Year 12 Software Notes

GOOD LUCK EVERYONE FOR TOMORROW!!! LAST EVER SDD EXAM EVER LETS GOOOOO! :3

Impact of Software

Inappropriate data structures

Y2K Problem

Early computers had limited memory and RAM was very expensive. To save memory, programmers that developed operating systems had reduced the storage of dates to a two – digit integer, reducing storage needs by half. As the year 2000 approached, there was a great concern about what would happen to software that governments, banks and businesses depended on. Created numerous computers crashes and incorrect calculations.

Malware

Software designed to damage or perform unwanted action on computer systems. Due to the constant threat posed by malware, security measures including anti-virus, anti-malware and anti-spyware products are implemented to protect systems from harm.

- Viruses can replicate and spread upon the execution of seemingly legitimate software
- Worms attempt to slow a system by installing themselves on hard-disks and consuming space by replication – often occurs in RAM due to security flaws with OS
- Trojan horses pose as legitimate software and once executed they cause damage in the system. Used to spread malware to provide access to another user. Employs use of botnets to slow down competitors – denial of service (DoS) attack.
- Spyware gains the user's information and provides it to another party, usually used for identity theft.

Reliance on software

The software development industry has a huge responsibility to ensure all these systems are reliable and perform their functions accurately. Includes household appliances, to motorised transport, utilities and even government authorities such as the Tax Office and hospitals.

Social Networking

Online virtual community where people share thoughts and activities with peers. Concerns regarding social networking, particularly privacy. Due to the encouragement of information sharing, there is a chance of identity theft and 'stalking'. It is also common for bullying to occur, particularly involving young people.

Cyber Safety

Cyber safety is about minimising the risks of cyber dangers, particularly children. While security software and firewalls attempt to protect the user, it is necessary for self-protection methods to be utilised should the user wish to remain safe. Government initiative (cybersmart.gov) informs users about cyber safe practices and procedures:

- Location-based services
- Unwanted contact
- Cyberbullying
- Online friends
- Your digital footprint
- Online purchasing
- Identity theft

Evaluating information through the internet

Due to the unregulated nature of the Internet, massive amount of information is available – often without a stated author. This leads to concerns regarding the credibility of the information. These questions should be asked to evaluate a source:

- Who is the author?
- Is the information up to date?
- Who is intended audience?
- Is information accurate and unbiased?
- What is the purpose of the information?

Rights and Responsibilities of software developer

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
- Responsibility to develop and use software in a social and legally ethical manner.

Software developers invest time and money into the development of their products. 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.

Intellectual property

There is often ambiguity on who is the 'creator of the software' as:

- Software product is developed in a coding language, which was developed by another company
- A team of developers is utilised to create software, often outsources
- Modules of source code can be purchased from third party,
- Artwork and images are created in a graphic design company

Software developers need to acknowledge all the 'creators' and ensure all compensation is paid.

Quality

Quality assurance is used to ensure standard of quality are being met by software developers. It is often difficult for software developers to maintain high quality due to financial/time constraints. Customer's perceive quality through their expectations. Involves correctness, reliability, efficiency, integrity, useability, maintainability, flexibility, testability, portability, interoperability.

External factors affecting quality include hardware, operating system, other software and run time errors.

Hardware – To ensure maximum reliability of their product, software application developers should test their product with several different hardware configurations

Operating system- software products should interface correctly with specifications of operating system. Software should also be forward compatible as OS is always being updated

Other software – software should not adversely react to each other to create unwanted results

Run-time errors – All applications developed should include error checking built into the code. Developers must include routines that will deal with possible errors in execution. For example, developer would include save procedure when detecting a fatal error.

Response to problems

Mechanisms are needed to assist in the identification of errors and their resolution, such as support departments and severity classifications

Code of Conduct

Set of standards by which software developers agree to abide. These standards increase the quality of developed software across the industry. Can be expelled if not adhering to code of conduct.

Malware

Developers have a responsibility to both do not develop malware and to constantly check for malware in their workplace environments. Users have the right to expect their software to be free of malware

Ergonomic issues

Determining user needs about their work routine assist in the user interface design which allows users to work efficiently including consistency of elements, colour, fonts and alignment properly. User testing is used to evaluate the ergonomic design of software.

Inclusivity Issues

Responsibility to ensure software is accessible to anyone regardless of their differences including cultural differences, economic market, gender and disability.

Privacy

Users have the right to know if their information is being held. Some organisations need to legitimately access sensitive information and therefore users are protected under the Privacy Act 1988.

Software Piracy and Copyright

Illegal copying and use of software. Occurs when intellectual property rights of the developer are infringed. 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.

Intellectual property is personal ownership of the creative ideas that develop from an individual's mind or intellect. Includes patents, trademarks, trade secrets and confidential business information.

Plagiarism is appropriating or imitating someone's ideas and manner of expressing them and claiming them as your own. Plagiarism in software is ambiguous and to avoid infringing another person's intellectual property and sources should be acknowledged and compensated.

Copyright Laws

Purpose of these laws is to provide economic incentives for creative activity. Copyright protects the expression of ideas rather than the ideas themselves. They give the owner the sole right to reproduce their own work. Copyright Amendment Act 2006. Lasts for 70 years.

Classification of software

Commercial

- Purchasing licence to use software
- Licensing company owns product and copyright
- Software covered by copyright
- One archival copy can be made
- Reverse engineering and de-compilation not allowed

Open Source

- Available to all to modify and redistribute, modified products must be released through same unrestricted open source licence. Encourages collaboration and sharing of ideas.

Shareware

- Covered by copyright
- Distributed for trial use before purchase
- User must pay to continue to use software after trial period

Public Domain

- Freely available for copying and modification
- Copyright has finished

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 is someone else does.

De-compilation is the process of translating object code (machine code) into code that is more easily studied by a programmer.

Software Licences

- Enforced by law
- Protects developer's ownership of the software that they have created
- Licence conditions – determine what can be done with software, many developers include a compulsory reading and acceptance of the EULA before installing can continue.
- Licence – formal permission or authority to use a product
- Agreement – mutual agreement or contract between parties
- Term – period the agreement is in force
- Warranty – assurance of some sort, a guarantee
- Limited use – restricted use of product
- Liability – an obligation or debt because of some sort of consequence
- Program – refers to computer software

Ownership versus licensing – purchasing media that contains computer software does not mean you own the software; you have been sold the right to use the software under certain conditions verified in your licence.

Current and emerging technologies used to combat software

- Non copiable datasheet
- Disc copy protection
- Hardware serial numbers – if software and hardware do not match, program will not run
- Site license installation counter on a network
- Registration code for software
- Encryption keys to scramble/unscramble data/ software
- Back to base authentication

Use of Networks

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.

Use of networks by software developer

Access to resources – graphical assets, source code, third party libraries

Ease of communication – networks allow developers to communicate with each other

Productivity – collaboration can effectively increase productivity due to both the ease of communication and access to resources

Use of networks by user

Response times – important for users and will usually give up if too slow. Factors such as server load are out of the hands of the developers and is an effect of networking

Interface design – Interface design can improve or ease response times. If interfaces are designed in such a way that the program provides a visual response even if it has not received a networking response, a user is more likely to continue use of the program

Privacy and security issues – Often sensitive data is transferred across networks. Precautions are often made to ensure that the data remains private.

The Software Market

The aim of a developer is to maintain their space in the software market.

To market a software product the following needs to be considered:

Product

- The expectation of customers should be met by product
- It is unethical to produce a product that does not meet the user's expectations.
- Also includes custom software, where software is designed specifically for a business

Place

- An audience must be specified before deciding on a place to sell the product

- There are many methods of distributing software; shop fronts, online, bookstores.
- Depending on where the software is sold, it can change the perception of customer’s expectations

Price

- A developer has a responsibility to arrange a reasonable price for a product
- Methods for pricing a product are either a cost-plus basis, or consumer-based pricing

Promotion

- The different ways developers use to persuade people to buy products
- Developers have an ethical responsibility in what they say about their product to their consumers
- Promotion should help customers make informed choices
- Information cannot be misleading
- Word of mouth is a very powerful promotional tool for software.

Impact of new developers and software products

New software breakthroughs are usually inventions of new thinking and therefore boom. New products will pop up to try and innovate against the existing product, but unless it provides a huge advantage, the consumer is more likely to stick with the original default software product.

Legal Implications

National Legal Action

RACV vs Unisys

Claims management system electronic storage and retrieval system

Failed expectations of software. RACV sought damages citing false representation and misleading conduct. Awarded damages of $4 million dollars to RACV

Microsoft vs Apple

Microsoft released windows after Apple released Macintosh. Both systems were distributed before legal action was taken. A settlement was reached allowing both companies to distribute their OS.

Software Development Approaches

Factors of defining approaches

- Scale of the product
- Complexity of the product
- Skills of the personnel developing the product
- Detail of the requirements
- Time for project
- Budget

5 software approaches:

- Structured
- Agile
- Rapid application development
- Prototyping
- End user

Each software development approach consists of the same stages continually repeated, but with the frequency and timing unique to each development cycle.

5 stages include:

- Defining and understanding the problem
- Planning and designing
- Implementation
- Testing and evaluating
- Maintenance

Structured Approach

- Time consuming, ordered and controlled
- Highest level of personnel skill required
- Product of highest size and quality
- Top down approach (waterfall method)
- Thoroughly planned, documented and monitored
- Used for large products in large companies
- Costs are high and errors can be hard to find and correct
- Requirements are understood before design and coding occurs
- Requires high project management

Agile Approach

- Emerged due to demand for products to be specifically tailored for individuals
- Places emphasis on system being developed
- In depth documentation is not needed
- Focus on team-work, cooperating, communication skills and efficient work methods
- Development team must be able to adapt to situation
- Small teams preferred; members are multiskilled
- Characterized by speed of getting to user, interaction between team and users, responds well to changing circumstances
- Product in development being continually updated

Prototyping approach

- Lies between structured and agile approach
- Predates the agile approach
- To Repeatedly iterates through development stages to refine user requirements
- Each prototype includes more functionality
- Intense user interaction
- Prototypes are just interactive models of the user interface
- Favoured over structured approach if ability to adapt to changing requirements is required

Two types of prototyping

- **Concept prototyping**
  - Inspire discussion and thought about product
  - Prototype will never be final product
  - Used to develop requirements and discarded after evaluation
- **Evolutionary Prototyping**
  - Prototype ends up being final product
  - Each successive generation of prototype is an improvement

Some software developers take this further, by considering the maintenance stage of the cycle as part of the designing and understanding stage of the next cycle of development

Acknowledges that the communication with users is very important and helps in the defining and understanding of any future products to be developed.

Rapid Application Development

- Main purpose of reducing time and money
- Result of low quality, less usability and less features.
- RAD has only become possible due to 4th generation languages which enable visual production of interfaces with little coding taking place in the background
- Ability to include re-usable code which has been developed
- Integrate with other software solutions
- Development team to work very close to each other
- Continuous feedback with users
- Only meets most important requirements
- RAD lacks formal stages to reduce time and cost
- Removes need for data structures, algorithms
- Uses existing routines and API of applications
- RAD is suited when
  - Distribution is narrow
  - Application runs on small LAN
  - Does not need to interface with other software systems
- Advantages of RAD
  - Reduced coding of modules where library code is used
  - Less errors
  - Library code can ensure consistent look and feel
  - Shorter development cycle

End User

- never used in software companies, not made for purpose of making money
- used to address a specific problem that end user needs a solution for
- very small budget

- does not require knowledge of workings of a computer, as it uses Customised off the shelf (COTS), Wizards or other automated code generating devices.
- Resulting programs will not be overly complex and may not meet specifications that the user may have initially wanted
- Advantages of End User approach
  - The solution can be revised and modified at will, without consultation with other users
  - No money required
  - Less development time required
  - A freedom to change requirements at any stage

Combinations of Software Development Approaches

- Does not have to be of one type or another
- Combinations of approaches are perfectly viable, as different components of a program may be developed using different approaches
- Combination can be used to make the most efficient, low costing, high quality solution

Use of CASE tools

- Computer aided software engineering
- Use of computer assisted method to organise and control development of software, especially on large complex projects
- Using CASE allows designers, code writers, testers, planners and managers to share a common view of where a project stands at each stage of development
- CASE tool may portray progress graphically and may be linked to the documentation
- Supports specific tasks in software development lifecycle:
  - Business and Analysis modelling
  - Development – design and construction
  - Verification and validation – analyse code and specifications for accuracy
  - Configuration management – control check in and out of repository objects
  - Metrics and measurement – analyse code for complexity, modularity, performance
  - Project management - manage project plans, task assignments, scheduling
- Another common way to distinguish CASE tools is the distinction between Upper CASE and Lower CASE
  - Upper – support business and analysis modelling, they support traditional diagrammatic languages such as ER diagrams, Data Flow diagram, structure charts, decision trees, decision tables
  - Lower – support development activities such as physical design, debugging, construction, testing, component integration, maintenance and reverse engineering
- CASE tools are used in following stages
  - Software version control – as software is improved it becomes more difficult to keep track of versions (1.0, 2.0)
  - Test data generation
    - Test correct operation of modules
    - Test and evaluate performance
    - Check error messages
    - Outputs can be compared
  - Production of documentation
    - Avoid delays in production
    - Data flow diagrams

        - Structure diagrams
        - Flow charts
    - Production of code

Methods of installation of new or updated systems

Typically there involves a conversion from an old system to the new system. There are four typically methods of installation or conversion:

1. Parallel conversion
2. Pilot conversion
3. Phased conversion
4. Direct cut over

Direct Cut-Over

- Involves the old system being completely dropped and new system being installed at the same time
- Need to make sure the new system is completely functional and operational
- This conversion method is only used when it is not feasible to operate two separate systems at the same time
- Any data conversion and import must be done before installation occurs
- New data lost if new system fails
- Implementation occurs over shorter period but may be more time consuming

