

Alternative Development Life-Cycles

Why have a Life Cycle? People like to organise things in phases or stages. Consider how an author writes a play or a composer composes his symphony. This concept of organising into phases extends into engineering, civil and mechanical engineers use a development life cycle. This idea of phased or staged progress has been taken across into the software or system development task.

There are two main life-cycle models, “Single-pass” life-cycles, for example the Waterfall model as given by Royce show a single pass through each stage. On the other hand the NCC SLC Cyclic model (revolutionary) or spiral model (evolutionary) implies iteration or “multi-pass”.

The different development life-cycles may be specified as part of a method, for example the SSADM life cycle shown in figure 1; but some methods are independent of life-cycles.

SSADM development lifecycle

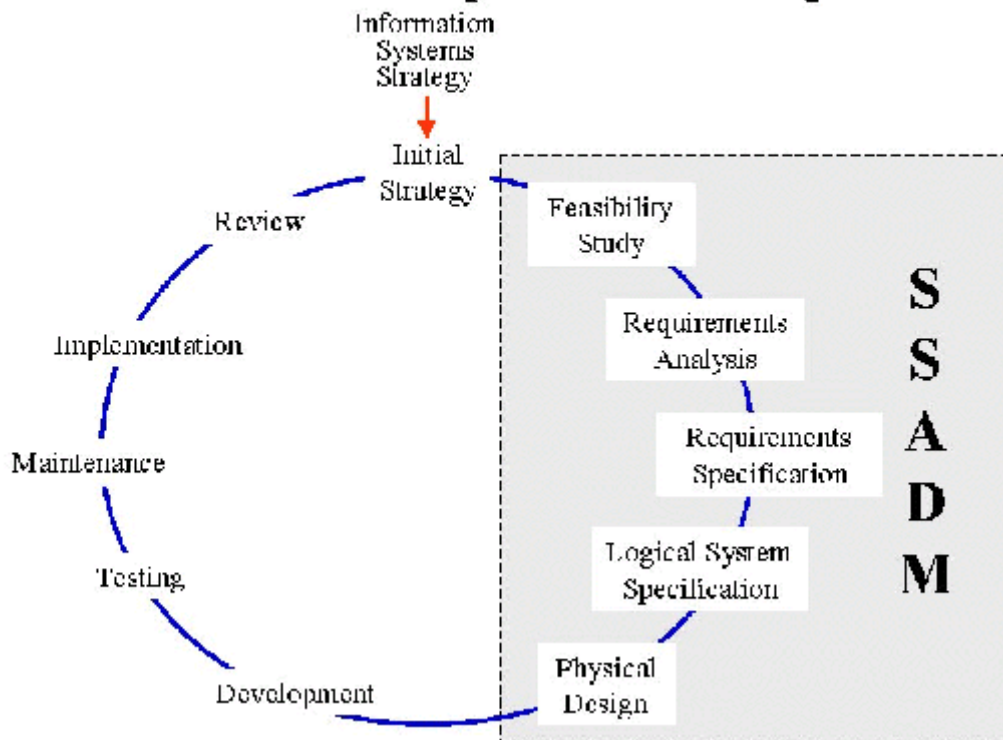


Figure 1

Some organisations allow the use of multiple standards, whilst other companies will develop their own system life-cycle. For example Oracle has developed a methodology called CASE*Method. This methodology is described in Richard Barker's two books "CASE*Method - Entity Relationship Modelling" and "CASE*Method - Tasks and Deliverables", (both published by Addison Wesley). In his books he uses the life cycle shown in figure 2.

