Systems Analysis

System Life Cycle (or **SDLC** - **System Development Life Cycle**) is the process of developing an information system from its beginning to its end, including the time when it is developed, used, modified and finally becoming obsolete.

There are many approaches to implement a system. One such method called the **Waterfall method**. It is made up of a sequence of stages that are shown here.

1. Identification of the problem
2. Feasibility study and project selection
3. System analysis of present system
4. System design of new system.
5. Programming and testing
6. Implementation
7. Changeover methods
8. Control and review
9. Maintenance

Identification of the problem

What can prompt an organization to develop a new information system? Two possible reasons can be:

- The organization has grown to new heights and its information system needs to be changed.
- Technological developments have made the present system obsolete.

Feasibility study and project selection

A **Systems Analyst** (i.e. the person responsible for the development of an information system) investigates the present information system and suggests a rough estimate of how much a new system will cost and how long it will take to be ready to be used.

The systems analyst can suggest more than one new system. The analysis of the present system can be done by means of:

1. Interviews
2. Questionnaires
3. Inspection of documents
4. Observation (of existing system)

The feasibility study will take into account not just the cost of material and labour together with the time that will take a project to be finished. It will consider environmental, legal and also social aspects.

After the submission of the feasibility report the company will select which project is to be implemented.

System analysis of present system

During this phase the systems analyst will carry out very detailed investigations in order to fully understand the current system. The following and other questions will be asked:

- How staff / customers interact with the current system i.e. how tasks are carried out
- How other systems interact with the current system
- What is good about the current system
- What causes problems with the current system
- Which parts of the system are critical to the business

Then the systems analyst ponders about the following:

- what the new system is expected to be able to do (i.e. system requirements – processing, data storage, input and output formats)
- how the new system is expected to do this
- what people want from the new system
- which working methods from the old system should be incorporated into the new system

The systems analyst considers whether '**off-the-shelf**' solutions exist thus avoiding 'reinventing' the wheel. Off the shelf solutions can be **customizable** to one's system. The parts that are programmed are called '**tailor-made**' or '**bespoke**'.

Both off-the-shelf and tailor-made solutions have their own advantages and disadvantages.

Advantages of off-the-shelf

- Is cheap
- Is immediately available

Disadvantages of off-the-shelf

- May not be exactly what one would like, it may have some missing features or some extra features that one would not require.

- Often, they need patches.

Advantages of tailor-made

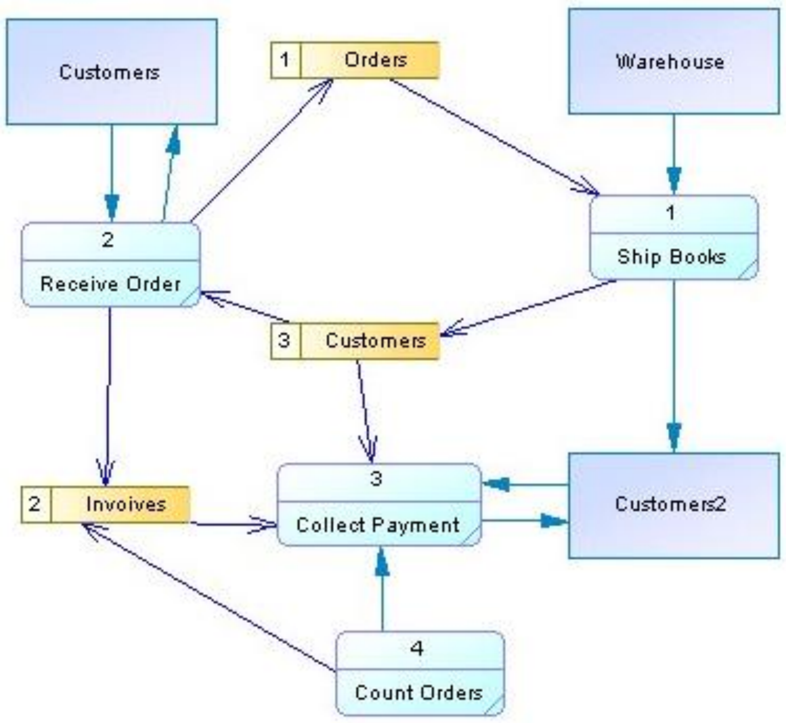
- It is perfectly written for one's system
- Can be easily upgraded later on

Disadvantages of tailor-made

- Costly
- Not immediately available

Information about the old system can be gathered from interviews, questionnaires, surveys, inspection of documents and observation (i.e. being physically present).

During the analysis and design phases, many diagrams are used to represent the workings of a system or a part of it. The following diagram is an example of such pictorial explanations.