Parallel

- Involve the operating of both systems being run for a period of tome
- Allows problems with new system to be found without loss of data
- Once new system is up and running, the old system can stop being used
- Old systems must be used until conversion

Phased

- Gradual introduction of the new and discarding of the old
- Done by introducing new parts while removing old parts
- Often used when product is still under development
- Over time

Pilot

- New system installed for small number of users
- User learn use and evaluate new system
- Once it seen as satisfactory it is installed and used by all
- Allows users to become experts and teacher of the new system
- Pilot conversion also allows for testing of product in an operational setting

Employment trends in software development

- Large increase 1995 □ 2000, employment continues to grow, albeit slower pace
- Employment based on experience
- Meet demand of jobs
- Outsourcing
    - Outsource work to specialists if they do not have expertise or resources

    - To reduce and control costs
    - Higher quality results from specialists
    - Access to new tech
    - Faster development times
    - Specialists respond to change
- Contract programmers
    - Short term to write specific software products
    - Common for analysts and programmers to change employers as they seek new contracts

Trends in software development

- Mobile application
- Big data processing – data analysts
- Cloud technology – iCloud, drop box; cloud security
- DevOPs software – development operation managers – serve customer and clients better; cloud management
- User interface developers- consumer focused, intuitive, create enterprise applications and apps make it user friendly
- Web based software – google apps, Facebook, YouTube Instagram
- Learning objects
- Widgets
- Cloud computing
- Mobile tech

Defining and Understanding the Problem

Identifying the problem

- Carried out by a system analyst
- Includes following aspects
    - Needs of the client
        - Functionality
        - Compatibility
        - Performance
    - Determining the objectives which will be met
    - Boundaries and scope of the problem is understood
- Understanding the problem will enable a better understanding of the inputs, processing and outputs of the system to be developed

Needs of the client

A need is an instance in which a necessity or want exists. In software development, a solution to meet those needs is usually sought. Through discussion and consultation of the needs, a set of requirements are developed to inform the development process. Tools to analyse the requirements such as:

- Surveys – info for large groups; limited in detail
- Interviews – freedom to discuss in detail; limited to smaller set of people and time consuming
- Time management studies and observations – involves observing people processing tasks
- Business analysis – examines business rules and tasks

Objectives

- Short term and long-term aim and plans
- Clarification of objectives leads to determining the requirements, boundaries and rules for the development.
- Measurable outcomes are also expressed in objectives

Requirements

- Statements that are indicators of what needs to be met
- The final evaluation of a project's success or failure is based on how well the original requirements have been achieved

Functionality Requirements

- What the system will do and what it needs to achieve
- Give direction to the project
- Requirements are defined as features, properties and behaviours a system must have to achieve its purpose
- Functionality requirements may be expressed in terms of measurable standards
- Non-functionality requirements are requirements which impose constraints on design or implementation

Compatibility Issues

- Software of various types runs on a variety of environments including: OS, browser, hardware and screen resolution.
- Environments can be varied by the user
- Software developers must ensure products are able to be used on multiple systems or devices and conditions
- Compatibility issues include:
    - Problems with different OS versions
    - Incompatible hardware such as graphics controllers
    - Different screen size and resolutions
    - Browsers and different HTML standards
    - Hardware and drivers not supporting software
    - Incompatible LAN hardware and drivers

Performance Issues

- Testing and real-world application performance may be very different
- Testing must be extremely thorough and broad but within boundaries
- Issues include:
    - Appearing to not be responding when time takes too long
    - Poor response times in networking operations
    - Screen refresh time lag
    - Memory management issues
    - CPU processing and process requests

Boundaries of the problem

- Define the limits of the problem
- Anything outside the system is said to be part of the environment
- System connects with the environment through an interface

Issues relevant to a proposed solution

Determining if an existing solution can be used

- Social and ethical considerations
- Consideration of existing software products
- Customisation of those existing software solutions
- Cost effectiveness
- Licensing considerations

Social and ethical

- Changing nature of work for users

    - New skills to be acquired
- Effects on levels of employment
    - Technology replacing work done by humans
    - Created new information technology jobs
    - Less costs for businesses
- Effects on the public
    - Large software systems can have substantial effect on general public e.g. ATM's
    - Older generation unwilling to accept technological innovation

Legal issue including license considerations

- Issues relating to copyright
- Software that is used to store and access sensitive information will need to include safeguards against unauthorised access

Customisation of existing software products

- Cost effective strategy for obtaining new functionality
- Many software developers spend much of their time modifying their own existing products to suit specific needs of individual clients
- Open source software is often customised to add new features; in many cases the modifications are built as add-ons which are then available to other users of the base software product
- The ability to customise software helps the original developer of the product widen their market
- It is common for tools to be included in many commercial products which allows the user to create customised screens and other functionality using wizards and multiple drag and drop design screens.

Cost effectiveness

- One of the constraints of a new software system will be that it falls within a certain budget. If it is not a requirement, then it will most likely be a constraint
- Compare the costs between developing a new product or modifying an existing product
- Development hardware costs
    - Any new hardware that needs to be bought/leased to enable development
- Development software costs
    - What software is required; programming languages, CASE tools, database managements systems, graphics tools
- Development personnel costs
    - Salaries of development team and other staff
- Outsourcing costs

A budget for the development project needs to be decided about a range of factors including:

- Available capital
- Predicted sales of finished product
- Cost savings as a result of the products implementation
- Predicted future upgrade and maintenance costs for the product
- Ongoing consumer support costs

Selecting appropriate development approach

- Selecting an appropriate development approach if there is no appropriate existing solution is completed by understanding the requirements and specifications of the problem
- The nature and type of problem will also affect the development approach and programming paradigm to be used.

Design Specifications

Specifications of proposed system

- A software requirements specification is a standard framework for a team to develop a large, complex software system and includes a complete description of the behaviour of a system to be developed.
- Includes a set of use cases that describes all the interactions the users will have with the software.
- SRS also contains non-functional requirements, and the methods which will be used to model the system will also be specified.

Developers perspective

- Data types
- Data structures and variables
    - Variables represent storage locations of data within the computer system
    - A programmer will look at the data items and determine how they are to be stored and accessed
    - The data type and an appropriate identifier is then determined
    - Meaningful variable names are considered intrinsic documentation and help programmers follow the logic of the program
    - Global or Local Variables
- Algorithms
    - Pseudocode or flow charts
    - Problem is modularised
    - Top down decomposition
    - Familiar modules which can be drawn from a library of code
    - This method ensures a minimum of testing is required
- Quality assurance
- Documentation
- Design approach

Users Perspective

- Interface design
- Social and ethical issues
- Relevance to user's environment and computer configuration

Specifications developed from the user's point of view should include and design specifications that influence the experience of the end user.

Standards for interface design will be specified to ensure continuity of design across the project's screens. The wording of messages, design of icons and the format of any data presented to the user need to be determined and a set of specifications created.

- **Ergonomic Issues** should also be considered
- User's existing computer environment will influence specifications created.
    - If users are familiar with existing applications, then some design elements should be incorporated in new solution so transfer of skills can take place
- Operating system settings and consideration of hardware necessary to execute new development
    - Models will assist in determining user-based specifications
    - Communication and feedback to and from users is especially important during the early stage

System Documentation

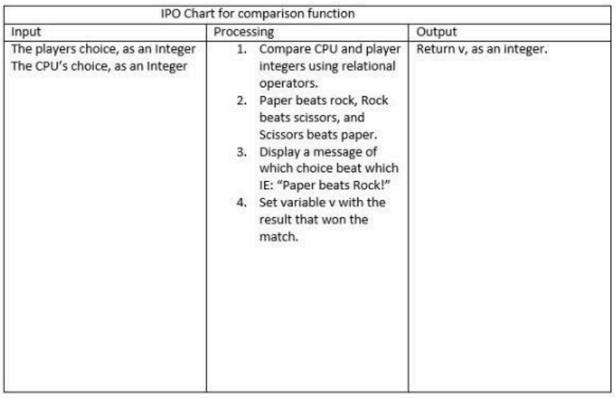
Representing a system using systems modelling tools

- Different types of documentation are produced throughout the software development cycle
- Many large companies utilise diagrams that form part of the UML (unified modelling language) which incorporated a variety of different modelling tools that are now available as part of many CASE tools and IDE's (integrated development environments)
- A model of a system is a representation of that system designed to show the structure and functionality of a system. Many system modelling tools are in the form of diagrams

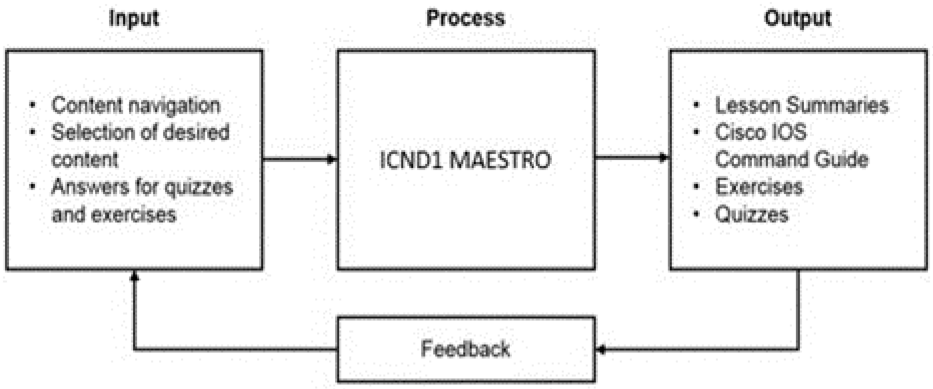
- The model gives directions and specifications for the developers
- Different types of modelling are applicable to different aspects of the system

IPO Diagrams

- Explain how inputs are transformed into outputs by processing
- Expand on the processes found in the data flow diagram and structure diagram
- IPO diagram can either be diagrammatical or table form

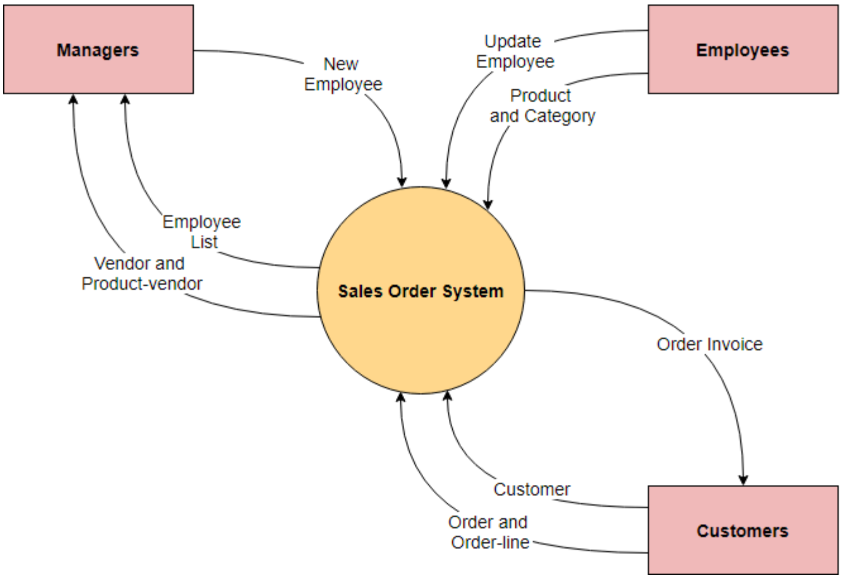


IPO Chart for comparison function		
Input	Processing	Output
The players choice, as an Integer The CPU's choice, as an Integer	1. Compare CPU and player integers using relational operators. 2. Paper beats rock, Rock beats scissors, and Scissors beats paper. 3. Display a message of which choice beat which IE: "Paper beats Rock!" 4. Set variable v with the result that won the match.	Return v, as an integer.