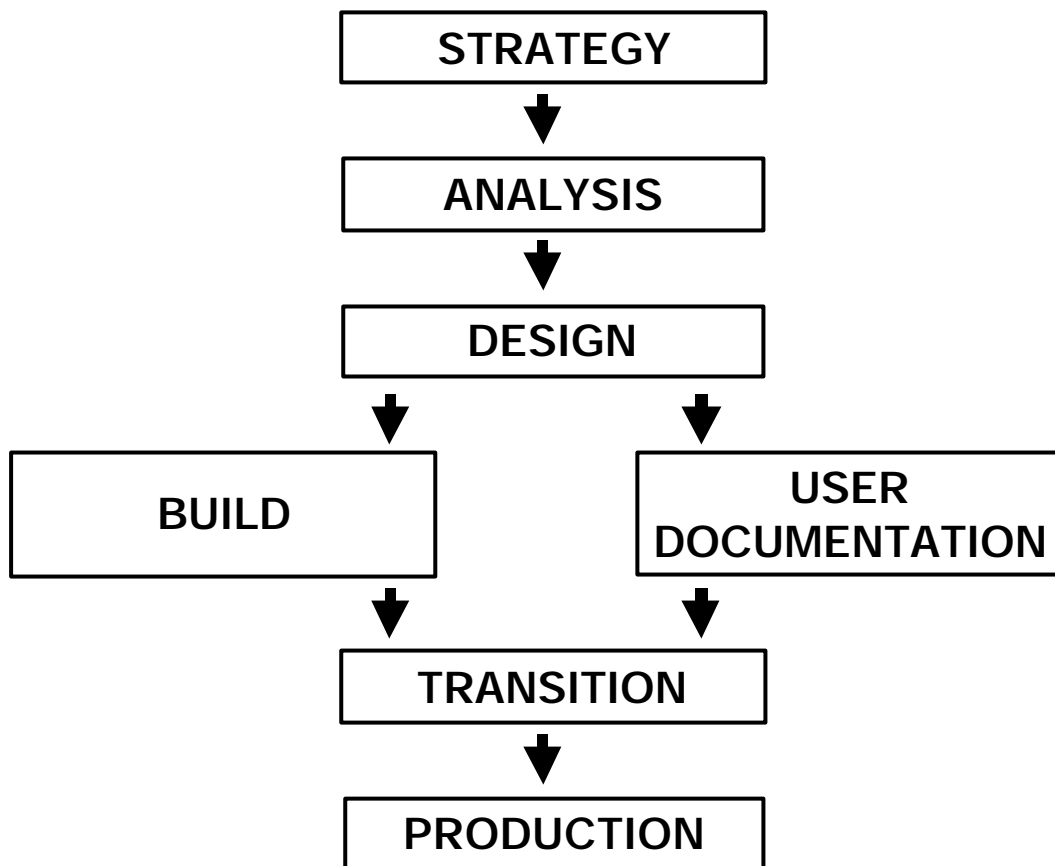


Figure 2

Why use a development life-cycle? There are a number of advantages to using a development life-cycle. They will counteract the tendency to rush into coding, something we all tend to do at the beginning unless there is a strong disciplined culture. A life-cycle give a natural structure to the project and therefore helps with the timing of the phases. On the larger projects the use of the life-cycle will help to structure the work for specialist teams. Therefore using a life-cycle model having identified the time and manpower for each phase, this helps to provide feedback on progress.

Notwithstanding, the use of a development life-cycle will not guarantee success, it is NOT a magic wand. It does not allow for making project staff interchangeable, i.e. programmers cannot necessarily do the work of the analyst and the analyst cannot necessarily do the work of the designer, etc. In this way it does not de-skill the software development process.

The main life-cycles discussed are the NCC cyclic model (figures 3 & 4), see handout on 'The System Life Cycle'. These two show similar phases but are viewed from the management viewpoint and the analyst viewpoint.

