



# 1. Historical Development of SSDMs

## 1.1. In Days of Yore

The development of software system design methods has been something of a melting pot. The earliest programmable computers were used by mathematicians and physicists for solving very difficult problems (e.g. Bletchley Park which cracked the Enigma code and the Manhattan Project which developed the atom bomb). Later on other sciences like chemistry and astronomy would make use of the new digital computers. Engineers would use computers for solving design problems. Out of this emerged the so-called discipline of computer science which found its own uses for the ever larger machines.



Figure 1

As computers became more powerful and more groups were able to use them, so more and more people developed software to run on the machines. The development of the transistor allowed computer memories to grow rapidly; consequently installed software also grew in size and complexity. By the time

## Notes

large commercial operations such as banks and insurance companies set up their early data processing departments the size of software programs had grown so much that the new third generation languages (3GL) were developed to help manage the growing source code size.

Out of all this emerged different styles and practices for designing and writing computer programs. Naturally, some approaches were more methodical and rigorous than others. With the increasing problems of software development and maintenance in the 1960s and '70s (eloquently described by Frederick Brooks<sup>1</sup>) came an acceptance that some "structured" approach was needed. The early '70s saw the emergence of so-called "structured programming" which Jackson's celebrated JSP method<sup>2</sup> coming along in 1975.

We are now in the position of being able to choose from a range of different methods each designed to meet specific needs within the software development industries.

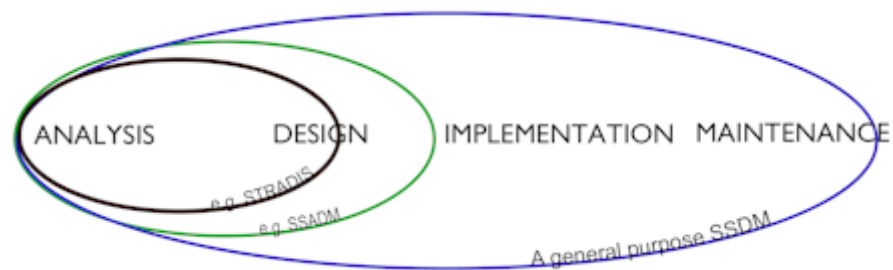


Figure 2 The SSDM Domain

A good general purpose design method should aim to address all stages of the system development lifecycle. As we shall see, not all methods attempt to address every aspect.

## 1.2. Early methods

The development of methods of systems design has followed a path which has been much criticised, wrongly in our view. Methods have evolved which have allowed the building of systems of considerable complexity and utility. The major problem is that user requirements for change have occurred at an alarming rate and hardware technology has progressed at a rate which has exceeded many expectations and has in fact fuelled many of the requests for change. Add to this the growing realisation that change is necessary for rapid business growth and is at least easier with computer technology, and it is readily seen what has put computer systems to the test. The result is that many methods have not facilitated change, at least not at the rate which is expected of them.

The very earliest methods assumed a fairly simplistic life cycle and a fairly common pattern of:

- Data capture, followed by
- Data validation, followed by
- Ordering of valid data, followed by
- Updating a master file, followed by

<sup>1</sup> Brooks, F.P. *The Mythical Man Month* (Anniversary Edition), Addison-Wesley, 1995.

<sup>2</sup> Jackson, M.A., *Principles of Program Design*, Academic Press, 1975.

- Extraction of required information from the master file.

Later systems merely combined one or more of the above processes. The problems of design within such a pattern are:

- What data are required to be kept in the master file(s)? (the trick here was to anticipate the information requirements);
- How, when and where to capture and check the basic systems data?;
- How many processes are required in maintaining the systems data and producing the necessary business information?;
- Matching hardware and systems software to the anticipated application software.

The problems were not normally very difficult to solve in a static situation, but become enormous in a highly dynamic business environment. Why?

- Because new data requirements tended to necessitate a very large number of systems changes most particularly in the software;
- New processes tended to affect existing ones often in unpredictable ways;
- The business environment has a much more complex structure than the simple notion of collect data and transform it into information.