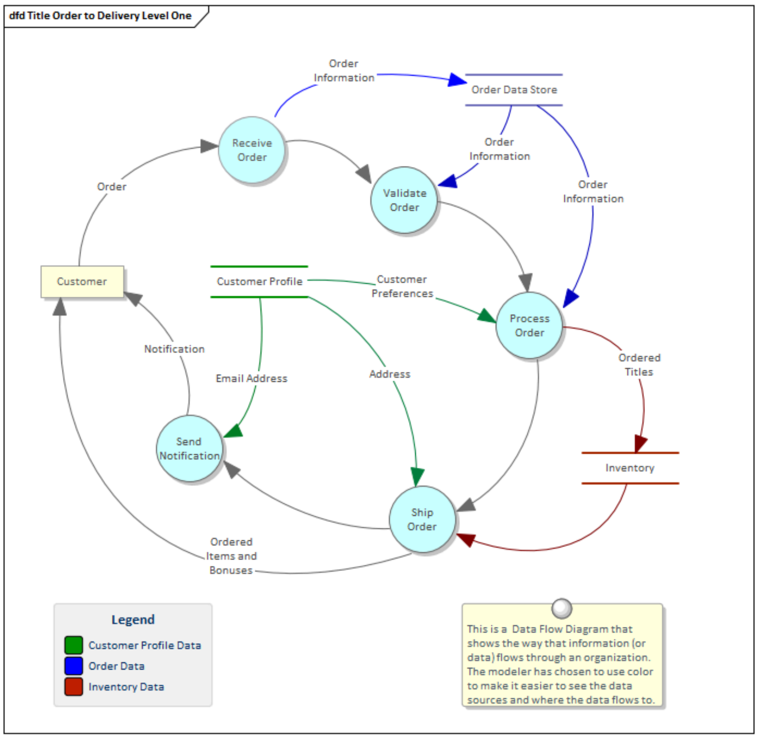
Context Diagrams

- Used to present an overview of the whole system
- Shown as a single process along with the inputs and outputs
- Attempt to show the data entering and information exiting the system without detailing the processing required in any detail.
- The squares are referred to as external entities.
- Connected to single process by data flow arrows
- Elements are labelled
- Does not show data stores nor internal processes
- Context diagrams are referred to as level 0 data flow diagrams
- Helpful with understanding how system interfaces with the environment
- Process: a circle
- Arrow – flow of data
- External entity/rectangle – person or organisation, source or sink that provides or receives data



Data Flow Diagrams

- Represent a system as several process that together form a single system
- Refinement of a context diagram
- Show a further level of detail not shown in context diagrams
- DFD's identify source of data, flow between processes and it's destination along with data generated by the system
- 4 symbols:
  - Squares – representing external entities, sources or destinations of data
  - Circles – processes, which take data as input, process it, and output it
  - Arrows- represent the data flow, electronic data or physical items
  - Open ended rectangles – represent data stores such as databases

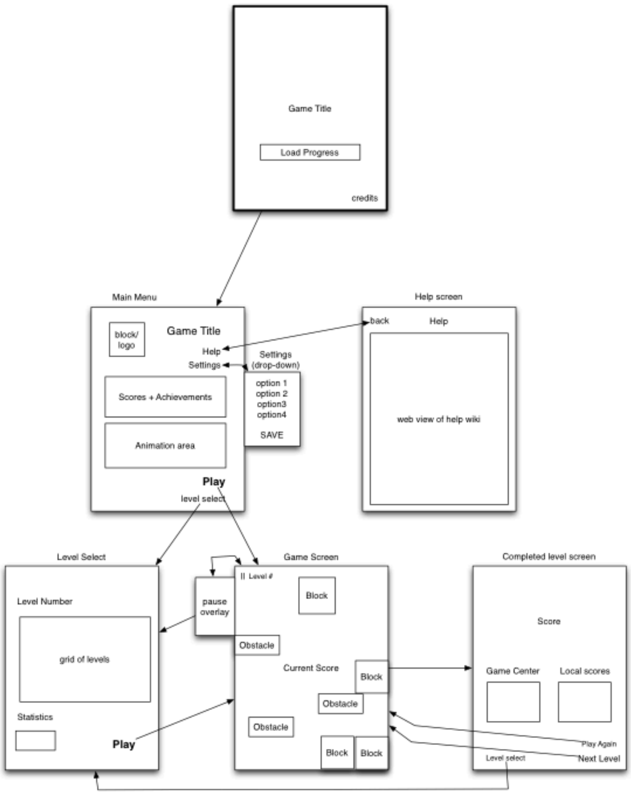


This is a Data Flow Diagram that shows the way that information (or data) flows through an organization. The modeler has chosen to use color to make it easier to see the data sources and where the data flows to.

Storyboards

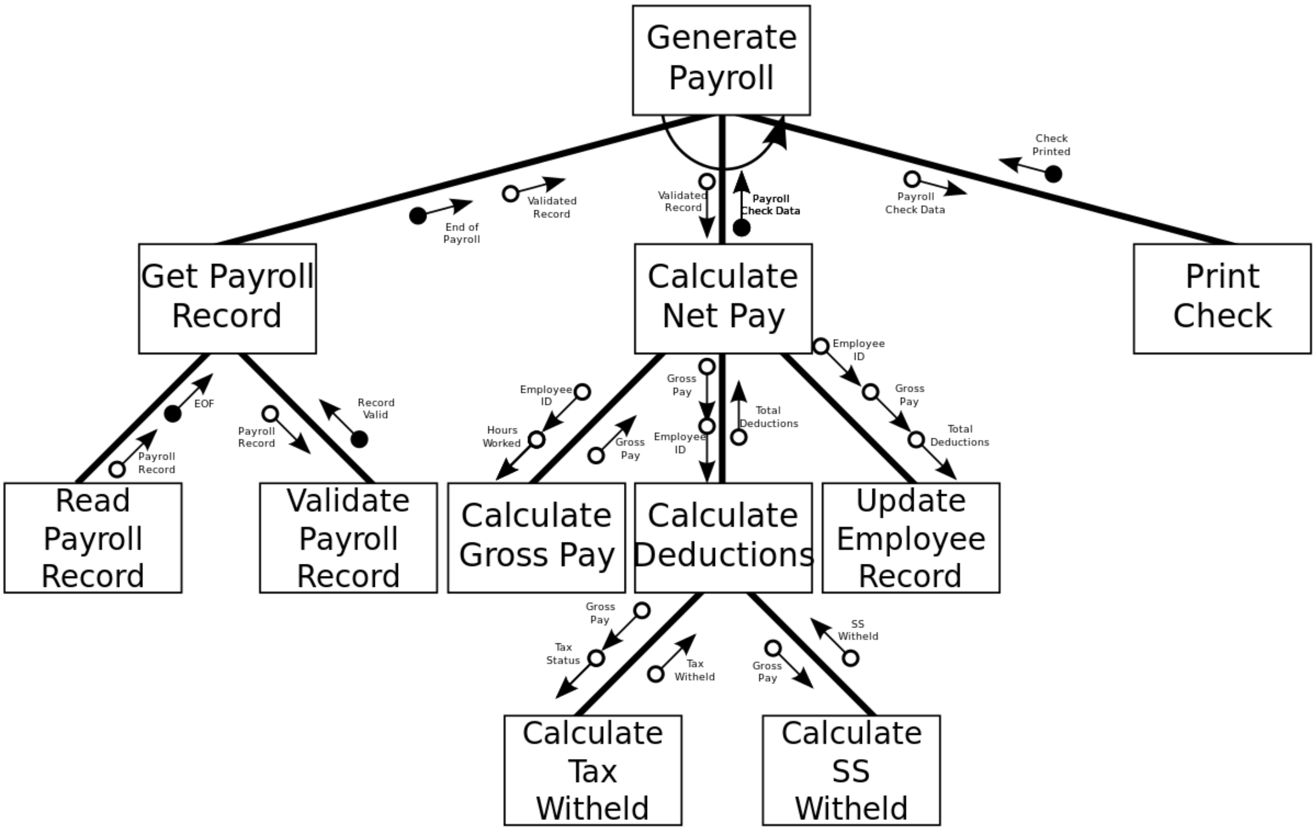
- Shows the various interfaces in a system as well as the links between them

- The representation of each interface should be detailed enough for the reader to identify the purpose, contents and design elements.
- Areas used for input output and navigation should clearly be identified and labelled
- Any links shown between interfaces should originate from the navigational element that triggers the link.



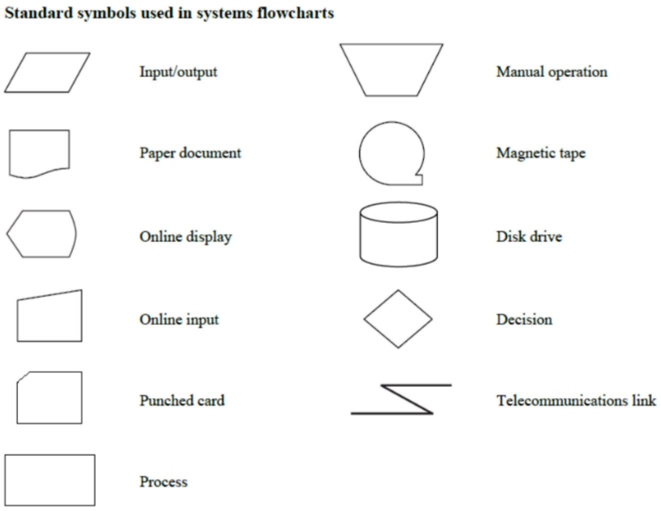
Structure Charts

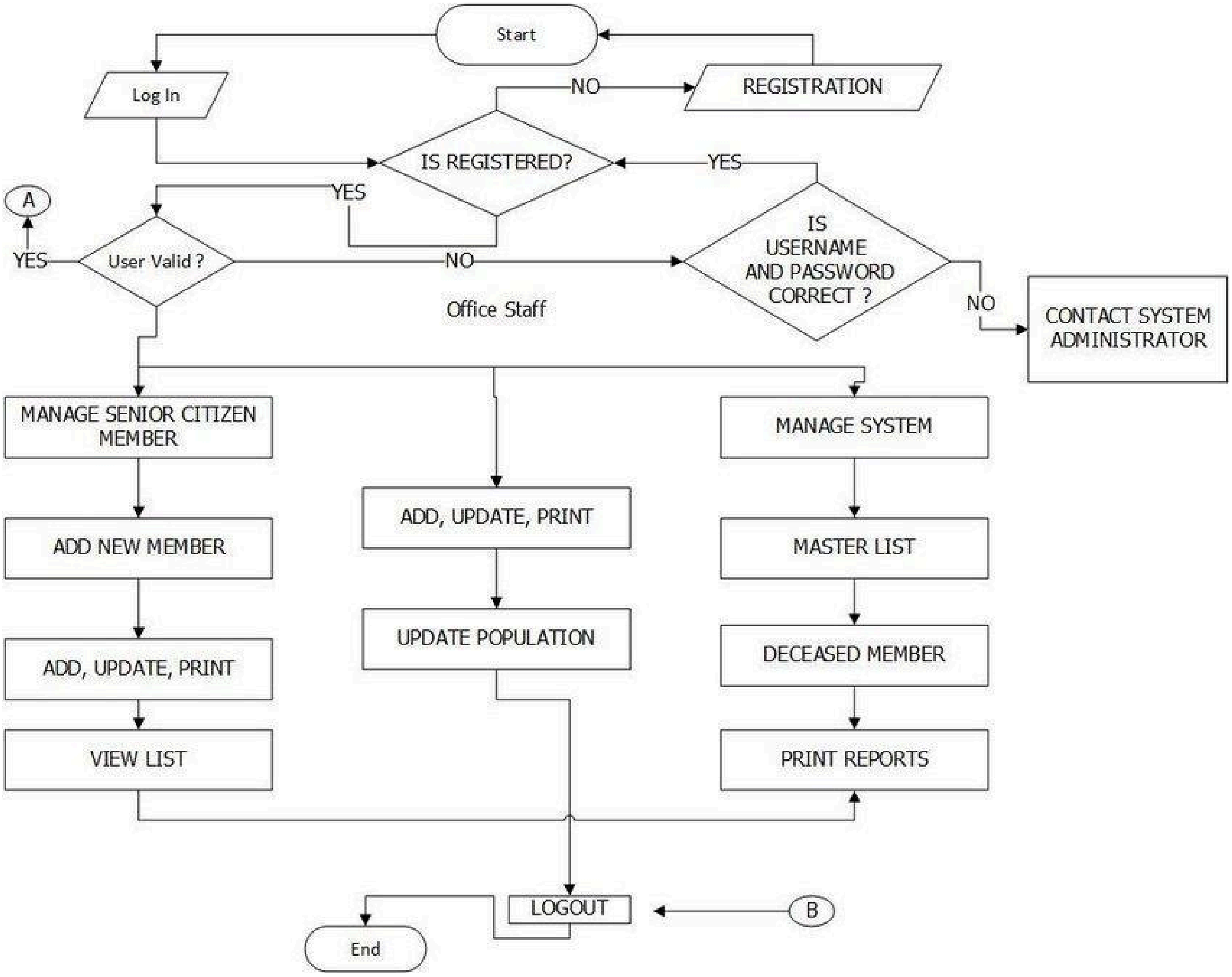
- Describe the top down design and sequence of processing
- Represent a system by showing the separate modules or subroutines that compromise the system and their relationship to each other
- Chart is read from top to bottom, with component modules or subroutines on successively lower levels, indicating these models or subroutines are called by the module or subroutine above.
- Modules are read left to right to show the order of execution
- These modules set the structure for the development of the IPO diagrams and then the development of the algorithm and subsequent coding of each module
- Structure charts are useful for maintenance
- Symbols used
    - Open arrow – data movement between modules or subroutines, usually passes as parameters
    - Closed arrow – indicate a flag or control variable
    - Small diamond at intersection – a decision
    - Undo arrow – repetition
    - Line – call the module



System Flowcharts

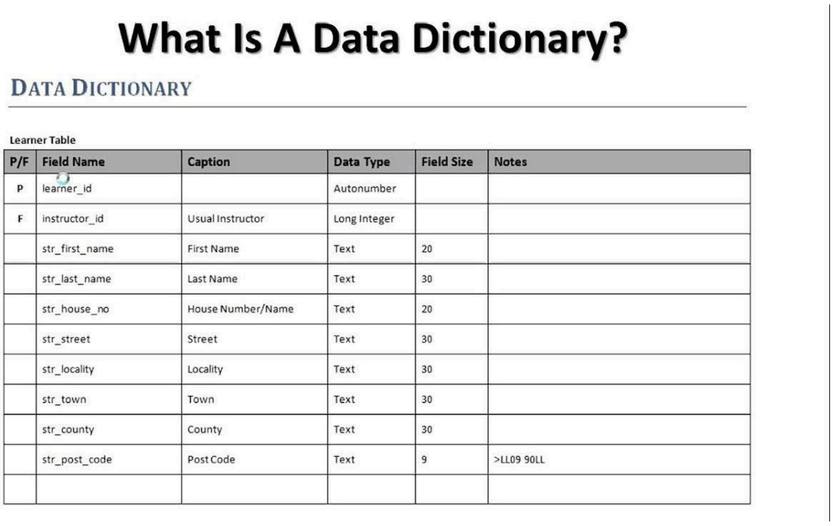
- Used to represent the logic and movement of data between the system components, including hardware, software and manual components
- System flowcharts are a diagrammatic way of representing the system to show the flow of data, the separate modules comprising the system and the media used
- Many symbols for system flowcharts have become outdated because of changes in technology
- Symbols:





Data dictionaries

- Comprehensive description of each data item in a system
- Commonly includes:
    - Variable name
    - Size in bytes
    - Number of characters as displayed on screen
    - Data type
    - Format including decimal points if applicable
    - Description
    - Example



What Is A Data Dictionary?