The System Life-Cycle: Project Stages

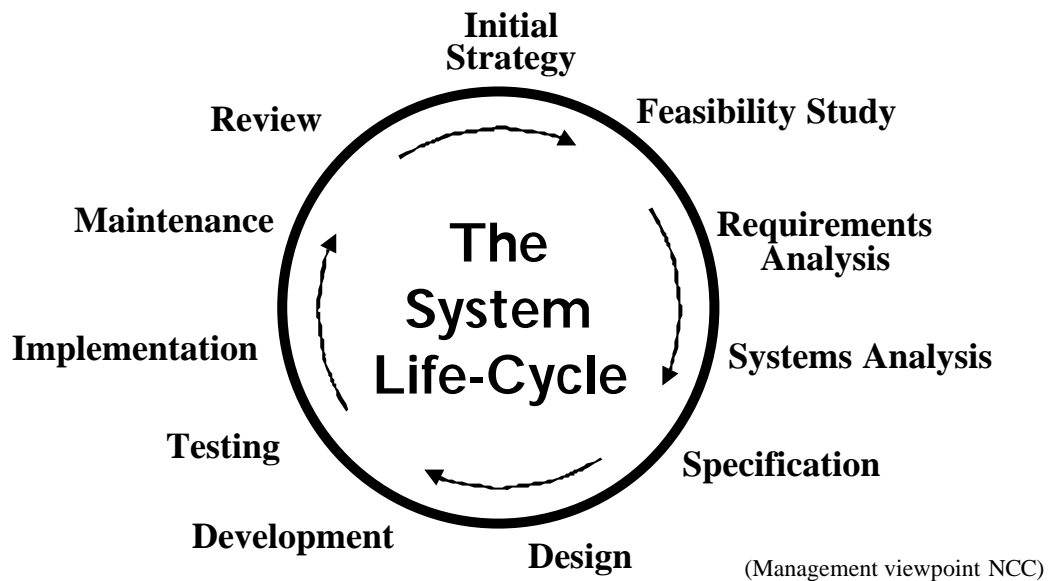


Figure 3

SLC-Systems Analyst's view point

(as given by NCC)

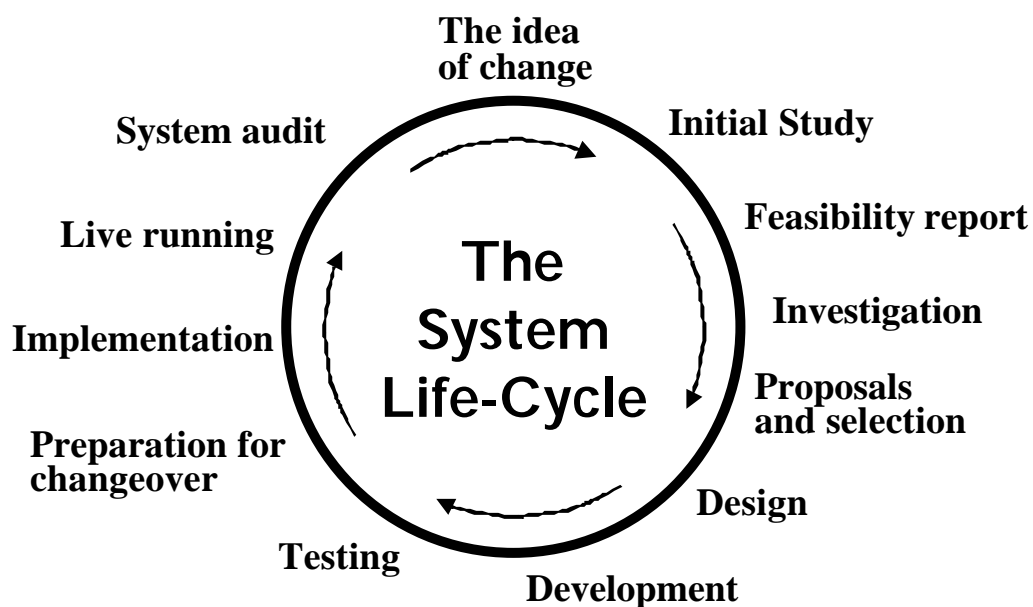


Figure 4

A further adaption of the cyclic model is that of the spiral which firmly places the idea of

iteration. The basic spiral model shows that each task or phase is divided into four stages:

1. Taking Stock (Stage 1)

Establish objectives, these may include performance, functionality, and ease of change.

