Modularity

- The idea behind modularity is that of splitting up the problem into a series of self-contained modules.
- In practice it is advised that a module should not exceed 100 or so lines and preferably be short enough to fit on a single page.

Advantages of modularity

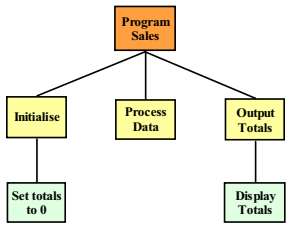
- Some modules may be defined by standard procedures which are used and reused in different programs or parts of the same program.
- A module is small enough to be understandable as a unit of code. Thus also easily debuggable
- Program maintenance is easier
- Several programmers may work on different modules concurrently
- Modules can be tested independently
- Large projects become easier to monitor and control

Jackson Structured Programming

- This design methodology is ideally suited to problems that can be expressed as a hierarchy of data structures.
- In particular those described by a top-down approach
- Operations include:
  - Sequence: represents a sequence of operations
  - Iteration: or loop as in while..do
  - Selection: conditional selection as in if..then..else

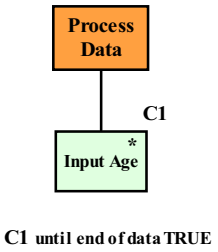
Sequence

- In a structured diagram, the levels are important since an operation can be made up of a sequence of other operations
- A is made up of B, D and E and in turn B is made up of C and E is made up of F



Iteration

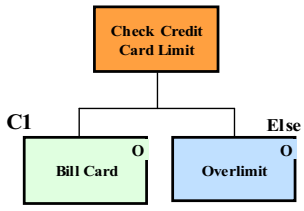
- The notation for a loop is an '*', which is written in the top right-hand side of a box which represents the iteration loop.
- A condition has to be defined
  - Normally referenced by a number
  - Number, references actual condition in a conditions list



C1 until end of data TRUE

Selection

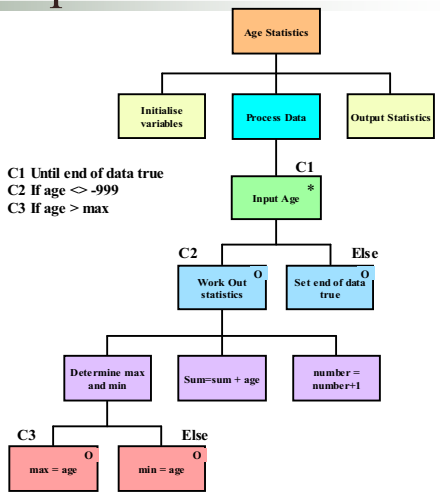
- Selection is similar to an if…then…else statement
- Representation is similar to an iteration except that the selection symbol is a small 'o'.
- Branching depends upon the condition



C1 IF limit not exceeded

Complete Example

Important to note that it is not allowed to mix different operations in the same level, i.e. children of the same process should be of the same type



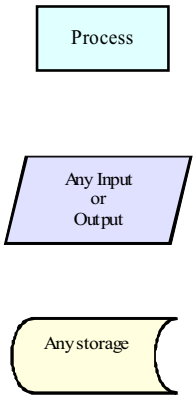
Prototyping

- Refers to the use of tools, such as CASE tools (computer-aided software engineering), that aid in quickly developing, testing and evaluating the software. Based on a **spiral model**.
- A prototype, which may be created in a few days, allows users to find out immediately whether the system represents/solves the customer's needs.
- The process can be thought of as a build and refine process. In the end a number of prototypes may have been created, before the actual system is developed. This is called throw-away prototyping
- One advantage of such methodology is the fact that the customer is constantly being consulted during the software building process

Systems flowcharts

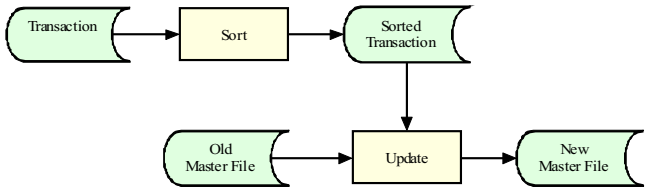
- Shows an overview of a complete system, by representing:
  - Tasks to be carried out in the new system, whether they are manual or computerised
  - Devices (disks, terminals etc) that are to be used in the system
  - Media used for input, storage and output
  - Files used by the system

Systems flowchart symbols



Example

- A customer file is held on disk. Receipts are held in a transaction file and are sorted and are then used to update a master file, by creating a new master file.



User Interface

- A good interface design is an important aspect of a successful system. Design should consider:
  - **Who**: will be using the system, whether the users are experienced or not etc
  - **What** tasks: will the computer perform, whether its repetitive, life-critical etc
  - **Environment**: whether this is noisy, hazardous
  - What is **technologically feasible**

User Interface [cont]

- Particular attention should be given to the design of the screens. Especially:
  - This should not be too cluttered
  - Use of colouring schemes
  - Shortcuts
  - Visibility of important functionality
  - Logical sequencing of items
  - Use of appropriate input validation and error messaging

Program Coding or Development

- This is a time-consuming stage.
- Requires a lot of effort (hence expensive)
  - Work is measured in man/months
- Aids to save time and money may include:
  - using existing library routines, subroutines and subprograms
  - other programming aids such as
    - screen painting software,
    - report generators,
    - application generators,
    - RAD (Rapid Application Development) tools.