DATA DICTIONARY

Learner Table

P/F	Field Name	Caption	Data Type	Field Size	Notes
P	learner_id		Autonumber		
F	instructor_id	Usual Instructor	Long Integer		
	str_first_name	First Name	Text	20	
	str_last_name	Last Name	Text	30	
	str_house_no	House Number/Name	Text	20	
	str_street	Street	Text	30	
	str_locality	Locality	Text	30	
	str_town	Town	Text	30	
	str_county	County	Text	30	
	str_post_code	Post Code	Text	9	>LL09 90LL

Algorithms used to document the logic in modules and subroutines

- The logic in modules and subroutines are represented before coding using pseudocode and Flowcharting
- Pseudocode uses English-like statements with defined rules of structure and keywords
- Flowchart is a graphical representation
- Flowcharts should not be used for large projects

Test data and expected output

- Many operational factors may be outside the control of the programmer, or unable to be foreseen (e.g. inputs, variation in hardware, different OS and changing technologies)
- Programmer can only test the program within the bounds that are set during the analysis stage of development
- With an increasing size of an application it makes it harder to create test data to run throughout the whole application
- Modularisation of programs means that data can be created to test each module however it is also likely that an unforeseen output from one module will cause problems in other modules in the program

Communication issues between client and developer

Need to consult with the client

- Implementation of a new system is generally easier if stakeholders feel that they have contributed to the development
- Many conflicts can be avoided if good communication is carried out between the developers and the users
- Developers are familiar with the technical aspects of the development of a new system
- The users are familiar with the operation of the current system and can provide good feedback and screen design operation and training requirements
- Good communication is achieved by:
    - empowering the user
    - acknowledging the user’s knowledge and perspective
    - enabling and accepting feedback

The need to incorporate the client’s perspective

- the developer and user are equally important to the success of the software project, and the developer must accept the expertise of the user and use it to better understand the system
- user perspective should be considered while designing both the processes and the interfaces

The need for the developer to enable and consider feedback

- for effective communication, developers need to establish both formal and informal channels and communication
- formal channels
    - memos and regular meeting to keep the users up to date with the development process
- informal channels
    - discussed at any time during development process, as quite often matters will arise that cannot be kept until the next formal meeting
- user should have the opportunity to provide feedback
- development process will proceed well if members can achieve constructive criticism and if there is two-way communication

The need to involve and empower the client during the development process

- changing work practices are often cause for discomfort, resentment and fear

- people who are going to be affected by a software change will more readily accept them and if they feel that they have had input into the design process and will have a sense of ownership of the product
- ownership is especially important after installation, as users are more likely to describe problems or suggest enhancements
- empowering the user also means giving them the ability to make decisions that affect their work

Quality Assurance

Refers to the planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled.

- Quality assurance involves:
  - the need to define the criteria on which the quality of the product will be judged and may include requirements
  - putting in place management processes to ensure that quality criteria is being met
  - an ongoing QA process to ensure quality criteria is being met
- A set or list of measurable criteria is required
- The criteria to judge the quality of the project is considered and set early
- SQA ensures that software will function reliably as intended and is free of errors
- Includes:
  - Efficiency – best use of computer's resources
  - Integrity – correctness of data within the system
  - Reliability – ability of the system to continue to perform its functions over time
  - Usability – ability of software to be learned and used by new users
  - Accuracy – software performs its functions correctly and according to specifications
    - Can only be assured when code has been thoroughly tested
  - Maintainability – measure of ease with change to software
  - Testability – individual modules and subroutines should be able to and have been thoroughly tested and system should be tested to ensure it performs according to requirements
  - Re-usability – ability to reuse code in other related systems
    - Programmers maintain a library of subroutines and modules that perform commonly used functions to ensure quicker and error free coding

Planning and Designing Software Solutions

Identification of appropriate modules

- Algorithms are usually written by designing modules and then the sub-modules
- This design process is called a 'top-down design' and is the most popular way to solve problems with structured programs
- Top down design is where the main module is written followed by the sub-modules/functions it calls

Inputs, Outputs and Processes

- IPO charts look at the processing of data
- Also address the management of data; where it comes from, how we store it, how it is used in processing and its format or type for output

Consideration of global and local variables

- Consideration of global and local variables should be done in the design stage
- Local variables are only used in the module which they are declared
- Any modules can access global variables

Scope of variables

- Where the variables are declared/created/used
- Scope describes where in the algorithm the variable can be used

Parameters

- Components of functions
- Identify values that are passed into a function
- Passed from mainline to sub-program

Standard Algorithms

- Linear search
- Finding maximum or minimum values
- Binary search
- Insertion sort
- Selection sort
- Bubble sort
- Processing strings (extracting, deleting and inserting)
- Generating set of unique random numbers
- Processing of sequential and relative files

Arrays

- Structured data type
- Collection of like data
- Data is held in a cell with an index
- Cells are linear in structure
- Array has a name or label
- Arrays hold a single data type
- You can nominate the size of an array
- You do not need to fill all cells in an array

Declaring an Array

- Array is declared before values can be added
- E.g. Declare Array (1-5) as type string

Accessing an Array

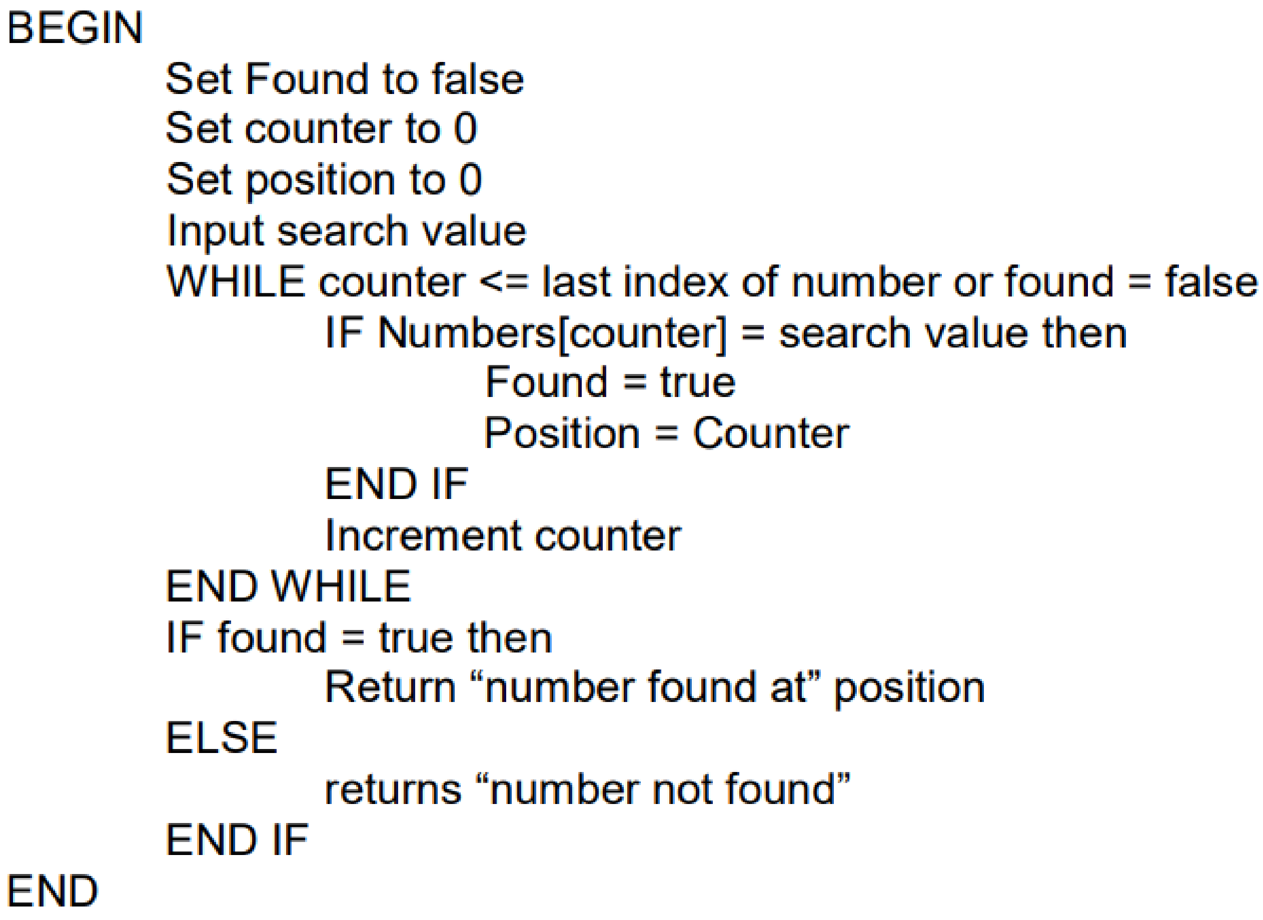
- To find (access) the third data element in the 'array' = array (3)
- Will return third index value
- To assign a simple variable a value from an array: myarray = array (2)

Sentinel Value

- Used to mark the end of a data list
- EOF – end of file
- Indicates last entry of fata in the structure, so processing can cease

Linear Search

- Examining one item at a time in an array beginning at the first item and moving to the end of the array
- Does not require numbers to be sorted
- Processing taken for the search is directly related to the size of the array searched
- Linear search involves:
    - The function accepting a search value
    - Traverses the array using iteration
    - Checks each value compared with search value
    - Returns 'number found' and the index of cell which value equals search value
    - Otherwise returns 'number not found'



```
BEGIN
        Set Found to false
        Set counter to 0
        Set position to 0
        Input search value
        WHILE counter <= last index of number or found = false
                IF Numbers[counter] = search value then
                        Found = true
                        Position = Counter
                END IF
                Increment counter
        END WHILE
        IF found = true then
                Return “number found at” position
        ELSE
                returns “number not found”
        END IF
END
```

Find Maximum or Minimum

- The value in the first element is stored in a temporary variable called Max/Min
- As the array is traversed each element is considered to determine if its value is larger/smaller than that stored value
- If so, the value in Max/Min is replaced by this larger/smaller value, and the index of this element is stored in a temporary variable called MAX/Minindex
- When all elements have been considered, Max will contain largest value and MaxIndex will contain the index of the largest element and vice versa for minimum

```
BEGIN FindMAX
        Max = Numbers (0)
        MaxIndex = 0
        i = 1
        REPEAT
                IF Numbers (i) > Max THEN
                        Max = Element (i)
                        MaxIndex = i
                END IF
                i = i + 1
        UNTIL i > NumElementsInArray
```

```
    Display “The highest value is ” Max “ at position ” MaxIndex
    END FindMAX

BEGIN FindMIN
    Min = Element (0)
    MinIndex = 0
     i = 1
    REPEAT
        IF Element (i) < Min THEN
            Min = Element (i)
            MinIndex = i
        END IF
        i = i + 1
    UNTIL i > NumElementsinArray
    Display “The smallest value is ” Min “ at position ” MinIndex
    END FindMIN
```

Binary Search

- Only used if the values in the array are already sorted in order
- The binary search starts by checking the middle value of an array to see if it matches the search value
- If not, the binary search then determines if the search value is in the first half of the array or the second half (after the middle value)
- This process is repeated with the remaining list of items
- Eventually the required item will be found or the list of possible items will be empty

```
BEGIN binarySearch with numbers and searchValue
    set lowBoundary to first index of numbers
    set highBoundary to last index of number list
    set middleindex to 0
    set foundit to false
    Input search value

    WHILE highBoundary >= lowBoundary or foundit = false
        Set middleIndex to (highBoundary + lowBoundary) / 2
        IF numbers[middleIndex] = searchValue THEN
            Foundit = true
            Return middleIndex
        ENDIF
        IF numbers[middleIndex] > searchValue THEN
            Set highBoundary to middleIndex – 1
        ELSE
            Set lowBoundary to middleIndex to +1
        ENDIF
    ENDWHILE

    Return “Number not found”

END
```

Insertion Sort

- Used when a large part of the data is already sorted
- During each pass the last element from the unsorted part is inserted into the appropriate place in the sorted part of the array

- A linear search is conducted to find the correct place to insert
- As each sorted element is checked it is moved to the left/right to make room for the new elements
- At each pass the sorted section of the array increases by 1
- Process continues until the unsorted section = 0

```
BEGIN
        Set First to 1
        Set Last to 6
        Set PositionOfNext to Last – 1
        WHILE PositionOfNext >= First
                Next = The Array(PositionOfNext)
                Current = PositonOfNext
                WHILE Current < Last AND Next > TheArray(Current + 1)
                        Increment Current              //shuffles sorted part along
                        TheArray(Current – 1) = TheArray(Current)
                ENDWHILE
        TheArray(Current) = Next
        Decrement PositionOfNext
        ENDWHILE
END
```

Selection Sort

- During each pass a linear search is performed
- A marker is placed at the first cell in the array and then search through the array from that position onwards looking for the smallest value
- When the smallest value is found, it is swapped with the marker's cell value. This naturally places the smallest value at the front of the array
- The next step is to increment the marker to the next cell and repeat the process. When the marker reaches the last cell, the array is sorted.

```
- procedure selection sort
-    list  : array of items
-    n     : size of list
-
-    for i = 1 to n - 1
-    /* set current element as minimum*/
-       min = i
-
-       /* check the element to be minimum */
-
-       for j = i+1 to n
-          if list[j] < list[min] then
-             min = j;
-          end if
-       end for
-
-       /* swap the minimum element with the current element*/
-       if indexMin != i  then
-          swap list[min] and list[i]
-       end if
-    end for
-
- end procedure
```

swap procedure would include temp value

Bubble Sort

- popular amongst novice programmers
- main logical structure is based on traversing an array and switching adjacent pairs of values that are not in the correct order
- after one travers the largest value will have ‘bubbled’ to the end of the array. This is repeated until all values are in the correct cells

```
BEGIN bubbleSort with numbers
    Set index to first index of numbers +1
    WHILE index <= last index of numbers
        IF numbers[index –1] > numbers[index] THEN
            Swap(numbers[index –1] and numbers[index])
        ENDIF
         Increment index
    ENDWHILE
END
```

