The Analysis and Design of an Automated Tool to Support Structured Systems Analysis

A thesis presented in partial fulfilment of the
requirements for
the degree of Master of Arts in Computer Science at
Massey University

Joanne Tucker
1988

Volume I

Abstract

Systems analysis is an inherently difficult task. Errors that are introduced in the analysis and design phases become progressively more expensive to fix in the later stages of the system life cycle. Systems analysis and design methodologies attempt to reduce the number of errors introduced into a system model and to detect (and correct) those errors that do occur as early as possible in the system development lifecycle. One such methodology that is widely used in New Zealand is Structured Systems Analysis. Users of Structured Systems Analysis tend to find that the documentation produced using the methodology is easier to read and understand than documentation produced by other currently used methodologies.

This thesis presents the functional specification of MUSSAT, a tool to provide automated support for the Structured Systems Analysis methodology. MUSSAT was designed for a specific group of users. The needs of these users are discussed, together with an introduction to the tools and techniques of Structured Systems Analysis. Existing versions of Structured Systems Analysis are reviewed and a modified form of the methodology, incorporated in MUSSAT, is presented.

A discussion of the tools and techniques used to specify the MUSSAT model are discussed. This is followed by an introduction to the MUSSAT system model. Details of the MUSSAT model are included as a series of technical appendices.

Finally, an overview of the extent to which Structured Systems Analysis is supported by existing Computer Aided Software Engineering (CASE) tools is presented together with a discussion of where MUSSAT fits with these CASE tools.

Acknowledgments

During the past two years there have been many people who have influenced the course and content of the research presented in this thesis. I am grateful to them all for their help.

In particular, I would like to thank my supervisor, June Verner, for painstakingly reviewing each part of this thesis and for her invaluable enthusiasm, encouragement and ideas.

I would also like to thank Professor Graham Tate and Richard Hayward for their constructive criticisms of earlier drafts of the first four chapters of this thesis.

Finally, I would like to thank Stephen Quinn and Jason Cruickshanks for their help in using the Department's Vax 11/750, Unix and Emacs and for helping me understand some of the basics of interactive graphics.

Table of Contents

List of Figures

CHAPTER 1

Introduction

The evolution of a computerised information system begins when the need for the system is recognised. If management is willing to commit resources to the development of the system, various design and construction phases will take place transforming the original idea into an implemented system. Once the system is operational continued financial commitment for maintenance will be required, until eventually, the system reaches the end of its productive life and is phased out.

This is, of course, a simplified picture of the development of a computer system. The feasibility of a proposed system will be reviewed periodically during development and if the perceived benefits of the system are outweighed by the perceived costs the project may be abandoned, even after significant expenditure.

The term 'system life cycle' is commonly used to describe the various phases through which a computerised system evolves. For the development of a system to proceed in a controlled fashion certain activities must take place and specific documentation should be produced during each phase of the life cycle. There have been many definitions of the

system life cycle including those in [KAN84], [DAV83], [WEI84], [POW84] and [DEM78]. Most systems development textbooks present a version of the traditional system life cycle, also called the project or software life cycle. Nevertheless, these life cycle models are similar even though the names and boundaries of the phases may differ.

Some system life cycle models, such as that presented by Boehm [BOE81], incorporate techniques such as prototyping, incremental development and advancemanship. However, these alternate life cycle models still include some of the earlier activities of the traditional system life cycle models. In this thesis, discussion of the system life cycle will refer to the traditional life cycle model in which systems analysis precedes systems design.

One life cycle model [POW84] divides system development into the following five phases:

Phase	Successful completion documented by
INVESTIGATION	
Initial Investigation	
	Initial Investigation Report which, among other activities, identifies the major options and recommends one or two of the options for further consideration in the next phase.
	Oral Report to Management.
Feasibility Study	Feasibility Report which includes: a description of the existing system; and rough cost/benefit analysis estimates and project schedules for each alternative.
	Project Plan
	Working Papers Which serve as an input to the next phase and include: a review of the existing and proposed system; and alternative implementations of the proposed system.