Identify constraints including cost, deadlines, and interface with other s/w components.

Identify alternative solutions for example the development may be by purchasing an off-the-shelf system, or by developing in-house software, or by engaging an outsourcing organisation to manage and develop the system.

2. Dealing with risk (Stage 2)

In this stage the analyst must evaluate alternatives identified in both system options and the risks involved. They need to identify risk areas and decide how to overcome the risk. This involves carrying out a risk reduction analysis.

3. Develop then verify the next product (or stage or output)

During this stage of the life-cycle the analyst will carry out the task identified in the previous phase or stage and deliver the required output.

4. Planning for next phase (Stage 4)

This stage involves the planning of the next phase or task.

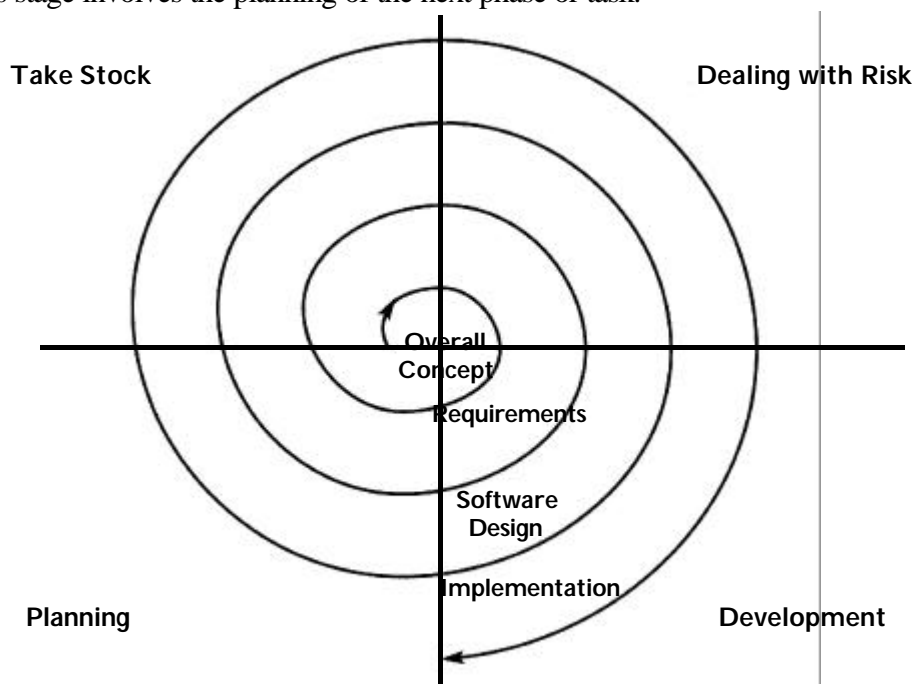


Figure 5

The Waterfall

The “traditional” waterfall is more suited to the software engineering discipline, although even this has variations as depicted by Booch in his book “Object-Oriented Design with Applications” as in figure 6 and Royce’s version as shown in figure 7.