Processing Strings

- string – string of characters one after another forming a sequence

String Concepts

- in traditional systems strings were not identified directly in a programming language
- a programmer had to work with the individual characters that make up a message or word
- a string was represented as an array of character
- commonly given a default value of 256 characters
- common operations include
    - determining the length of the string
    - joining strings together – concatenation
    - extracting characters from the string
    - inserting characters into the string
    - deleting characters

Determining length of string

```
BEGIN StringLength(String)
    Index = 0
    WHILE String[Index] contains a character
        Increment Index
    ENDWHILE
    Length = Index
     RETURN Length
END StringLength
```

Extracting

```
BEGIN ExtractString(StartPosition, NoOfCharactersToExtract)
    Index = StartPosition
    Counter = 0
    Set NewString() to an empty array of characters
    WHILE Counter <= NoOfCharactersToExtract
```

```
        NewString[Counter] = OriginalString[Index]
        Increment Index
        Increment Counter
    ENDWHILE
    RETURN NewString
END ExtractString
```

Inserting

```
BEGIN InsertString(NewString, PositionToInsert)
    'Calculate Length of original and new strings'
    OriginalStringLength = StringLength(OriginalString)
    NewStringLength = StringLength(NewString)
    TempStringLength = 0
    Index = 0
    ExtractOriginal(PositionToInsert, OriginalStringLength)
    InsertNewString(PositionToInsert, NewStringLength)
    ReplaceOriginal(Index, TempStringLength)
END InsertString

BEGIN ExtractOriginal(PositionToInsert, OriginalStringLength)
     'Create a copy of the end of the original string'
    Index = PositionToInsert
    Counter = 0
    Set TempString() to an empty array of characters
    WHILE Index <= OriginalStringLength
        TempString[Counter] = OriginalString[Index]
        Increment Index
        Increment Counter
     ENDWHILE TempStringLength = Index
    RETURN TempStringLength
END ExtractOriginal

BEGIN InsertNewString(PositionToInsert, NewStringLength)
    'Insert new string'
    Index = PositionToInsert
     Counter = 0
    WHILE Counter <= NewStringLength
        OriginalString[Index] = NewString[Counter]
        Increment Index
        Increment Counter
    ENDWHILE
    RETURN Index
END InsertNewString

BEGIN ReplaceOriginal(Index, TempStringLength)
    'Attach end of original string back onto the string'
    Counter = 0
    WHILE Counter <= TempStringLength
        OriginalString[Index] = TempString[Counter]
        Increment Index
        Increment Counter
    ENDWHILE
END ReplaceOriginal
```

Deleting

```
BEGIN DeleteString(StartPosition, NoOfCharactersToDelete)
    Index = StartPosition
    OriginalStringLength = StringLength(OriginalString)
    WHILE Index <= OriginalStringLength - noOfCharactersToDelete
        'Overwrites elements to be deleted
        OriginalString[Index] = OriginalString[Index + NoOfCharactersToDelete]
        Increment Index
    ENDWHILE
    WHILE Index <= OriginalStringLength
        OriginalString[Index] = ""
        'Clear contents of these elements
        Increment Index
    ENDWHILE
    'The original str string will now be NoOfCharacters Shorter
END DeleteString
```

Generating set of unique random numbers

- most high-level languages have a function to generate a random integer or floating-point number
- generally, a range in the random function is specified
- random number generators have applications in gambling, statistical sampling, computer simulation, cryptography
- the generation of pseudo random numbers is an important and common task in computer programming
- different applications require different amounts of unpredictability
- most computer-generated random numbers use Pseudo random number generators (PRNGs) which are algorithms that can automatically create long runs of numbers with random properties but eventually the sequence repeats
- the series of values generated by such algorithms is generally determined by a fixed number called a seed
- one if the most common PRNG is the linear congruential generator which uses the recurrence:

X n + 1 = (a X n + b) mod m {\displaystyle X_{n+1}=(aX_{n}+b)\,{\textrm {mod}}\,m} X_{n+1} = (a X_n + b)\, \textrm{mod}\, m to generate numbers, where a, b and m are large integers, and X n + 1 {\displaystyle X_{n+1}} X_{{n+1}} is the next in X as a series of pseudo-random numbers. The maximum number of numbers the formula can produce is the modulus, m.

- processing a set of unique and random numbers requires
  - an array to store the numbers
  - a loop to generate each number
  - and random generator function
  - adding to final array of numbers
  - a check to see if the number is unique

```
BEGIN randArray(min, max, num)
    FOR I = 0 to max – min
        rNum(i) = I + min
    NEXT i FOR I = 0 to num – 1
    R = random integer from I to (max – min) inclusive
    Arr(i) = rNum(r)
    rNum(r) = rNum(i)
    NEXT i
    RETURN arr
END
```

The VB Rnd function returns a value less than 1, but greater than or equal to zero. The value of Number determines how Rnd generates a random number. For any given initial seed, the same number sequence is generated because each successive call to the Rnd function uses the previously generated number as a seed for the next number in the sequence. Before calling Rnd, use the Randomize statement without an argument to initialize the random-number generator with a seed based on the system timer.

In VB a random number function using Rnd(): randomVa lue = CInt(Math.Floor((upperbound - lowerbound + 1) * Rnd())) + lowerbound

Processing of sequential files

- the data stored in a sequential file is stored in a continuous string
- the data is accessed from beginning to end

- text files are sequential files
- sentinel files are used to indicate the end of a file
- once sequential files are read and the sentinel is reached, files are closed

Priming read

- the statement which reads the first input data record prior to starting a structured loop
- the main purpose of a priming read is to allows the processing block that follows it to be structured

Open for input, output or append

Sequential files can be opened in one of three modes:

1. Input – used to read the data within a file
2. Output – used to write data to a new file
3. Append – used to write data at the end of an existing file

Processing of relative files

- Relative refers to the position of a record in the file as each record is the same length
- Allows the relative position of a record to be known
- The position of each record in the file is used as a key to allows direct access of a record (instead of sequential)
- Relative files are used to store records
- Each record is the same data type and length
- Fields are usually padded out with blank characters

Retrieving, writing and updating a record in a relative file

- Relative files need to be open for relative access to be updated and must be closed before the program ends
- All records are accessed using a key which specifies the relative position of that record within the file.
- The key fields used must contain positive integer values only
- E.g. follow algorithm allows the price of any product to be changed:

```
BEGIN UpdateRecordsInARelativeFile
    Open ProductData for relative access
    Display "Please enter the product number for the next product
    whose price you wish to update:"
    Display "Please enter 999 when you are done"
    Get RequiredProdNumber
    'priming read in case they wish to exit immediately by
    entering 999
    WHILE RequiredProdNumber < > 999
        NotFound = 0
        Read ProductData into ProdNumber, description, quantity,
        price using RequiredProdNumber
        'note the use of the variable RequiredProdNumber as the
        key field, specifying where this record will be found in
        the file
        IF RecordNotFound THEN
            'note the use of the flag RecordNotFound returned by
            the operating system
            Display "Sorry - no such product"
            NotFound = 1
        ELSE
            Display ProdNumber, description, quantity, price
            Display "Is this the correct product?"
            Get Reply
        ENDIF
        'do not update until the correct product record is
        retrieved
        IF NotFound = 0 AND Reply = "Y" THEN
            Get NewPrice
            Write ProductData from ProdNumber, description,
            quantity, NewPrice using ProdNumber
```

Dog

Interface Design in software Solutions

- Designs of screen will be influenced by the nature of the problem.

Consideration of the intended audience

Each screen in a program will have a target audience. If the screen is to be effective, the needs of that audience must be met:

- Organisation of screen elements
- The way they are presented
- The way the user interacts with the interface and the way help is provided

Communication with the user is key to finding out the specifications for interface design depending on the target audience.

Identification of screen size

- Screen size will be affected by the intended hardware the software will be installed on
- Will also impact screen resolutions and graphics hardware and drivers

Identification of data fields and screen elements

- Screens are designed to present data, input data or output data
- It is important for the developer to have a clear understanding of how data items need to be presented to the user
- The context in which data is displayed is closely related to the processes being carried out.
    - Menus are used to initiate executions of modules within a program
    - Command buttons are used to select a different path for execution, often used for confirmations
    - Tool bars; used to quickly access commonly used items
    - Text boxes; receive input in the form of strings
    - List boxes; force the user to select from the given options
    - Combinations boxes; combine functions of text box and list box
    - Check boxes; obtain Boolean input from the user
    - Options or radio buttons; select from one of the displayed options
    - Scroll bars; navigation
    - Grid; likened to an array of records
    - Labels; provide guidance and information to the user
    - Picture/image box; display graphics

Online help and user feedback

- Help system should be designed to encourage user to seek assistance immediately a problem is encountered
- Context sensitive help
    - If user asks for help, new window displayed containing information about user’s last action
    - Small windows can open to display simple tips relating to current screen elements if the user holds the mouse over an element for more than a few seconds
- Procedural help
    - Concise and accurate instructions on how to complete a specific task
    - ‘how’ rather than ‘why’

- Conceptual help
  - Aims to explain ‘what’ and ‘why’ rather than ‘how’
  - Tours, tutorials and wizards

Consistency in Approach

- Must be consistency between screens within an application
- Allows the user to anticipate actions and placement
- Design rules should be created before the development process is undertaken
  - Especially important when a team of programmers is employed
- Transfer across to new product
- Aspects of consistency
  - Names of commands
  - Use of icons
  - Placement of screen elements
  - Feedback
  - Forgiveness

Customised off the shelf packages

- Identifying an appropriate package
  - Cost benefit analysis; if a package will take more time and effort rather than to build a new application from scratch it is not worth modifying
- Identifying the changes that need to be made
- Obtaining permission from original author
- Identifying how the changes are to be made

Standard modules and library routines

Reasons for development and use of standard modules

- Consideration of local and global variables
  - Local variables can only be used within their own module/subprogram
  - Global variables can be accessed throughout the whole program
  - Scope: refers to the amount of the program in which the variable can be used
- Appropriate use of parameters

Identification of appropriate modules or subrdrivers outine

- Thorough documentation including intrinsic naming of variables and objects
- Standard control structures are utilised
- Choice of language suits the module
- Social and ethical issues related to the modules
- Modules that contain redundant tasks will only waste memory

Appropriate testing using drivers

- Drivers are temporary code used to test the execution of a module when the module cannot function individually without a mainline

Recognition of relevant social and ethical issues

- Ease of use
- Accessibility of technical language
- Copyright
- Ergonomics

Factors considered when choosing a language

Sequential or event driven software

- Sequential
    - Screens follow one after the other and minimum user input is required
    - Data items are accessed from outside of the program
- Event-driven
    - Data items are accessed from within the program and the user controls the order of processing, creates an interactive and dynamic pattern to follow

Driven by user or programmer

- User
    - Program logic
    - Requires user's actions to trigger an event
    - Features menus, buttons, icons etc.
    - Order of module execution is defined by the user
    - Polling: process of continuously checking the status of events
    - Event parsing: executes events that the user has instigated
    - Used in computer games
- Programmer
    - Sequential approach
    - Follows set of steps to solve a given problem
    - Utilises standard control structures such as Begin… End, Do… Until
    - Order of module execution is defined by the programmer
    - Used in data handling programs such as databases and word processers

Other

- Hardware requirements
- Is GUI required
- Experience of the developers

Features requires and available in the language

- Commands within the language to interface with the required hardware
- Ability to run under different operating systems

Factors considered for use of technology

Performance requirements

- Minimum hardware configuration
    - Processor type and speed
    - Primary storage (RAM) available
    - Specific input and output devices
    - Secondary storage size and type
    - A minimum operating system
- Requirements either come from purpose of the software, others from the system being used to convert the source code into executable code

Benchmarking

- Involves creating a set of tasks that can be used to measure the speed with which a computer system will perform a task
- Allows for a comparison between different combinations of hardware and software
- Purely objective process and so subjective measurements of aspects such as user friendliness and ergonomic factors are not included in the process
- Interpreting these results are just as important as obtaining them.

Implementation of software solutions

Language syntax required for software solutions

Railroad diagram

- Alternative graphical method used to define the syntax of a programming language
- Rectangles are used to enclose non-terminal symbols, that is, symbols that will be further defined
- Circles or rounded rectangles are used to enclose terminal symbols
- Linked by paths to show all valid combinations
- By starting at the left-hand side of the diagram and tracing any path in a forward direction to reach the right-hand side of the diagram, a syntactically correct construct is defined

Extended Backus-Naur Forms (EBNF)

- Symbols:
    - <...> represents a defined element (non-terminal symbol)
    - | represents an alternative
    - : = represents a statement is defined
    - {…} represents repeated elements
    - […] represents optional elements
    - (…) represents grouped elements

Metalanguage descriptions of programming languages

Declaring multi-dimensional arrays and arrays of records

In general, declaring a variable for use within a program involves two stages. Firstly, if the data type or structure does not already exist in the language then it must be defined. Secondly, a variable of the required type is declared.