Phase	Successful completion documented by
ANALYSIS AND GENERAL DESIGN	
Systems Analysis	User Specification which includes: a model of the proposed system; and specification of performance, security and control requirements etc.
General Design	New System Design Specification which includes: internal software design, file or data base design, hardware specifications, internal controls, etc.
DETAILED DESIGN AND IMPLEMENTATION	A fully tested and documented system.
INSTALLATION	No new products. Activities include file conversion and personnel transitions.
REVIEW	System Development Recap Report and Post Implementation Review Report.

Weiner and Sincovec [WEI84] provide an alternate description of the outputs that should be produced as a result of the Systems Analysis phase. These components comprise a Software Requirements Document and consists of the following ([WEI84] pp. 17-18):

(1) A description of the functional requirements, with an emphasis on describing the system as it will appear to the user, including a description of the user interface.

(2) A description of the non functional requirements of the new system.

(3) A project plan.

(4) Maintenance information.

(5) An initial user manual.

(6) A glossary of all technical terms.

The outputs of one phase of the lifecycle are used as inputs to the next. Therefore, the successful completion of a phase in the development cycle is, in part, dependent on the quality of the outputs of the previous phase: if the outputs of one phase are deficient or contain errors, then the next phase will not have an accurate description of the current state of the system on which to base successive work. If an error is detected in the system model then the outputs of each phase, from the phase in which the error was introduced to the phase in which the error was detected, must be amended to present a more accurate model of the system. In the worst case deficiencies in the Investigation phase will not be detected until acceptance testing when the users discover that the system developed is not the system that they

wanted.

The earlier that deficiencies are detected in the development of a system the less costly they will be to correct. Boehm ([BOE81], p. 40) states that errors in large system development projects not detected until the maintenance phase are likely to be at least 100 times more expensive to correct than if detected during the requirements phase. To compound the problem, another author [CON82] suggests that during the development of a system a large proportion of all errors (that are later detected) are introduced during the analysis and design phases ([CON82] p. 215).

Systems analysis is an inherently difficult task. Analysts are required to produce complete and unambiguous documentation of a system by working with users who are often unsure exactly what they want the system to do and, in some cases, may not even want a new system. The analysis phase needs to be carried out by experienced computer professionals with expertise in both computer systems and the application area. The specification of the new system produced should fulfill all requirements specified by the users and be detailed enough to serve as an input into the Detailed Design and Implementation phase.

Various systems analysis and design methodologies have been specified in an attempt to reduce the number of errors introduced into a system model and to detect (and correct)

those errors that do occur as early as possible in the system development lifecycle. One such methodology is Structured Systems Analysis (SSA). SSA became popular in the late 1970's and is now widely used, particularly in North America and New Zealand.

1.1. Structured Systems Analysis

The SSA methodology incorporates a set of tools and techniques to describe a logical model of an existing or proposed system. The main components of SSA are:

data flow diagrams (DFDs); and an associated
data dictionary (DD).

The logical system model produced during the analysis phase, using SSA, corresponds to the Software System Model component in Weiner and Sincovec's list of systems analysis products given above. Note that the Software System Model is only one component of several required to fully specify a proposed computer system.

There are two major variants of the SSA methodology: the DeMarco [DEM78] and Gane & Sarson [GAN80][1] SSA methodologies. Although both of these versions of SSA are each based

[1]The text reference throughout this thesis is a republication of: Gane, Chris and Sarson, Trish, Structured Systems Analysis: Tools and Techniques (New York: Improved Systems Technologies Inc., 1977).

on the same principles, they differ in:

physical representation of DFDs;

DFD construction rules;

contents of the DD; and

representation of complex file structures.

The discussion in this thesis assumes a familiarity with the basic SSA tools and their application (as described in [DEM78] and [GAN80]), however, a brief overview of SSA is given in Chapter 2.

Neither the Gane & Sarson nor the DeMarco SSA methodology have proven to be the ideal analysis tool for all situations. SSA, in general, has a number of weaknesses, including:

reliance upon the skill of the analyst to produce a good system model;

lack of consistency and completeness checks;

difficulty in modelling interactive systems, and related to this, the inability to model different processing or response constraints such as monthly report runs as opposed to the response times required for interactive query facilities; and

lack of support in translating the new logical model produced during systems analysis into the new physical model required as the output of systems design.