A “traditional” waterfall

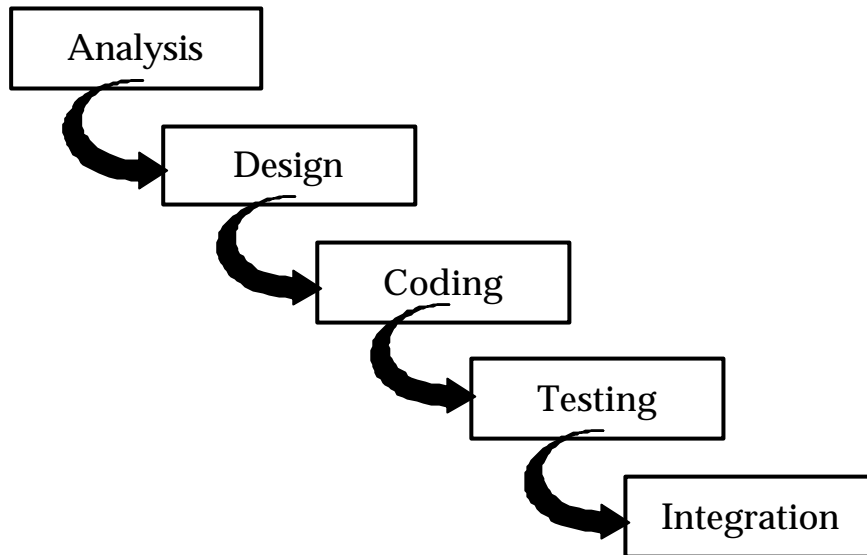
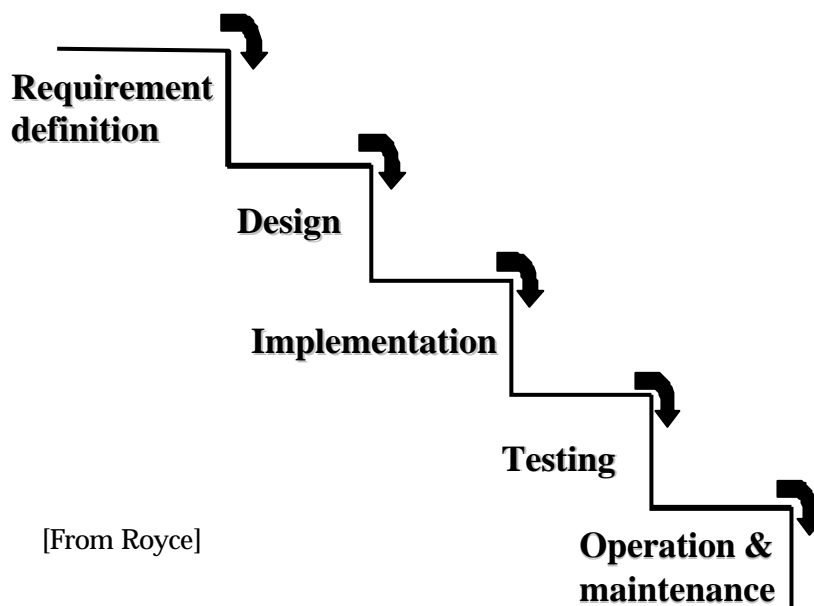


Figure 6

The Waterfall Model



[From Royce]

Figure 7

Whilst the waterfall model tends to imply a sequence without any iteration this can be include by adding feedback loops at each stage as shown in Budgen’s waterfall model in

figure 8.