Translation methods in software solutions

- High level languages cannot be directly understood by a processor. Instructions must be converted into binary. Source code □ machine executable code
- Source code is said to be machine independent; can be used on several different processors.
- Executable code is very processor specific, each different family of processors will have different machine language instructions

Compilation

- Takes source code and produces a complete translated file
- Compiling is machine CPU specific
- This file can be used on other machines with same CPU without a need for translator
- Batch process
- Unwanted code, such as comments/remarks, are removed therefore creating code that is more efficient
- Allows for use of share resource and libraries
- Each time a change is made, the whole program must be recompiled
- Slower testing and error detection phase
- All coding errors are reported at the end

Interpretation

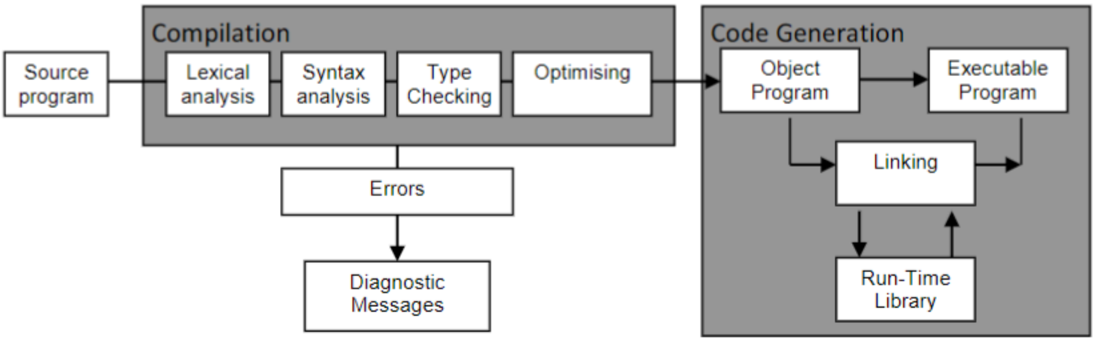
- Translation but slower, less efficient object code as each line is translated when loaded into memory
- Line by line translation and execution
- Much slower overall executions
- Errors reported line by line; instant error feedback
- Early error detection of runtime errors
- When error is detected, execution is halted
- Users must have the interpreter on their computer

Incremental compilation

- Compiles each line of source code and adds it to object file
- Line by line translation
- Errors reported instantly
- Result is still a compiled object file

	Compilation	**Interpretation**
Advantages	Runs faster and more efficient Source code is private	Cross platform Easier debugging
Disadvantages	Executables created with compilers are machine specific; not cross platform If it is to operate on a different processor/operating system, source code must be recompiled, longer to develop in	Slows down execution significantly because of each statement needing to be translated before it is executed Public source Users of interpreted programs must have copy of the interpreter

Compilation process in detail



The three translation methods are similar differing only by what occurs after a line of code in converted.

Lexical analysis (scanning)

- Source code is broken up into lines of code
- All formatting is removed, and comments are removed
- Each group of characters on the line checked against a syntax library held in memory.
- Individual language elements such as constructs called lexemes are identified and labelled with a token
- The process checks reserved words, identifiers, constants and operators are correct
- Identifier tokens are stored in a symbol table or token dictionary
- The translator uses attributes in the symbol table to allocate memory to variables

Syntactic analysis (parsing)

- Parsing implements the syntax rules of the programming language
- If a group of tokens do not conform to the syntax rules the analyser cannot place them in the parse tree □ error is reported

- The analyser reports the error to the programmer
- The tokens pass from the scanner to the syntactic analyser
- The syntactic analyser arranges tokens so computer (CPU) understands logic of program being translated
- Arrangement can be written as a parse tree

Type checking (part of parsing)

- Parsed tokens are sent to the type checker
- Type checker has two purposes
    - Detection of data types within the tokens and passing data type detection to the translator
    - Detecting incompatible operation between data types and creation of error messages

Code generation

- If the process of lexical analysis and syntactical analysis have been completed without error, then the machine code can be generated.
- Involves converting each token or series of tokens into their respective machine code instructions
- No errors should be found during code generation
- The parse tree created during syntactical analysis will contain the tokens in their correct order for code generation and these tokens will be converted into their respective machine code
- Links are created to any external files required at runtime – such as DLLs and other executable libraries

Role of machine code in the execution of a program

CPU

- Fetches the instruction from primary storage
- Decodes the instructions
- Executes the instructions that are expressed and stores the result
- Control unit coordinates the actions of the processor
- Arithmetic and logical unit is responsible for all the arithmetical and logical operations carried out by the processor
- Registers (memories) are provided to store results, locations of instructions and flags

Above components are joined by internal data bus system

The processor is connected to main memory via an external data bus system

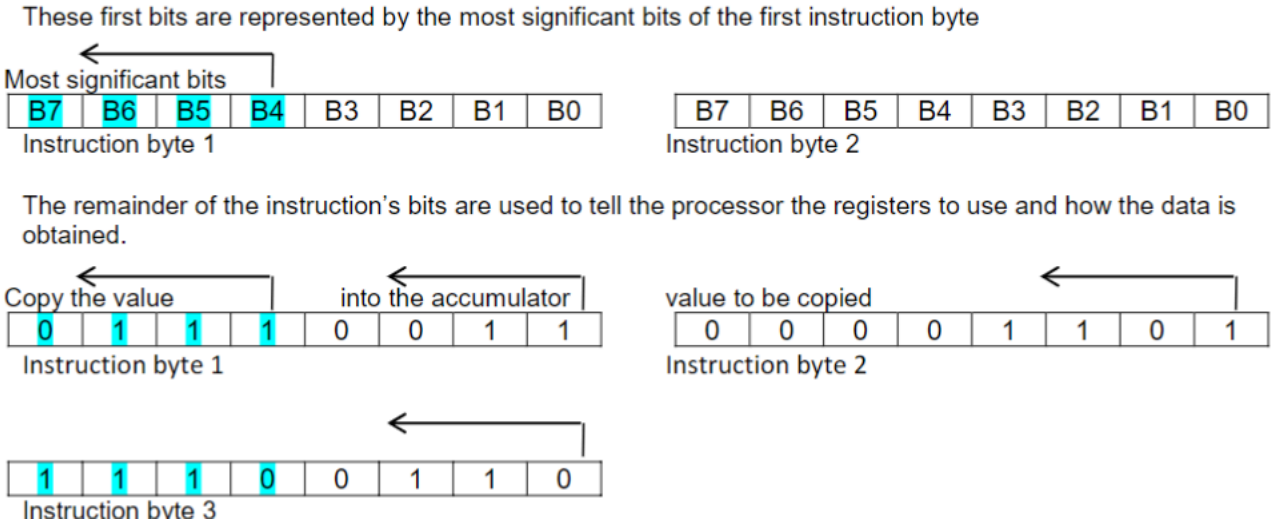
The CPU contains a microcode instruction set which are permanently set in the CPU sometimes referred to as firmware.

Machine code and CPU operation

Instruction format

- Machine instructions need to convey several pieces of information
- An instruction is the command given to the central processing unit by a computer program
- Instructions are made up of an operation code and a memory address
- The first few bits are used to tell the CPU the type of instruction to be carried out
- An operation code is a machine language for a single operation
- The control unit interprets this code to determine the appropriate action

-

- 

These first bits are represented by the most significant bits of the first instruction byte

B7	B6	B5	B4	B3	B2	B1	B0

Instruction byte 1

B7	B6	B5	B4	B3	B2	B1	B0

Instruction byte 2

The remainder of the instruction's bits are used to tell the processor the registers to use and how the data is obtained.

0	1	1	1	0	0	1	1

Instruction byte 1

0	0	0	0	1	1	0	1

Instruction byte 2

1	1	1	0	0	1	1	0

Instruction bvte 3

- First four values represent command 'copy'
- Next four represent the command 'into the accumulator the value that follows'
- Instruction byte 2 and 3 represent the memory location that supplies the data
- This type of instruction is called direct addressing of a memory location
- Processor instructions are grouped into sets

Use of registers and accumulators

- A register is a temporary storage location within the central processing unit
- **Working registers** are referred to as accumulators. Data leaving the main CPU goes through the register buffer
- **Accumulators** hold the data items currently being processed
- an accumulator is a register which stores the result of the latest computation carried out by the CPU or the last data that is about to be computed
- **data is stored in the registers** and results are passed from registers back to main memory
- **special purpose registers** are used to store bits known as flags
- After being in the accumulator, the data must be moved to a different memory location

Fetch-execute cycle

1. Fetch instruction from primary memory and place in instruction register
2. Set program counter to the next instructions address
3. Decode the instruction
4. Load operands from memory into a register
5. Execute the instruction and store result in an accumulator (generally the ALU will execute the task)
6. Reset

Use of a program counter and instruction register

- The program counter is a register that is part of the computer's CPU
- The program counter stores the address of the next instruction to be executed
- Instructions are generally stored sequentially unless the instruction is to change the program counter
- The instruction register is where the instruction is either being decoded or executed is stored

Execution of called routines

- A register called the 'stack pointer' is used to keep track of the location of the beginning of a part of main memory known as the stack
- The stack is then used by CPU to hold the address of the next instruction when a subprogram is run or called
- The CPU can then return to the correct position in the main program when the subprogram is finished
- The stack is a set of locations in main memory used by the CPU

- The stack is a LIFO list
- The last location in the stack to have a value places in it is called the top of the stack

Linking, including use of DLLs

- Linking allows for a machine code program to be combined with another machine code program
- Because larger programs are modular in nature, the linker is used to join these modules
- Since programs must be in the RAM to be executed, linkers save resources because only the needed modules are loaded
- DLLs are a library of machine code programs which can be accessed by others to perform specific tasks
- DLL files allows programs common access to resources and provide efficient use of the memory
- When a program requires a sub procedure it's address is stored in a linker
- The linker calls the sub routine into the main program
- The sub routine gains control
- When finished, control is passed back to the main program
- **Provides method of updating software without rewriting main program**

https://www.youtube.com/watch?v=jFDMZpkUWCw

fetch decode execute cycle steps

1. Program counter stores address of next instruction to be fetched
2. Before fetch instruction memory address must be put in memory address register
3. Contents of memory address loaded into memory data register
4. Instruction placed inside current instruction register
5. Program counter increment by 1 (always points to next instruction before previous is fulfilled)
6. Loads current instruction register into control unit using internal bus links
7. Memory address of memory to be fetch goes into memory address register
8. Take contents of memory location and put in memory data register
9. That contents of memory address placed into accumulator
10. Then fetches next instruction from program counter and puts in memory address register
11. Memory address contents into memory data register
12. Then move to current instruction register
13. Program counter incremented
14. Data moved to control unit
15. Add data to accumulator
16. Contents of accumulator moved to ALU
17. Fetches next memory using memory address register and memory data register
18. Put into accumulator then added to ALU
19. Place result in accumulator
20. Fetches next instruction using same process then stores accumulator into memory address

Techniques used in developing well-written code

Use of good programming practices

A clear and uncluttered mainline

- Software projects should be developed using a clear modular approach
- In general, the mainline should be clear and uncluttered so it is easy to follow and identify subroutine calls
- This means that the subroutines are coded and tested independently which is better for maintenance reasons

One logical task per subroutine

- The name should be able to describe the algorithm succinctly
- When the algorithm becomes too long it becomes more difficult to understand and therefore to maintain

Use of stubs

- A stub is an incomplete function that has the same name and interface as the final form but simpler
- Stubs usually only contain an output statement to indicate the function was called
- It is used in testing the implementation of a program

Appropriate use of control structures and data structures

- Control structures should only use recognised statements such as pre or post-test loops
- The condition for the loop should determine the only exit from the loop
- There shouldn't be jumps that cause control to exit a loop or selection unexpectedly
- Types and unusual cases should be considered when making repetition or selection structures
- Data structures such as arrays, records, array of records and other should be selected and designed to assist processing

Writing for subsequent maintenance

- Coding should be done in such a way that future maintenance programmers can easily update the code
- Good ways to make the code accessible for maintenance includes:
    - Clear documentation within the code
    - Appropriate identifiers names
    - Indenting within control structures

Version control and regular backup

- During coding all programmers should regularly save their work to prevent loss and allow for different versions of the program or module
- Regular backup should be done to various devices instead of single one to prevent absolute loss
- Version control allows complete applications to be continuously developed with regular version being released to users
- Version control allows reversion to a previous version any time in the future
- Version control systems are routinely used by teams of programmers because it routinely does all the above functions

Recognition of relevant social and ethical issues

- Social and ethical issues should be considered when developing:
    - Intellectual property
    - Thorough testing
    - Documentation for future developers and maintainers

The process of detecting and correcting errors

Syntax errors

- Syntax errors are involved in the translation process
- Syntax errors prevent the translating high-level code to object code
- Syntax errors are detected as a result of lexical analysis and syntactical analysis in the translation process
- Once a syntax error has been detected it is usually a simple task to correct

Logic errors

- Logic errors are the result of syntactically correct code that does not complete the required task
- Logic errors are the most difficult error to correct
- It is impossible for the development environment to detect logic errors and logic errors are detected in the running of the program

Runtime errors

- Runtime errors are errors that occur while executing:
    - Arithmetic errors
        - Involves the incorrect use of data types and data structures

        - Usually occurs when the result cannot be stored properly
    - Division by zero
    - Accessing inappropriate memory locations
        - Error occurs when inappropriate memory locations are accessed

Methods of error detection and correction

Use of flags

- Flags are a marker placed in the code to signal a change in status
- Code that will change the value of a flag is added to an appropriate location

Methodical approach to the isolation of logic errors

- The programmer may use the comment tool to stop code from running
- Programmers may also comment out blocks of code and systematically find the precise location of an error

Use of debugging output statements

- Debugging output statements are temporary lines of code added to display the value of a variable
- It may also be used to state that a part of code has been reached

Peer checking

- Involves the checking of the code along with the other person/programmer

Desk checking

- Test data is used to compare the actual result with the expected results
- Use structured walk through to step through the code and find where errors occur
- Desk checking should be performed on sub and whole programs

