient because there is a limited number of decimal places. Double precision reals (using 64-bits) are often used to represent numbers more accurately.

Problems of inaccuracy can occur when using real data types. Some numbers cannot be accurately represented as exact binary fractions which can lead to truncation where the number of cut off at a certain number of decimal places without rounding it up or down. This can cause errors in high precision calculations.

BOOLEAN DATA

Boolean data only has two values – either true or false. Data can be stored as only one bit and it is the simplest data type.

Binary Arithmetic, Including:

ADDITION

Addition is the basis of all arithmetic operations inside the computer and is carried out by the special ALU. The rules are simple:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10 \text{ (ie: 0 carry the 1)}$$

SUBTRACTION USING TWO'S COMPLEMENT REPRESENTATION

Rules:

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ with 1 being borrowed from the next higher bit position}$$

Subtraction is too hard for computers to perform so instead they “add negatives”. Essentially they add the first number to the two's complement of the second number

MULTIPLICATION, SHIFT AND ADD

The multiplication of two binary numbers is performed using addition. Each successive bit of the multiplier is looked at. If it is a 1, then the multiplicand is written down. If it is a 0 then all

zeros are written down. The numbers in successive lines are shifted one place to the left of the previous number and then they are all added together.

DIVISION, SHIFT AND SUBTRACT

Division of binary numbers can be carried out using long division. This process involves:

- Subtracting the divisor
- Shifting the next digit into the remainder until the divider can be subtracted again.
- Repeat above

ELECTRONIC CIRCUITS TO PERFORM STANDARD SOFTWARE OPERATIONS

Computers perform operations on binary data. In a computer, a 0 represents off and a 1 represents on.

Logic Gates, Including:

Logic gates are hardware circuits that produce a 1 or a 0 output signal if the input requirements are satisfied. Logic gates are usually implemented in the computer using ICs.

THE AND FUNCTION

Involves two input variables to produce one output. If both input variables are 1, then the output will be 1 otherwise the output will be 0.

THE OR FUNCTION

Involves two input variables to produce one output. If either or both of the input variables are 1, then the output will be 1.

THE NOT FUNCTION

Involves a single input and output. The state of the output is opposite the state of the input.

THE NAND (NOT AND) FUNCTION

Is the inverse of the AND function ie: the output is 1 except if both inputs are 1.

THE NOR (NOT OR) FUNCTION

Is the inverse of the OR function ie: the output will be 1 if both of the inputs are 0

THE XOR (EXCLUSIVE OR) FUNCTION

Will output a 1 if either of the inputs are 1 but NOT both.

Truth Tables

Name	Description	Logic Gate	Truth Table		
AND	The output value of C is 1 when both input values A and B are 1		Input		Output
			A	B	$C=A \cdot B$
			0	0	0
			0	1	0
			1	1	1
OR	If either of the input values are 1 then the output will be 1.		Input		Output
			A	B	$C=A+B$
			0	0	0
			0	1	1
			1	1	1
NOT	The state of the output is the inverse of the input		Input		Output
			A		$C=A$
			0		1
			1		0
NAND	Not AND. The output value C is 1 except where both A and B are 1		Input		Output
			A	B	$C=(A \cdot B)$
			0	0	1
			0	1	1
			1	1	0
NOR	The output C is 1 when both inputs A and B are 0		Input		Output
			A	B	$C=A+B$
			0	0	1
			0	1	0
			1	1	0
XOR	The output C is 1 except when both inputs A and B are the same		Input		Output
			A	B	$C=A+B$
			0	0	0
			0	1	1
			1	1	0

Circuit Design Steps

IDENTIFY INPUTS AND OUTPUTS

The number of inputs into the circuit must be determined, as well as the number of output.

IDENTIFY REQUIRED COMPONENTS

The circuit designer must then determine which logic gates will be needed.

CHECK SOLUTION WITH A TRUTH TABLE

Once a possible solution has been developed, it must be tested to ensure it works as required.

EVALUATE THE CIRCUIT DESIGN

Specialty Circuits, Including:

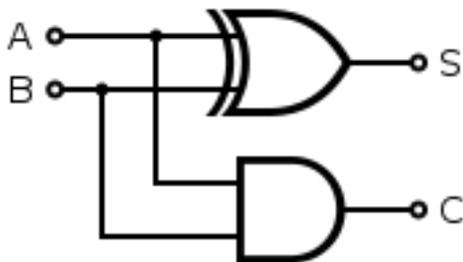
All circuits within the computer are made from a combination of the basic logic gates. Circuits can be designed to perform addition, subtraction and comparisons. Combination circuits produce instant results as determined by a combination of logic gates. Sequential circuits contain memory cells as well as logic gates.

HALF ADDER

A Half adder is a combinational circuit that performs the addition of two bits.