Budgen's Waterfall

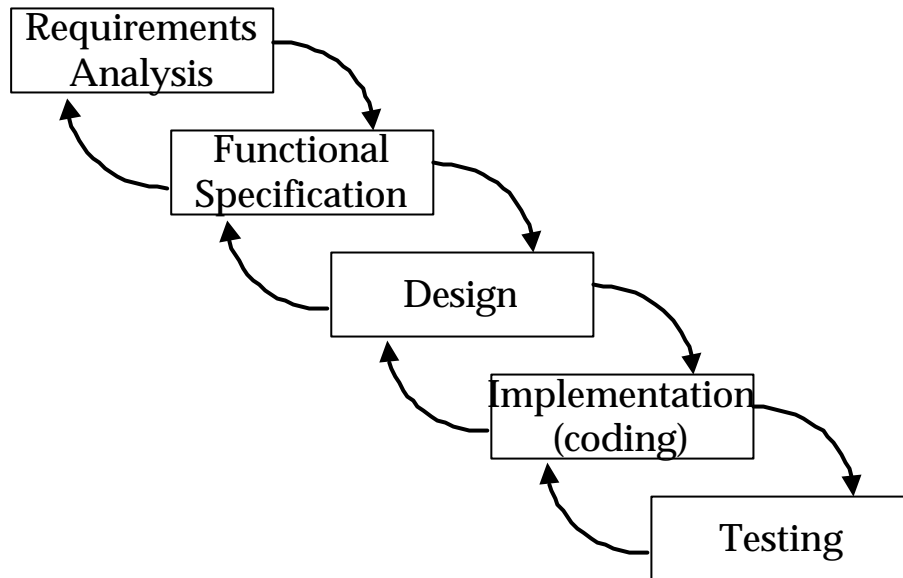
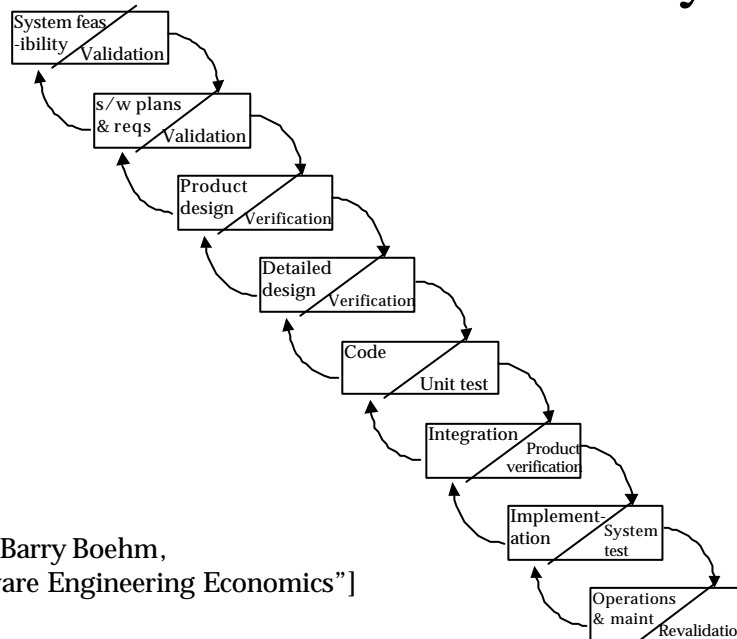


Figure 8

Boehm takes this concept further in his version of the waterfall by adding the concept of verification of each stage by the following stage. This model is shown in figure 9.

Boehm's Waterfall life-cycle



[From Barry Boehm,
"Software Engineering Economics"]

Figure 9

The V Model

The idea of verification of stages of the system life-cycle is better portrayed in the model used in software engineering and known as the 'V' model, shown in figure 10. Each stage of the development cycle has a related outcome or document and this output can be tested against the output of a later stage. The left vertical of the 'V' shows the analysis and design stages, these being verified against the development stages on the right hand side.

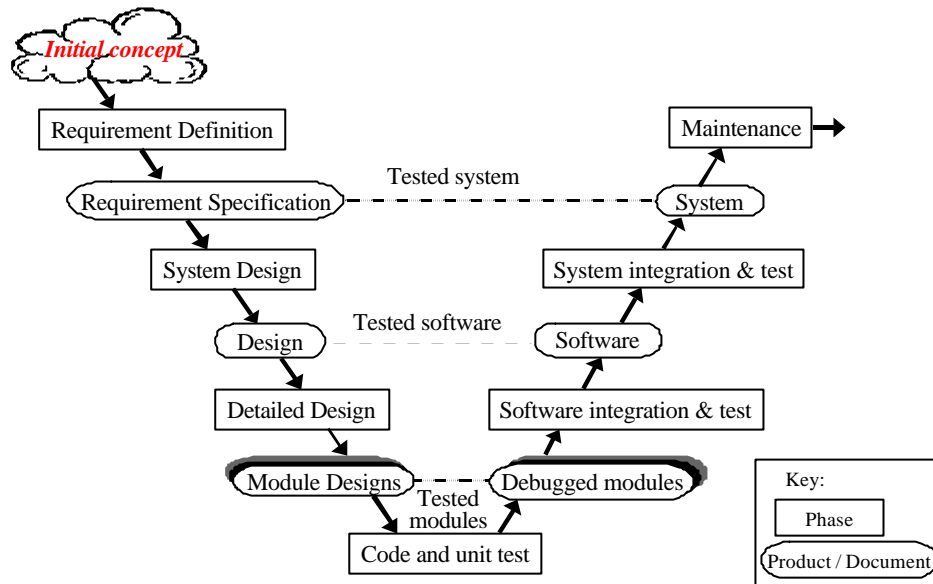


Figure 10

Prototyping

Another very common life-cycle is that of prototyping as shown in figure 11. Again this is a method 'borrowed' from engineering where a 'model' of a required system is built often with heavy involvement of the user.