Structured walkthrough

- Structured walkthrough are more formal than peer checks
- No attempt is made to correct problems only feedback is given
- Made to evaluate design at different levels

Comparison of actual with expected output

- If initial tests actual output should be compared with expected output that has been determined in the planning and design stage
- If actual and expected output are the same, then it is considered an error in logic has occurred
- All paths in logic should be tested
- Often in large productions subroutines or individual modules are tested individuals for logic errors
- Testing software may assist in the testing of logic particularly in large software developments

Use of software debugging tools

Use of breakpoints

- A simple technique which may temporarily or permanently stop executions
- Used to check variable contents before and after processes occur
- Breakpoints are placed in areas where error may exist to locate the source of error
- Development environments usually allows lines of code to be marked with a breakpoint

Resetting variable contents

- The data stored in a variable can be changed while the program is running
- The program is usually halted using a breakpoint
- Can be used to check which values are giving errors by changing the values when execution has stopped

Program traces

- Allows the programmer to view progress of the program execution in detail
- The order of line executions is tracked analysing the flow of the statements
- Particularly useful for following the progress of the program through nested loops or complicated sections
- A log of transactions is often created, and this trace log can be used to identify source of errors

Single line stepping

- A variable trace enables the programmer to observe the variable changing throughout the program
- Single line stepping is the process of halting the executions after each line or statement
- A keystroke is usually required to step through lines
- Usually used to concentrate on a sub routine instead of the whole program

Watch expressions

- A watch expression is an expression whose value is written to a separate watch window
- Usually watch expressions are variables and calculations separate to the program lines of code
- This can provide automated desk checking

Documentation of a software solution

User documentation

User manual and reference manual

- Paper or online external documentation that is user friendly so an everyday user can understand the information presented, and is provided to the user for instructions on how to use the software effectively
- The manual needs to be designed with accordance to the level of expertise the target user had and needs to cater for the experienced and inexperienced
- Should state:
  - How to get started
  - How users may use common functions
  - How to fix mistakes
  - How to recover work that may have been lost
  - Troubleshooting guide

Installation guide

- A program installation guide is also included within user documentation and should include:
  - Details on how to install the program and descriptions of the mediums supplied
  - Minimum hardware specifications
  - Details of any known conflicts with any other software
- It is very useful to use screen shots in the user manual as the users can effectively compare what stage they are up to and confirm that they are on the right track

User tutorials

- Guide the user through the steps of working with features of a program
- Tutorials usually train the user how to use the program

Online help

- Online help systems provide updated solutions to various problems, providing the user with information from the internet – FAQ's are also provided to aid the user
- Having the help documentation online enables easy addition and updating of the help information
- Internet connections are usually required

Technical documentation

Logbooks

- Log kept by the development team of all the steps taken in the development process
- Annotated and dated regularly to avoid loss of important steps and tests

Systems documentation

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

Algorithms

- Primarily used during program design to provide a clear description of the steps carried out to solve the problem
- Allow maintenance programmers to determine the structure of the program

Source code

- Programming code that makes up a program
- Documentation within the source code are in the form of comments and intrinsic information

Use of CASE tools to assist in the documentation process

- CASE tools may be used during the development
- CASE tools use templates for documentation
- CASE tools provide a standardised approach to documentation
- Automates process of documentation

Hardware environment to enable implementation of the software solution

Hardware requirements

Minimum configuration

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

- What type of computer hardware CPU?
- The amount of RAM required for the program to run
- The amount of hard disk space required for the program
- The OS under which the software will run

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 device drivers or extensions

Drivers: additional small programs required to run or execute tasks carried out by peripheral devices

Extensions: files that usually reside with the OS and assist in the execution of some programs

- Dynamic link libraries are extensions

These add-ons are necessary for a program to run successfully and provide all its features

If a program is required to have any of these add-ons, they should be provided by the software developer to the user or have instructions in how to download and install them.

Emerging technologies

The effect of emerging hardware and software technologies have on the development process

Involves learning about:

- New programming techniques
- New programming languages
- Emerging hardware developments
- Emerging software
- The effects software may have on the users
- The effects on traditional developments processes.

Testing and Evaluation of Software Solutions

Alpha testing: Testing of the final solution by personnel within the software development company prior to the product's release.

Beta testing: testing of the final solution by a limited number of users outside the software development company using real world data and conditions.

Two aspects to testing a software solution:

1. Validation: process of comparing the solution with the design specifications
2. Verification: process of ensuring that the software performs its function correctly

Comparison of solution with the design specifications

Design specifications should be written in a form that provides a set of measurable performance criteria. The following guidelines can be used to review the design specifications.

- Ensure specifications are written in terms of measurable outcomes
- Clarify any vague terms (sometimes, usually etc.)
- Clarify and ambiguous statements
- Incomplete lists of items should be avoided
- Calculations should be accompanied by examples
- Pictures and diagrams should be used to clarify structure

If the above is followed in the specification document, an objective evaluation of the project can be made. The process of validation involves the use of the software in a real situation with real data so its performance can be compared under conditions that are close as possible to final environment.

Generating relevant test data

Operational factors outside the control of the programmer:

- User and environment inputs
- Variations in hardware such as CPU, memory, graphics processors, and peripherals
- Different OS
- Changing technologies such as cloud storage and wireless systems

Black box testing

- Also known as functional testing. The inputs and expected outputs are known; the processes occurring are unknown.
- Does not attempt to identify the source of the problem but rather to identify that a problem exists.
- Concentrates on their input and their expected output
- Boundary analysis: test data tests elements that represent either side of the boundary where the effects of processing are different
- Equivalence partitioning: breaking up the input data into groups that have the same properties

White box testing

- Also known as structural or open box testing
- Software testing technique whereby explicit knowledge of the internal workings of the products being tested is used

Statement coverage testing

- Execution of every statement in the module

Decision-condition coverage testing

- Involves testing the execution of each decision in a control structure with a true and false at least one

Exhaustive condition coverage

- Tests all the possible combinations of true and false for each condition
- Bring out any unexpected errors that may occur with unusual data combinations
- Very thorough

Levels of Testing

Testing should occur at:

- Module or subroutine level
- Combination of modules as program testing
- After compilation known as system level testing

Module testing

- Black box
- White box
- Treats every module as a stand-alone application
- Different errors:
  - Arithmetic
  - Comparison
  - Control logic
  - Data structure
  - Input/output
  - Interface errors

Program testing

- Also known as integration testing
- Interaction between modules may causes errors

top down approach

- Program driver module is first tested with stubs representing the lower modules
- Once driver works with stubs, modules are gradually added
- Tested every time you add module until full program is made
- Understand which modules are causing the errors

Bottom up approach

- Involves testing the lower end module first then adding them together one by one then testing, modifying and retesting the larger module until it functions correctly

Followed by **acceptance testing** to ensure that the program fits into the processes and procedures of the system for which it was designed.

System Level Testing

System components to create suitable test environments

Hardware

- Software product should be installed and tested on different combinations of hardware ranging from minimum requirements to additional hardware
- Focus on:
  - Performance: how well program operates; will test load and volume
  - Recovery: force system to fail and then measure the ability to re-establish normal operating state
  - Security: can and how system is breached
  - Stress: push hardware and software beyond normal operating conditions

Other system testing:

- Software: other software installed on the system may have effect on operation of other software products
- Data: data input is the most likely source of errors; needs to be tested with large combination and quantity of live test data
- Personnel and procedures: users of the system will have various skill and knowledge base level
  - Beta testing is useful to test requirements relating to personnel who will use the system

Use of live test data for system testing

- During development, modules are tested with 'designed' data used to test operation. The development team cannot foresee all live scenarios of data.
- Therefore, tested using live data to test operational conditions

Large file sizes

- Use of large files will highlight problems associated with data access.
- Often systems that perform at acceptable speeds with small data files become unacceptably slow when large files are accessed
- Particularly the case when data is accessed using networks

Mix of transaction types

- Testing needs to include random mixes of transaction types
- If data is altered by another transaction, then problems can occur
- Software must include mechanisms to deal with such eventualities

Response times

- The response time of an application is the time the system takes to respond to data input
- Live data will subject the system to real processing requests and therefore will provide a simulation of a real response time

- Any processes that are likely to take more than one second should provide feedback to the user e.g. progress bar

Volume data (load testing)

- Large amounts of data should be entered into the new system to test the application under extreme load conditions
- Multi-user products should be tested will large number of users entering and processing data simultaneously
- CASE tools are available that enable automatic completion of data entry forms to simulate thousands of users entering data to load test during alpha testing

Interfaces between modules and programs

- Provides a communication link between system components
- Interface is usually provided with parameters that are used to pass data to and from modules.
- Require testing to make sure connections between modules work efficiently

Comparison with program test data

- Comparison can highlight any errors or problems
- The output from live data testing is compared with the output of developer testing to determine effectiveness

Benchmarking

- Process of evaluating a product using a series of tests; these tests provide numerical results that are compared with recognised standards
- Benchmark: a point of reference from which quality or excellence is measured
- These results allow users of software products to make informed purchasing decisions
- Benchmarking aims to reduce the number of variables and to report results objectively
- Compare results to competitors to maintain and improve market penetration

Quality assurance

- Clarity- precise and unambiguous instructions
- Correctness – consistent output from the same input.
- Documentation– accurate without spelling grammar and process errors.
- Economy – economical use of software and hardware resources
- Efficiency – ease of completing tasks
- Flexibility– ability to cope with situations encountered within the operation of the program
- Generality– software resembling real processing
- Integrity– maintain system security
- Interoperability – software communicating with existing systems
- Maintainability– correction of problems and errors
- Modifiability – ability to change the software
- Modularity – software parts to be easily be able to be replaced.
- Portability – ability to be executed within different software and hardware environments
- Reliability - a measure of failure rate
- Resilience – ability to recover from abnormal situations and errors
- Reusability – modules may be reused within other similar software projects
- Understand ability – how well the design of the software is understood
- Useability - aspects important to the user, how easy program is to learn
- Validity – how the software meets the specifications of the user

Reporting the testing process

Test description including the test requirements

- States scope of testing
- States purpose and nature of system

- Procedure necessary to prepare hardware and software for testing
- A description of each test including prerequisite conditions, test inputs, expected test results, evaluation criteria, assumptions and constraints
- Connection of relationship of the tests to the requirements

test data including test data and expected results

- Scope of testing
- Overview of system
- Description of test environment
- List of modules to be tested, the tests to be performed and the scheduling of testing
- Connection or relationship of the tests to the requirements

Test report including results and recommendations

- Scope of testing
- Overview of testing outlining deficiencies and impacts on other parts of system
- Steps that may be taken to overcome deficiencies and impacts
- Assesses impact of testing environment
- Test result details

CASE tools

- Test data generator
- Word processor
- File comparator
- WinRunner
- LoadRunner
- DataTech
- UsableNet

Communication of testing

- The nature of the project will determine the audience for the test results
- Large software development companies usually have separate development and testing personnel and testers must communicate their findings back to the development team

Communication with the client

- Language must be non-technical
- Address any requirements and design specifications that have not been adequately fulfilled
- Rank problems in terms of severity
- Upfront and honest about the capabilities of the product
- To assist in the communication, it is common to give a demonstration of the system to the users

Communication with the developers

- Testing departments need to communicate their results back to the development team
- Highlight problems in order of severity
- Recommendations must be backed up by technical justifications
- Actual test data will help developers correcting or modifying product

Evaluating the software solution

Quality assurance and verification of requirements

Functional quality: how well it complies with or conforms to a given design, based on requirements

Structural quality: how it meets non-functional requirements that support the delivery of the functional requirements such as robustness or maintainability

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality.

Post implementation review

- Open discussion with client if requirements have been met to a standard
- For a large development, independent review may be performed to remove any unintended bias

Acceptance testing

- Formals tests conducted to verify whether a system meets its requirements.
- Enables the client to determine whether to accept the new system

Client sign off process

The client will sign off on the project when acceptance testing has been carried out and the client is happy with the outcome.

Maintenance of Software Solutions

Modification of code

Reasons for software maintenance involves making changes to:

1. Meet changed requirements
2. To correct problems or errors in the solution

It is common for changes after the development as user requirement evolve and change. Modifying and maintenance of the problem may involve the structured program development approach. Following this approach ensures effective changes are met.

It is important for software developers to create and maintain the documentation as well. Developer websites often provides the option to report an issue and download a patch for known issues.

Help desk support may also receive requests for changes or identity issues that need to be fixed. Changes would need to be prioritised based on the severity of the issue.

Patch: Used to correct a problem in a software solution. A patch is an actual piece of executable code that is inserted into an existing executable program.

Identification of the reasons for change

Maintenance may be carried out for the following reasons:

1. Errors in code
2. Changes in the operating environment

3. Changes in requirements, such as increased functions required by the client

Examples of reasons for modifying code:

- Poor logic in original code
- Changing user requirements
- Improvement for efficiency
- Improved security
- Upgrading the user interface
- Changes in data to be processed
- New hardware and/or software
- Changes in business requirements
- Changes in government requirements

Operating environment changes may include:

1. Changes in the programs that interface with the program;
2. Changes in the operating system
3. Changes in hardware and associated utilities/drivers

Location of the section of code to be changed

- When an error in a program has been detected the developer must be able to find the location within the source code.
- Well-structured and well documented code will simplify the process of isolating and correcting errors.
- When a modification is to be made, design documentation and source code will be used to determine the exact location of the change is required

features that improve maintainability may include:

- Well set out design documentation – such as IPO, data flow diagrams, structure charts, algorithms, and source code;
- Use of variables instead of literal constants
- Constants or global variables defined at the beginning of code
- Use of meaningful identifiers (internal documentation)
- Design comments/remarks throughout code (internal documentation)
- Code that incorporates a good error reporting system
- Use of correct control structures
- Modularisation of code
- One logical task per function/sub routine

Determining changes

There may be different reasons for changes to be made:

1. Changes due to logic will require source code changes
2. Changes to interface design may involve minor layout adjustments or a completely new layout design
3. Addition of new functions and requirements

How are changes made or implemented?