Input		Output	
X	Y	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The half adder is made by combining the AND and XOR gates

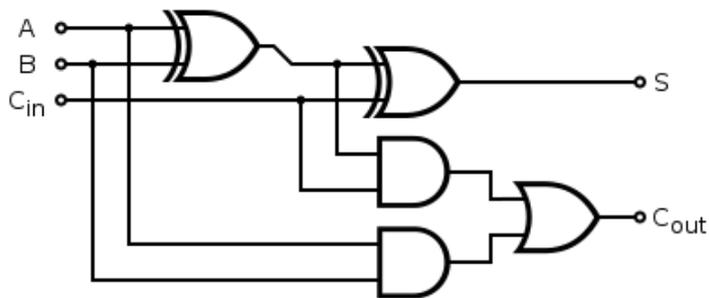


FULL ADDER

A full adder is a combinational circuit that can be used to add three binary digits. It consists of three inputs and two outputs.

Input			Output	
X	Y	Z	Carry	Sum
0	0	0	0	0
0	1	0	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

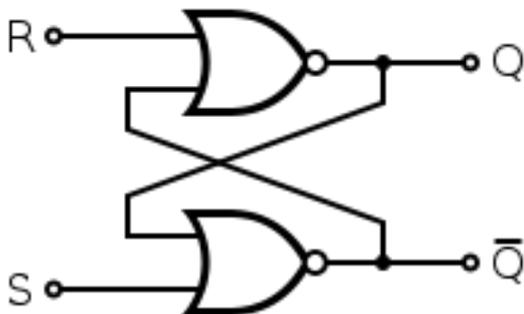
A full adder can be implemented by combining two half adders.



FLIP-FLOPS AS A MEMORY STORE

A flip-flop is a circuit that can store a binary value as long as power is supplied to the circuit. A flip-flop can store the binary values 1 and 0. When a value has been set, the flip-flop remains in this state until it is told to change states.

A flip-flop has two outputs, A and NOT Q, and two inputs, set (S) and reset (R).



If the set input is 1 and the reset input is 0, gate 2 (bottom) will have an output of 1. But if either of the NOR inputs are 1, then the output has to be 0 regardless of the other input.

PROGRAMMING OF HARDWARE DEVICES

For a computer to be able to communicate with other computers or with input and output devices, the data transmissions must be structured. The way that the data is structured varies with different communications protocols (a set of rules that governs the transmission of data between computers).

The Input Data Stream from Sensor and Other Devices

Data read into the computer is accessed as a sequence of zeros and ones. The computer must be programmed to interpret these bits of data to produce useful information. A protocol must be established before two hardware devices can exchange data. The protocol used will determine the rules for structuring data packets.

HEADER INFORMATION

The header contains a set sequence of bits to indicate the start of a block of data. The header may also specify how much data is to be transmitted. Headers often contain error-checking data such as parity bits or cyclic redundancy check characters.

DATA CHARACTERS

The body of data will contain the instructions that are to be processed by the CPU.

TRAILER INFORMATION

The trailer contains data bits to indicate the end of a block of data. The use of a trailer and a header help the CPU to manage the data that it receives. Data for error checking may be included in the trailer (CRC, hash etc)

CONTROL CHARACTERS

Control characters permit the checking and correct reassembling of a message. A packet contains control characters to enable correct interpretation of the packet.

HARDWARE SPECIFICATIONS

The input data stream will be structured according to the standards or protocol followed by the hardware developer. Driver or extension software such as DLL's may be required to enable an operating system to communicate with a particular hardware device.

DOCUMENTATION

Technical documentation accompanying hardware should specify the communication protocols used for that device and the format of data, at the very least; the documentation should contain a description of the protocol used.

Processing Of Data Stream

THE NEED TO RECOGNISE AND STRIP CONTROL CHARACTERS

The protocol used for data transfer will determine how control characters are recognised and stripped from the body of data.

COUNTING THE DATA CHARACTERS

Header fields may contain information concerning how much data is being transmitted.

EXTRACTING THE DATA

The hardware device connected to the CPU is responsible for extracting and using data sent to it from the computer.

Generating Output to an Appropriate Output Device

The output data stream functions similarly to the input data stream. The CPU sends information to connected devices. Output includes acknowledging that data has been received successfully and data packets to print information or control peripheral devices etc.

Output data packets must specify the device to which the information is being transmitted.

REQUIRED HEADER INFORMATION

REQUIRED CONTROL CHARACTERS

DATA

REQUIRED TRAILER INFORMATION

Control Systems

A computer-controlled system is a combination of hardware and software designed to instruct that computer to control a connected device.

Sensors (input): Sensors are used to capture information from the environment. They are a form of transducer as they convert one form of energy into another.

Instructions (input as well): Instructions control the actuators and effects and allow the controller to interpret the input from the sensors in an appropriate way.

Process Controller (processing): may be a computer or a specially designed circuit built into an appliance to allow stand-alone applications.

Effectors and actuators (output): perform the actions of modifying the environment such as motors, relays etc.

RESPONDING TO SENSOR INFORMATION

Sensors are used to record both analogue and digital data. Analogue signals are continuously variable with the signals having any value within a particular range. Digital signals have discrete levels only (0 or 1)

SPECIFYING MOTOR OPERATIONS

Printer Operation

CONTROL CHARACTERS FOR FEATURES, INCLUDING PAGE THROW, FONT CHANGE, LINE SPACING

Specialist Devices with Digital Input and/or Output