When considering criticism number three in the above list it is possible to argue that using SSA the analyst should not be concerned with, nor have decided the type of interface

that the proposed system should incorporate at the analysis phase. Hence the inability to show any one particular type of user interface in the logical new model should not be a concern

Nevertheless, in some cases the analyst knows even before the analysis phase begins that the system to be produced must be interactive. In such cases it would be naive and counterproductive to ignore the requirements of an interactive user interface, especially when the incorporation of such an interface is likely to have a significant impact on cost and size estimates (made both at the feasibility and later stages) and the system design itself. Size and cost estimates should be as accurate as possible so that the system alternative selected or even the decision to continue system development is made with the best possible information.

Even in view of the above weaknesses of the methodology, SSA has one main advantage over other design methodologies such as ISAC [LUN81], JSD [JAC83] and Information Engineering [MAR81]: namely, that the specification produced using SSA serves as a good communication tool, especially between the analyst(s) and users. Data flow diagrams are conceptually simple and are similar to other diagrams familiar to many users in the business world. Unlike ISAC, a SSA system specification is relatively concise which means that management of the documentation may be simplified and intended

users of the system are not given copious and daunting amounts of documentation to approve.

Data flow diagrams also highlight the transformation of data, that is, the processes within the system. Users are process oriented: they think in terms of objects and the operations performed on objects, such as editing a document or updating a master file, not in terms of entities, attributes and relationships [TAT86]. Information Engineering takes the latter approach. SSA helps users to identify errors and omissions in systems modelled using DFDs since the model more closely represents the way they view the real world.

Individuals and organisations who use SSA are likely to tailor one of the SSA methodologies to suit the peculiarities of their working environments. Such adaptations can be considered to produce hybrid versions of the SSA methodology.

The research documented in this thesis is concerned with the development of an automated SSA tool. Such software is required to provide computerised assistance in developing and maintaining SSA system models.

A tailored form of the SSA methodology will be required for the proposed automated SSA tool. The user groups of an automated SSA tool are expected to have specific needs that are not explicitly catered for in either of the two standard

SSA methodologies, therefore, a hybrid SSA needed to be defined to suit the requirements of the intended users of the system.

Investigation of automated support for SSA was considered a suitable topic for this thesis because:

> Most of the automated SSA tools currently available are expensive;
>
> Automated SSA tools may be able to offer assistance in other areas of system development in addition to systems analysis; and
>
> An automated SSA tool may be able to provide a means of overcoming some of the deficiencies in the SSA methodology.

1.2. Users of Structured Systems Analysis

Although SSA has been in use for over a decade, automated tools supporting the methodology have only recently become commercially available. It is not known how useful systems analysts find such automated tools: whether they are useful aids in developing the logical model of a system, or only useful in documenting and checking a system model developed using pencil and paper.

In either case, even an experienced analyst usually requires several iterations to produce the final version of a DFD. Drawing and updating DFDs manually is tedious and automated tools are expected to, at least, help reduce the time consuming nature of DFD maintenance and development. Time sav-

ings may not be as readily apparent in the initial specification of a DFD, where the time taken to specify a DFD using an automated SSA tool and drawing a DFD by hand may be similar. The real time savings are expected to be most evident in DFD maintenance where, using an automated SSA tool, the user does not need to redraw the entire DFD in order to produce an updated copy.

Post-implementation changes to a system should be reflected in the system documentation. The maintenance of system documentation can be error prone and time consuming and is often poorly done. An automated SSA tool may be a useful aid in maintaining system models developed or simply documented using SSA.

Automated SSA tools may also encourage more creative experimentation with system designs if changes are easy to make and control. Hence, alternative system designs may be able to be produced quicker and at less cost. In addition, alternate system designs may be more complete using an automated SSA tool since less effort may be required to develop these alternate designs. Designs that are more complete or thorough (even at a very high level) may help users to understand their needs and the proposed system better and hence help create a system meeting their requirements as closely as possible.