### 1.3. Evolution

Very rapidly systems analysts/designers started to use a variety of diagrammatic aids to describe existing and new systems (not that this was new in itself in the business world, Organisation and Methods analysts have used a number of different charting methods for many years). These, in the main, both clarified the specification of systems and their implementation when they were up-to-date. Modern CASE (Computer Aided Software Engineering) tools using quite sophisticated graphics techniques aim to eliminate this problem.

While tools were being developed (comparatively slowly), the software design aspects of computer systems concentrated at first on the notions of modularity, top-down development and step-wise refinement culminating in the almost universally accepted if ill-defined structured programming which was then extended into various forms of structured systems analysis and design. Although many of the methods overlap and terminology is by no means universally accepted we can at least recognise three basic categories of method worthy of study:

- Data flow-orientated design;
- Data structure-orientated design, and
- Object-orientated design.

Of these, you will be most familiar with the first two. We shall be reviewing these and developing a deeper understanding of one of the most specific methods in the second category (JSD). It is possible to view JSD as having a largely object orientated flavour, and we shall attempt to bring this out.

### 1.4. Methods/Methodologies

One definition of a methodology is *a collection of procedures, techniques, tools and documentation aids which will help the systems developers in their efforts to implement a new system* .

The objectives of methodologies often differ greatly. The following list gives six reasonable objectives.

1. **To record accurately the requirements for an information system.** The users must be able to specify their requirements in a way which both they and the systems developers will understand, otherwise the resultant information system will not meet the needs of the users.
2. **To provide a systematic method of development in such a way that progress can be effectively monitored.** Controlling large scale projects is not easy, and a project which does not meet its deadlines can have serious cost implications for the organisation. The provision of checkpoints and well defined stages in a methodology should ensure that project planning techniques can be effectively applied.
3. **To provide an information system within an appropriate time limit and at an acceptable cost.** Unless the time spent using some of the techniques included in some methodologies is limited, it is possible to devote an enormous amount of largely unproductive time attempting to achieve perfection. A methodology reflects pragmatic considerations.
4. **To produce a system which is well documented and easy to maintain.** The need for future modifications to the information system is inevitable as a result of changes taking place in the organisation. These modifications should be made with the least effect on the rest of the system. This requires good documentation.
5. **To provide an indication of any changes which need to be made as early as possible in the development process.** As an information system progresses from analysis through design to implementation, the costs associated with making changes increases. Therefore the earlier the changes are effected, the better.
6. **To provide a system which is liked by those people affected by that system.** The people affected by the information system may include clients, managers, auditors, and users. If a system is liked by them, it is more likely that the system will be used and be successful.

## 1.5. Stages in SSDM development

Just as we can trace the development of programming languages from first generation languages through to today's fourth-generation environments, so we can map a generational development of systems methods.

### 1.5.1. First generation methods

In the first-generation methods of the 1960s developers tended to rely on a single technique and modelling tool, although various techniques and tools existed for varied sets of problems (see the melting pot earlier). These early methods made use of the so-called structured techniques associated with program design. These centred around functional decomposition as a way of reducing complexity, a form of *divide et impera*. Structured design became ingrained and almost innate and spawned the development of techniques such as data-flow diagramming.

Despite rapid technological changes, there is much inertia in the software industry and many organisations did not adopt any form of structured analysis and design until the 1980s.

### 1.5.2. Second generation methods

If the first generation models were data- or process-oriented, the more mature second generation models placed much more emphasis on the construction and checking of models. The aim was to provide a smoother path from initial requirements gathering and specification through to design and implementation. So, we see life-cycle models as being integrated much more into the design methods. The models used were seen sequential with each individual model addressing different stages in the life cycle. Thus, the first model constructed would aim to capture system requirements in policy terms (i.e., what must the system do). This description would then be elaborated on and refined in later stages to show how these requirements could be realised using available technology. The logical/physical model approach of SSADM is a good example of this.

Where first generation methods tended to model a system from a single viewpoint, e.g. looking only at data or only at process, second generation methods recognised that both data *and* function are equally important aspects of a single system.

### 1.5.3. Third generation methods

In the second generation of methods, although models are used, the view of them is still rather low level. That is, a 2G method tends to deal with individual, discrete diagrams. Issues about how analysis and design units fit together and interact tended to be ignored.