Once errors have been detected and located how will the change be implemented.

Methods for correcting code may include:

1. Minor source code adjustments
2. Software patch which may include several minor source code adjustments
3. New version implementation which may include redesign pf several functions
4. Software reengineering may be required if several modules need improving
5. A completely new system development

All changes should be thoroughly tested before being fully implemented. Even minor source code adjustments will need to be compiled.

Regression testing is the process of retesting a program when changes have been made.

Documentation of changes

Documentation should always be carried out when making changes to software. It is the responsibility of the software maintainer to document changes.

The process of keeping track of software changes, versions and documentation changes is part of project management. Management of software resources is called configuration management.

Documentation should be accurate and up to date reflecting changes as they occur.

Source code, macro and script documentation

Documentation of source code, macro and scripts may include:

- Modification notes within source code headers
- Comments and remarks
- Reprinting program code listings
- Developers process diaries
- System developers notes

Modification of associated hard copy documentation and online help

Some changes may have little effect on user manuals and help systems

Where changes affect manuals and help systems changes may be accompanied by:

- Release notes
- Addendum to manuals
- Reprint of manuals or providing new manuals in pdf
- Re install of help files

Programming Paradigms

Development of different paradigms

Programming Paradigm

- Computer programs are written in a paradigm or model
- A paradigm is a pattern, an example or a model.
- **Paradigm:** philosophical or theoretical framework. A different approach under which laws, theories and generalisations are produced.
- Most computers use the Von Neumann computer architecture model:
    - Memory and processing are separated
    - Algorithms are expressed in a sequence of commands
    - CPU executes the lines in sequence
- Examples of different programming paradigms are:
    - Imperative: variables and control structure
    - Logic: non-procedural, declarative
    - Object-oriented: event driven

Historical development of programming languages

First generation – Machine Language

- Binary (0's and 1's)
- The operation codes correspond to actions in the fetch-execute cycle
- Process dependent therefore not transportable

- Only language which computer can understand
- Specific to the CPU and BIOS

Second Generation – Assembler Language

- Binary was replaced with mnemonic code, which uses short code words
- Processor dependent, therefore not transportable
- Easier to remember and read
- Introduced labels and variables
- Converted to machine language using assembler
- E.g. Autocoder

Third generation

- High level
- Processer independent
- Portable across platforms
- Sequentially executed line by line
- E. BASIC, PASCAL

Fourth Generation

- Very high-level languages
- Non-procedural
- Query and retrieval
- What program must do instead of how it must be done
- Developed to make interaction with user easier
- Errors are easier to detect
- SQL

Fifth Generation

- Known as natural and knowledge base languages using spoken English language
- Uses logic programming paradigm
- Employ problem solving algorithms based on logic
- ppl0**6h**E.g. PROLOG

A procedural language is where a sequence of execution is written by the programmers usually in a main line program. All procedural languages use the concept of a variable as a storage location.

With improvement of hardware came use of Graphical User Interfaces

Due to GUI **event driven language and object-oriented languages OOP** were developed. In **Event-driven programming** the order of execution is not sequential; non-linear and controlled by events.

Non-procedural languages were developed to provide more flexibility in the way problems may be solved; rely on logic and inference.

Limitation of the imperative paradigm

Difficulty with solving certain types of problems

- Computers were initially designed to solve mathematical calculations
- Many real-life problems do not have definitive answers, or they may have a variety of answers
- Situations in which strategies to solve are not purely mathematical render the imperative paradigm unsuitable

The need to specify the code for every individual process

- The imperative paradigm requires that every part of the problem must be solved for the entire product to work

Difficulty of coding for variability of problems

- Ability to reuse code is inherent
- Preferable if code could be used for different problems without rewriting the code

Emerging Technologies

All commonly used CPUs process instructions sequentially. Therefore, all higher-level languages still need to be translated into machine code that will be processed sequentially.

Emerging Technologies development may lead to non-sequential processing.

Some emerging technologies may include:

1. Artificial Intelligence AI: using multiple processors, multi-threading and multitasking
2. Quantum Computing: involves use of atomic particles
3. The Internet of Things IoT: refers to a system of interconnected devices that speak to each other in real time, collect data and function as a single system.
4. Cloud Computing: networks that allows information applications to be stored and shared between any number of devices. It allows us to access our digital files and applications from anywhere.
5. Security, Privacy and encryption of data: hardware and software developments including encryption to keep data safe from unauthorised access over networks.
6. Virtual Reality VR: refer to immersive hardware and software, originally for the entertainment industry, with application in a range of other industries including military, engineering, education and training.

Imperative/Procedural Programming paradigm

- **Lower level programming** has a dependence on the machine architecture and memory allocation.
- Use procedural programming languages
- Memory locations are represented as variables and parts of the program may be executed repetitively
- Various control structures are used to govern the sequence of execution
- First to third generation languages use this programming paradigm

Imperative languages are suited to the translation process that interprets one line of code at a time. Contain the following constructs:

- Non-repeated processes in sequence
- Decision constructs
- Repetition constructs

Procedural programming is a programming paradigm based upon the concept of the procedure call. Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

Procedural programming is often a better choice than simple sequential or unstructured programming in many situations which involve moderate complexity, or which require significant ease of maintainability. Possible benefits:

- The ability to re-use the same code in at different places in the program without copying it
- Easier way to keep track of program flow
- Ability to be strongly modular or structuredinters

Object-oriented programming (OOP)

Object-oriented languages:

- Are based on a graphical user interface
- Are event driven programming

Important features:

- Emphasis on data rather than procedure
- Programs are divided into objects
- Data is hidden and cannot be accessed by external functions

- Objects can communicate with each other through functions
- New data and functions can be added easily whenever necessary
- Follows bottom up approach

Classes

- Blueprint that defines the variables and the methods common to all objects of a certain kind.
- Expanded concept of a data structure; instead of only holding data, it can also hold functions and methods
- Specifies a data structure and the permissible operations or methods that apply to each of its objects
- Abstract data types in object-oriented languages are called classes; abstractions refer to the attributes or properties of that data item
- A class generally refers to a group or collection of similar objects
- A class defined through inheritance from another class is called a subclass
- The class from which the subclass is derived is called the parent class or superclass
- Objects are accessed like variables with complex internal structure, and in many languages are effectively pointers.

Objects

- An object is an instantiation of a class
- An object is a software bundle of variables and methods
- Instantiation is the use of object classes

Methods

- How the objects data is processed
- Methods are implemented as individual programs
- Subprograms that defines the operations on objects of a class are called methods
- Class methods
    - Belong to the class as a whole and have access only to class variables and inputs from the procedure call
- Instance methods
    - Belong to individual objects, and have access to instance variables for the specific object they are called on, inputs, and class variables

Abstraction

- An object contains some properties and methods
- We can hide them from the outer world through access modifiers
- Abstraction allows the programmer to simplify the programming process. In a modular solution, abstraction can be viewed in levels
- Two essential forms of abstraction:
    - Data abstraction, which is looking at data items in familiar terms such as characters and numbers
    - Process abstraction, which is a sequence of instructions that have a clearly defined and limited purpose

Messages

- A message is a request that asks for a specified operation to be initiated using one or more objects

Inheritance

- An object class may be composed of several different object types
- The concept of classes automatically containing the variables and methods defined in their supertypes

Polymorphism

- Allows different objects to use a behaviour or method
- Used for maintenance and expansion of code

Encapsulation

- Separates how an object behaves from how it is implemented

- Is the process of including all the attributes and procedures that an object needs within the object itself
- Disallows calling code from accessing internal object data and forces access through methods only.
- Result of packaging the data and methods of an object in a way that prevents the data from being accessed except via the objects own methods
- Some languages let classes enforce access restrictions explicitly, for example denoting internal data with the private keyword. Vice versa for outside classes with public keyword.

OOP and game development

- Most commercial games are written in C++, C and some assembly language
- Many games tax hardware to it's limit and therefore, highly optimized code is needed for these games to run at an acceptable frame rate

Web based data application

- In object-oriented database management, information is stored as an object
- Each object consists of two things
  - One is the data (sound, text video)
  - Other is instruction for the software
    1. Instruction is how the data will be transferred
- Saves time and money in maintenance
- Web applications are a good use of object-oriented programming

Logic Paradigm

- Logic paradigm is non-procedural
- Associated with artificial intelligence software
- Logic programming is an attempt for software to do what human beings do such as:
  - Ability to learn
  - Process knowledge
  - Make inferences
  - Gain knowledge
- Logic programming uses a set of statements that are executed in an order to produce or infer a result or truth

- Three types of statements in logic programming
  - Facts
  - Rules
  - Queries
- A collection of facts and rules is known as a knowledge base and Prolog programming is all about writing knowledge bases

Facts and rules

- Referred together as clauses

Heuristics

- Involves finding a set of rules and procedures to find a satisfactory solution to a problem
- Provide different criteria for chaining to a solution or goal

Goal

- Goals are achieved when a group of rules from the database are matched
- Goals are achieved when the facts and rules are true

Inference engines

- An inference is the processing portion of an expert system
- It is the control engine and provides the reasoning ability providing conclusions
- An inference engine searched through the knowledge base using forward and backward chaining

- Inference engines carry out the processing and applies a knowledge base to resolve a goal

Backwards chaining

- Involves matching the current goal against each fact
- If a goal matches a fact the goal succeeds and the resolution proceeds.
- If it matches with a rule with all the sub goals the rule succeeds
- The user is informed of the success or failure of the choices made and the variables that were input to get to that point
- Starts with a statement and a set of rules, and works backward, matching the rules with the information from a database of facts
- A process continues until statement can be proven true of false
- Eventually the source of the material that can be hypothesised is met

Forward chaining

- Problem solving procedure that starts with a set of rules and a database of facts and works to a conclusion based on facts that match all the premises set forth in the rules
- This methodology works best when there are many results

Recursion

- Occurs when a rule contains itself and is a repetition mechanism
- Would normally cause a stack overflow error if it continues indefinitely
- Method of solving a problem where the solution depends on solutions to smaller instances of the same problem
- Recursion solves such problems by using functions that call themselves from within their own code

Logic paradigm and pattern matching

- Pattern matching is the automated recognition of patterns and regularities and data
- Pattern recognition is closely related to artificial intelligence and machine learning, together with applications such as data mining and knowledge discovery in databases
- Involves the automatic discovery of regularities in data using computer algorithms and with the use of these regularities to act such as classifying the data into different categories
- Pattern recognition systems can be supervised learnt or unsupervised learnt, in which they are used to discover previously unknown patterns
- Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to perform “most likely” matching of the inputs
- Pattern recognition is the basis for computer aided diagnosis which has a procedure that supports the doctor’s interpretations and findings

Logic paradigm and artificial intelligence

- A typical AI analyses it’s environment and takes actions that maximises its chances of success
- Utility for an AI can be simple or complex, and goals can be simply defined or induced.
- If the AI is programmed for “reinforcement learning”, goals can be implicitly induced by rewarding positive behaviour and punishing negative behaviour
- Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future.

Logic paradigm and expert systems

- An expert system is designed to act like an expert in a field or application
- Sometimes called knowledge-based systems
- Expert system is designed to deal with uncertainty within its problem solving function
- The logic of the expert system has a way to deal with the uncertainty
- Three main sections:
  - Knowledge base
  - Inference engine
  - User interface

Knowledge base

- Consists of a database containing simple facts and rules that describe relationships and possible methods
- Database may have the ability to grow as it learns new facts
- This is how it deals with the uncertainty apparent in the paradigm

Inference engines

- Processing portion of an expert system
- Control engine
- Applies knowledge to the data to form a conclusion
- Forward and backward chaining

User interface

- Provides smooth and clear communication between the user and the system
- Should give an insight to the problem-solving process being carried out

Heuristics

- Rule of thumb often based on experience
- Results in one or more possible solutions
- Rules that are generally accepted as true within the specialist area
- Gives expert system a human feel

Neural networks

- Expert systems that can learn
- Used in the future to reach new conclusion
- Each neuron has several inputs and a single output
- Each input is given a different weighting and once weighting has passed a threshold a neuron is fired

Issues with selection of appropriate paradigm

Nature of the problem

- Different paradigms provide the best structure and coding language choice to solve the problem
- Aim of the most appropriate paradigm is to increase productivity and enable the production of efficient and reliable solutions

Speed of code generation

- High level languages faster

Approach to testing

- Structured
  - Always testing
- Program
  - Testing at end
- Automated
  - Case tools

Effect on maintenance

- Use of reusable modules
- OOP is recommended

Efficiency of solution once coded

- Low-level languages are the fastest to execute but take the longest time to develop.

- High-level languages are the slowest to execute but take the shortest time to develop.
- Imperative languages are often more efficient for basic sequential flow applications, as they work closely with the hardware using variables, assignments and identifiers and follow sequential structures.
- Object-orientated languages are more efficient for the development of user productivity but are less efficient in their speed of execution.
- Logical languages can often result in excessive / inefficient processing requirements due the computer's ability to find its own solution based on facts and rules, the processing path is not specified by the programmer.

Learning Curve (training required)

- Imperative and object-oriented are easier to learn as they are considered more popular and are often introduced to students as variables and standard basic constructs are easy to understand for a beginner
- Despite OOPs being 'visually' easier to learn, they still require training to fully understand their full capabilities.
- Logic programming languages are often thought out to be harder and specialised, although these are often easier to teach to children as they don't require as much code or constructs to be coded. They rely on basic mathematical principles with rules and facts, so beginners can often relate these principles to the coding of these rules.