There are three different ways of using the prototyping life-cycle:

- Evolutionary** - the prototype grows into the final software. This produces a quality system from an outline specification.
- Revolutionary** - the prototype is thrown away. A prototype built to identify initial problems and after experimentation an improved specification is formulated. Hence can be used when requirements are unclear.

The outcome from this type of prototyping would be the system specification from which the full system could be developed. It tends to offer quick and dirty solutions.

- Exploratory** - this option can be viewed as being "research-oriented" or more frequently referred to as "hacking".

There are a number of advantages and disadvantages to prototyping for example:

Advantages:

- * copes with lack of clarity in requirements;
- * gives user chance to change their mind before committing to final system;
- * systems can be developed faster;
- * development effort can be reduced as resultant system is correct;
- * maintenance effort reduced as system meets users needs;
- * end-user involvement;
- * user-developer communication improved;
- * users sees progress;
- * increased chance of a user friendly system;
- * allows system to be piloted.

Disadvantages:

- * system developed in artificial environment;
- * can increase user expectation due to speed of development;
- * may have inconsistencies between prototype and final system;
- * user may never be satisfied as given too much opportunity for change;
- * can cause incomplete analysis - instant design;
- * iteration not always accepted due to discarding of work done;
- * non-functional requirements missed as prototyping focuses on functionality;
- * planning and control difficult;
- * configuration management unsuitable.

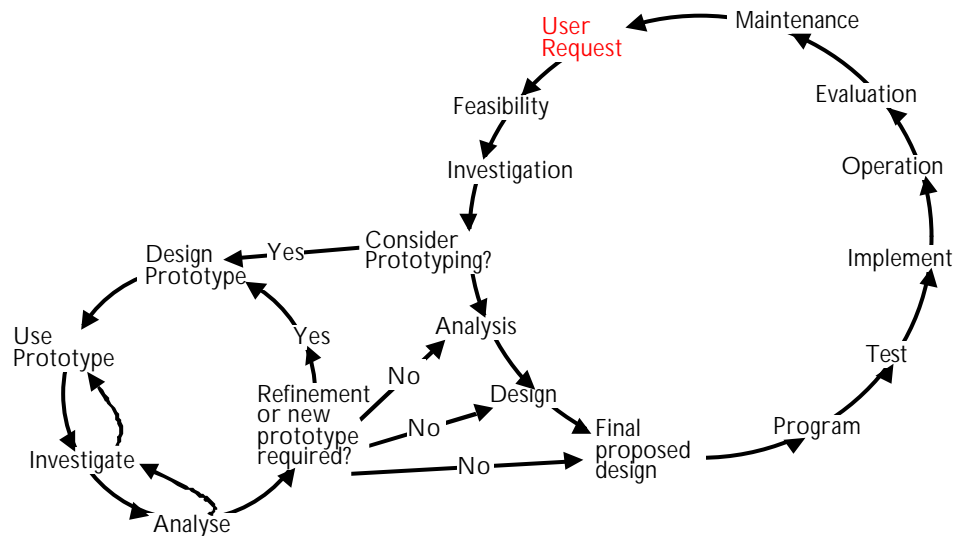


Figure 11

Other Life-Cycle Models

There are plenty of other Life-cycle models that can be explored including:

- Soft System Modelling
- RAD - Rapid Application Development
- JAD - Joint Application Development
- FAST - Facilitated Application Specification Technique
- Business Process Re-engineering.