Existing automated SSA tools may possibly incorporate version(s) of SSA that overcome some of the deficiencies of the original methodology.

An automated SSA tool is required or would be useful in a number of fields in which research is currently being done. Within the Massey University Department of Computer Science an automated SSA tool may be useful in the following areas:

(1) The Department offers some courses that require students to produce SSA system models. An automated SSA tool would be a useful teaching aid for such courses.

(2) Two groups of researchers within the Department require software that allows users to specify SSA system models using a graphical interface.

Many of the existing automated SSA tools are not available in New Zealand, and those that are available are expensive and often unable to be evaluated thoroughly enough, either by hands-on experience or detailed documentation. Without the tools being available for evaluation it is difficult to decide whether they are cost-effective. In addition, unless potential users are able to gain hands-on experience of available tools, it is difficult to specify exactly what constitutes a useful automated SSA tool. Without a thorough analysis of what is required of an automated SSA tool by potential users, short-term exposure to such tools would be an inadequate foundation on which to base any decision about

the usefulness or otherwise of an automated SSA tool.

There are a number of automated SSA tools and aids commercially available.[2]

These include:

Excelerator	(Index Technologies Corp.)
Teamwork/SA	(Cadre Technologies Inc.)
PCSA	(StructSoft Inc.)
DesignAid	(Nastec Corp.)
Anatool	(Abvent)
ProkitAnalyst,	(McDonnell Douglas Automation Co.)
Blues	(De Landgraff Consultancy).

Potential users of an automated SSA tool are expected to require specific features of such software. This thesis identifies the features that one group of users would require in such an automated SSA tool so that the tool will be useful to them. These needs can also be used as criteria to evaluate existing automated SSA tools and in determining whether the ideal automated SSA tool exists. Given the current technology, it may be that an ideal tool at a reasonable price is not commercially available.

---

[2]The names and addresses of the suppliers of the named automated SSA tools are listed in Appendix I.

1.3. Research Goals and Methodology

It was the aim of the research documented in this thesis to design an automated SSA tool that allows systems analysts to develop and maintain system models using SSA. The design goal was broken into the following sub-steps:

(1) Identify the needs of potential users of an automated SSA tool.

(2) Investigate the SSA methodology in light of the requirements of the users.

(3) Present the design of an automated SSA tool that satisfies the needs of the selected user(s) and incorporates an appropriate form of the SSA methodology.

(4) Compare features of commercially available automated SSA tools with the system designed presented in this thesis.

It may have been more appropriate to examine existing automated SSA tools before steps 1 to 3, however, at the time this research was undertaken it was not known which, if any, automated tools would be available for evaluation. Permission was granted by the Reserve Bank of New Zealand (after steps 1 and 2 were under way) to use their copy of Excelerator for evaluation. Later in the year a demonstration version of PCSA was ordered from the U.S.A. but was not received until December, 1986. In addition, the proposed

automated SSA tool was designed before using any existing tools so that the design was not biased by the characteristics (and perhaps limitations) of existing tools. Hence, an evaluation of existing SSA tools was not carried out until steps 1 to 3 were completed.

1.4. Thesis Organisation

The body of this thesis is divided into six chapters and an nine appendices:

Chapter 2 presents an overview of the SSA features common to both the Gane & Sarson and DeMarco versions of the methodology.

Chapter 3 describes the potential user groups of an automated SSA tool and outlines their general needs. A discussion of the reasons for selecting a subset of all identified users is also presented in Chapter 3.

Chapter 4 includes a discussion of the differences between existing SSA methodologies and introduces the characteristics of the hybrid SSA methodology suggested in section 1.1.

A discussion of the major design decisions and techniques used in the development of MUSSAT: Massey University Structured Systems Analysis Tool, is given in Chapter 5. Chapter 5 also includes an overview of the user interface of MUSSAT. Details of the MUSSAT design are included in a series of technical appendices, each of which is introduced in Chapter

5.

Chapter 6 gives an overview of CASE Tools and compares the goals of MUSSAT with the goals of existing CASE tools. The user interfaces of two automated SSA tools are discussed and compared with the user interface of MUSSAT.