In 1983, with the introduction of Jackson System Development, we see the emergence of the third generation of system design methods. In JSD we have a more holistic approach to system design. The second generation methods, although taking a multi-viewpoint model-based approach still compartmentalised and dealt with individual diagrams and models. In the third generation we see more concern with the system as a whole (from policy statement right down to implementation) rather than with its different parts.

We see third generation methods attempting to focus on the 'real world' of the system; much attention is given to the essential policy and purpose of the required system. Any models constructed support the transition from problem statement through to implementation without losing sight of this high-level view.

## 1.6. The Conventional (NCC) Approach

The conventional approach advocated most lucidly by the NCC has been described in detail elsewhere. Essentially it consists of the so called waterfall model life-cycle of:

- Feasibility study;
- System investigation;
- Systems analysis;
- Systems design;
- Implementation;
- Review and maintenance;

and was characterised by attempts to produce technical and user documentation using standard forms and check lists. The underlying document categories are as shown in Table 1 and illustrated by the Poly Fleet Hire Case Study example

# Software Systems Planning & Design

Notes

documents appended.

1	Background	Terms of reference.
2	Communications	Discussion records, correspondence, manuals etc.
3	Procedures	System outline, run charts, flow charts etc.
4	Data	Document specs., file specs., record layouts etc.
5	Supporting Information	Grid charts, organisation charts, data item definition, hardware & software facilities.
6	Testing	Test dat spec., test plans, test operations, test logs.
7	Costs	Development & operation cost information.
8	Performance	Estimates or reports of timings, volumes, growth etc.
9	Documentation Control	Copy control, amendments incorporated list, outstanding amendments.

Table 1: NCC Document types

The major criticisms of this approach were:

- **Failure to meet the needs of management:** Although systems developed by this approach often successfully deal with such operational processing as payroll and the various accounting routines, middle management and top management have been largely ignored by computer data processing. There is a growing awareness by managers that computers ought to be helping the organisation to meet its corporate objectives.
- **Unambitious systems design:** Producing a computer system that mirrors the current manual system is bound to lead to unambitious systems which may not be as beneficial as more radical systems.
- **Models of processes are unstable:** The conventional methodology attempts to improve the way that the processes in businesses are carried out. However, businesses do change, and processes need to change frequently to adapt to new circumstances in the business environment. Because computer systems model processes, they have to be modified or rewritten frequently. It could be said therefore that computer systems, which are models of processes, are unstable because the real world processes themselves are unstable.
- **Output driven design leads to inflexibility:** The outputs that the system is meant to produce are usually decided very early in the development process. Design is output driven in that once the output is agreed, the inputs are decided and the processes to convert input to output can be designed. However, changes to required outputs are frequent and because the system has been designed from the outputs backwards, changes in outputs usually necessitate a very large change to the system design.
- **User dissatisfaction:** Sometimes systems are rejected as soon as they are implemented, often because the user requires more flexibility than the computer system has been designed to give. Users have found it difficult to understand technical matters which are often given far more attention than the underlying business problems.
- **Problems with documentation:** Although claimed to be one of the major benefits of the NCC approach, it has been criticised for being too technical, too easy to leave until too late, too easy to forget altogether and too easy to forget to update.
- **Incomplete systems:** Exceptions are frequently ignored because they are too expensive, and often not diagnosed or simply forgotten.
- **Application backlog:** Some users literally have to wait for years for a system to be implemented. Others simply do not bother asking.
- **Maintenance workload:** Keeping operational systems going whether they have been designed well or badly will nearly always take first place sometimes leading to patches upon patches. The maintenance workload is in most cases an increasing one.

## 1.7. Exercises

1. From your own practical experience, describe the disadvantages of the conventional approach to systems development.
2. From the point of view of management, what are the major drawbacks of the conventional approach?
3. What major benefits can be attributed to the N.C.C. standards approach to systems development?

## Software Systems Planning & Design

### Notes

4. In your opinion, are the criticisms given above fair? Explain.
5. Using the Poly Fleet Hire documents identify the car registration number field and comment on the implications of an increase in the length of this field.
6. To what extent are the criticisms of this method dependent upon its being a manual documentation approach and, hence, would be ameliorated in a machine implementation?