System Design

During this stage the systems analyst:

- Must describe how the new system will work.
- Must describe the following:
    - Design of **screen layouts** (**user interfaces**)
    - Design menus

    - Design inputs (how inputs will be passed on to the system)
    - Design outputs (printer, screen, reports etc.)
    - Design of report formats
    - Decide what hardware configuration is required
    - File contents and organisation
    - Program specification of each program in system using:
        - Pseudocode, or
        - Flowcharts or other means
    - Decide what conversion plan is to be used (see later on)

Tests for the programs are prepared at this stage including the **test data** and the **testing strategies**.

Human Computer Interfaces

Interface styles (screen-keyboard-mouse interfaces)

- **Command line interface**
    - Instructions are typed
- **Menus**
    - Full screen menu e.g. front-end of an application
    - Pull-down menu
    - Pop-up menu
- Natural language
- Forms and **dialogue boxes**
- **Graphical user interface** (**GUI**)
    - Also known as **WIMP** (Windows, Icons, Mouse or Menus, Pull-down menus or Pointer)
- Touch-screen

Other interfaces:

- Speech input (voice recognition)
- A speech synthesis system (for output)
- Joystick etc.

Testing

There are various types of testing. Some are mentioned hereunder. The tests, together with test data, are prepared at this (design) stage.

- **Dry run** testing:
    - The programmer follows the code manually.

    - A trace table will help.
- **Black-box** testing (functional testing):
    - Testing the program without looking at the code.
- **White-box** testing (structural testing):
    - Each possible path of the program is tested.
- **Unit testing**:
    - Testing each unit individually (e.g. method, class) and make sure that it functions correctly.
- **Integration testing**:
    - When all modules have been individually tested, integration testing ensures that they work together (as one whole program or system) correctly.
- **Alpha testing**:
    - Performed only by users within the software house developing the software.
- **Beta testing**:
    - This involves giving the package to a number of potential users who agree to use the system and report any problems to the developers.
    - The product is corrected and can be sent out for further beta testing until the developer is confident enough in the product to put it on the market.

Documentation

All phases during the SDLC are documented.

Documentation is an important part of software engineering. Types of documentation include:

- Requirements: i.e. a description of what is required out of the system. This is the foundation for what ought to be or has been implemented.
- Architecture/Design: i.e. the plan of the new computer system.
- Technical: i.e. documentation of code, algorithms, diagrams etc. This documentation is required for the system maintenance and upgrade.
- User documentation e.g. manuals (for the end-user, system administrators and support staff), tutorials etc.
- Marketing: i.e. how to market the product and analysis of the market demand.
- Description of files used.
- Tests performed on software.

Coding

One can divide the writing of a program in these steps:

1. If the problem is large divide it into a number of modules. Different modules can be assigned to different programmers.
2. To program a module first express it as an algorithm – as a flowchart, in pseudocode or in another way.
3. Write the code by following the logic of the algorithm. An issue here is what language to use. If what you need to program is a user interface then maybe a visual program (Visual Basic etc.) will best suit your needs. If you are to program a systems program then a low-level language will help you best. If you need a mathematical program the language FORTRAN is a very good candidate for your choice.
4. Test the module and correct as necessary. The testing has to be done by following the testing strategy devised in the design phase.
5. Test that the modules work well together (integration testing) and make the necessary corrections if required.

It is also important to have a **debugger** with quite a number of features. Some features of debuggers are:

- **Breakpoints** (a breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point).
- Stepping through the code
- Viewing the values of variables (**tracing**)
- See the values returned by functions
- Modifying the values of variables while you are executing and debugging a program
- Go back to a place in the program that has already been executed to see what the values of the variables at that stage were.

Implementation

During this phase:

- All programs are coded and tested.
- Necessary hardware and software are acquired.
- Users are trained for new system.
- New master files for the new system are created.
- Conversion from the old system to the new one takes place.
- Creation of master files.

Conversion from Old to New System

There are various methods of conversion:

- **Direct changeover**.
    - The user stops the old system on one day and starts using the new system the next.
    - Advantage: fast and efficient with minimum duplication of work involved.
    - Disadvantage: if errors are found in the new system operations could be seriously disrupted.
- **Parallel conversion**.
    - The old system continues alongside the new system for a few weeks or months.
    - Advantages: (1) results of the new system will be checked against known results from the old system, (2) if difficulties occur work can continue under the old system.
    - Disadvantage: duplication of work.
- **Phased conversion**.
    - It is used with a system that can be broken down into modules that can be implemented separately at different times.
    - Could be direct or parallel.
- **Pilot conversion**.
    - The new system will be used first by only a portion of the organisation.

Maintenance

The system would require some maintenance every now and then to keep functioning correctly and at top efficiency. There are different types of maintenance:

- **Perfective maintenance**: the system is improved e.g. database response to queries made faster.
- **Adaptive maintenance**: changes are required e.g. new government legislation may mean that different methods of calculating tax are required.
- **Corrective maintenance**: i.e. correcting errors in code.
- **Predictive maintenance**: e.g. making changes before predicted failure of some devices.