Program testing and debugging

- **Program Errors :** A program may have any or all of the following types of errors:
  - Syntax Errors: A statement in the program violates a rule of the language,
  - Semantic Errors: Violating rules of language, semantic errors are concerned with the meaning of language statements (semantics).
  - Logical Errors: The program runs to completion but gives wrong results or performs wrongly in some way.
  - Runtime Errors: Program crashes during execution.
- The translator detects syntax and semantic errors but it does NOT detect Logical and Run-time errors. These require rigorous testing for detection and correction.

Program Debugging

- Aids and techniques:
  - single stepping through the program: use of watches
  - setting breakpoints
  - displaying contents of specified variables in memory
  - dumping and examining the contents of a file
  - dumping and examining the entire contents of memory

Stages of Testing

- May be quite a lengthy and expensive process.
- Stages of testing:
  - Desk–checking /Dry run: programmer follows through code manually, using test data to check that an algorithm is correct
  - Unit testing: testing of each individual subroutine or module, also referred to as Module testing:
  - Integration testing: software testing in which individual software modules are combined and tested as a group
  - System testing: is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements
  - User acceptance testing
    - Alpha testing in the case of bespoke software
    - Beta testing in the case of packaged product

Testing Strategies

- There are two major testing strategies:
  - Black box or Functional Testing: Involves looking at the program specification and creating a set of test data that covers all the inputs and outputs of the program functions
  - White box or Logical Testing: this depends on the code logic and derives from the program structure rather than its function. The code is studied and tests are created to test each possible path at least once.
- Remarks:
  - Black box testing is not adequate by itself.
  - White box testing by the programmer during program development is most effective, though it will not detect missing functions.

Documentation

- **Program Documentation should contain:**
  - A description of the problem to be solved.
  - A program abstract that describes the various tasks which the program performs (e.g. the files used, etc).
  - Difficulties encountered during development and how these were solved
  - Operating instructions on how to run the program.
  - A summary of the program controls which are built into the program.
  - A test, plan and data used to test the program for accuracy. Any changes made during testing should also be recorded

Types of Documentation

- User manual: aimed at the various end-users of the system
- Operational: documentation of procedures necessary to run the system .
- Technical: important to ensure that the system can be maintained after completion

User Manual

- Typical information in a user manual:
  - Overview of options available
  - Guidance on the sequence of operations to follow
  - Screen shots showing screen input forms
  - Instructions on how to enter data
  - Sample report layouts
  - Error messages that may be displayed and which action to take

Operational Documentation

- Typical information:
  - System setup procedures, including details for each application, the files required and consumables requirements
  - Security procedures
  - Recovery procedures in the event of system failure
  - A list of system messages that might appear on the operator's console and what action to take

Technical Documentation

- Typical information in a technical manual include:
  - Overall system plan
  - Data organisation and flow
  - Full annotated listing
  - Details of test data and results

Systems Development

System Implementation,
Evaluation and Maintenance

Implementation

- Implementation phase may include a number of activities:
  - acquisition of new hardware and its installation.
  - When installing new hardware, this may involve extensive re-cabling and changes in office layouts.
  - Training the users on the new system
  - Conversion of master files to the new system, or creation of new master files.

Evaluation

- Evaluation will help those responsible of the implementation to judge whether the expenditure on the new system has been justified.
- Evaluation also helps to improve the quality of the future developments by highlighting what worked well and what caused problems
- Some areas to be covered in this evaluation:
  - **Accuracy**: identify whether the output is consistently accurate during day-to-day operations
  - **Quality of output**: determine the quality of information outputted from system (is it accurate, reliable, timely)
  - **User satisfaction:** see that the users are comfortable with the new system and any encountered problems are resolved quickly
  - **Reliability:** this is judged by how often the system breaks down
  - **Performance:** speed of performance evaluated
  - **Controls and Security:** ensure that system is fully protected from unauthorised access and the integrity of the data used is highly protected

System Maintenance

- Maintenance is generally triggered by requests for changes from system users and or by management
- Maintenance is a very expensive process
- Therefore more cost effective to put time and effort into the development phases
- Level of maintenance depends on:
  - Environment and type of software: e.g real-time applications necessarily have to change
  - Increased complexity: due to evolution, programs tend to become more complex, and therefore more maintenance requires more effort

System Maintenance [cont]

- Perfective maintenance
  - This implies that while the system runs satisfactorily, there is still room for improvement.
- Adaptive maintenance
  - All systems will need to adapt to changing needs within a company.
- Corrective maintenance
  - Problems frequently surface after a system has been in use for a short time, however thoroughly it was tested. Any errors must be corrected.

Factors affecting maintainability

Maintainability depends on:

- Good programming design
- Well structured programs written in a modular fashion and in line with standards of best practice
- Use of appropriate high-level language and good system and program documentation
- The availability of a record of all maintenance work carried during software evolution

Systems Development Cycle