Summary of "Associates systems for decision support" by A. P. Sage

A. Christofides

7 March 2000

1 Introduction

Decision Support Systems (DSS) are software systems that support the decision making process by helping decision makers understand the implications of their judgements [Fre00]. Most such software systems are general-purpose and can be used in any application domain, for example medical, environmental, organisational. In his paper, Sage [Sag93] provides an overview of associates systems. By *associates*, we mean decision support systems that are specific to a particular domain. Most associates systems have been developed for command and control, that is, the exercise of authority and direction by a purposely designated commander over assigned forces in the accomplishment of a mission. However, there are a few exceptions, and Sage attempts to provide an overview of a number of diverse associates systems, namely DSS's for software engineers, pilots, mission planners, systems engineers, designers, and operators.

Unfortunately the paper by Sage is somewhat disorganised and occassionally becomes incomprehensible. He begins by an introduction to command and control, explaining the purpose and function of related computer systems; he describes how such systems and humans work together to comprise a larger "command, control, communications and intelligence" system. He then explores some conceptual frameworks for information processing and problem solving, namely the Rasmussen model of judgment and choice, the Klein model, the Dreyfus model, and the Janis and Mann model, and he concludes this first section of the paper with some general comments concerning the need for decision associates. The way all this diverse information is related is not quite clear. However, one of the points made is that command and control decision making, and probably any decision making, involves five activities:

1. Data gathering
2. Processing the data so that it becomes useful information that can be used for situation assesment
3. Evaluation of alternatives
4. Decision
5. Action

The results of the action are then observed, and the procedure is repeated.

The identification of these five activities involves considerable simplification. Even the division into steps is artificial; the steps may overlap or occassionally be performed in parallel. It is thus natural for different investigators

to come up with different models of the procedure of decision making, but most such models essentially tell the same thing.

Sage covers most of the rest of the paper by describing a software engineering tool called the Programmer's Apprentice. It is not understood what exactly the connection of the Apprentice to the foregoing discussion is; it is not even clear why the Apprentice should be considered a DSS. In fact, the original authors [RW90] of the Apprentice do not claim it to be a DSS, and Sage himself admits that "it is difficult to make a precise determination concerning what qualifies as a [DSS] and what does not." Systems like the Apprentice are sometimes called *expert systems*, because they attempt to replace human expert thinking; however, in almost all cases, expert systems are used supportively, the final decisions being made by humans. One of the differences between expert and associates systems could be that an expert system actually suggests decisions, whereas an associates system only provides insight into the implications of possible decisions. According to this, the Apprentice is an expert, not an associates system, but admittedly the difference is subtle and probably vague.

Most of the rest of this summary describes the Apprentice and discusses its relationship to decision making.

2 The Programmer's Apprentice

The development process is divided into four steps:

1. Requirements specification
2. Design
3. Implementation
4. Verification

After delivery, the cycle typically goes over again, though some textbooks consider this iteration to be a fifth step, which they call maintenance. As with decision making, these steps are artificial; they overlap, they are performed in parallel, they are further subdivided, and different authors make slightly different divisions, which, however, all describe essentially the same thing.

The person responsible for this process is the software engineer. The Apprentice's authors, Rich and Waters [RW90], mention that the engineer's productivity is dramatically increased when they are assisted by support staffs, including junior programmers, testers, documenters, and program librarians, since this allows them to concentrate their effort on the most difficult parts of a task, without having to waste time on the routine details. Thus, the main idea of the Apprentice is to provide the engineer with a substitute for a support team.

It is worth considering whether an automatic program generator could be developed, that would produce the software given the requirements. Such a goal is realistic only for specific very restricted domains, and the Apprentice aims to be a tool for any kind of application. Thus, it has been developed with the philosophy of providing assistance to the software engineer rather than replacing him.

In order to assist the engineer, the Apprentice must share some knowledge with him. This knowledge is organised into small pieces of information that are called *clichés*; for example, "information system", "program", and "linear-search" are clichés. About 1500 clichés form the core of the Apprentice's knowledge base, but probably many more are needed to make it effective for a wide-range of application domains. Clichés are encoded using the *plan calculus*, which has been especially developed for the Apprentice.

The Apprentice aims to provide assistance in the first three steps of the development process, and is, accordingly, divided into the Requirements Apprentice, the Design Apprentice, and the Implementation Apprentice. At the time of publication of [RW90], it seems that considerable progress has been achieved only for the implementation apprentice, which has been given the name KBEmacs. Accordingly, Rich and Waters go into more detail with KBEmacs, the rest of their exposition being theoretical. From their discussion, KBEmacs seems like a high-level language compiler, that translates the very high-level instructions of the user into a programming language like LISP or Ada. One difference from a compiler is that the resulting program is a suggestion only, and the user is able to modify it.

3 Is it a DSS?

Today we would say that the Apprentice is a Computer-Aided Software Engineering (CASE) tool. Rich and Waters [RW90] also seem to think that, but they classify it as an *intelligent* CASE tool. Whether this is the case or not, it remains to be seen whether the Apprentice can be thought of as a DSS.

First of all, it is very difficult, if at all possible, to give a precise definition of a DSS. For example, a CASE tool that provides drawing facilities enables the software designer visualise better the design he has in mind, and the resulting drawing helps him understand the implications of his judgements and accordingly improve his design decisions. Thus, by French's definition of DSS's, which was stated in the beginning of this summary, such a simple CASE tool could be considered a DSS, which it definitely isn't. Consequently, whether a piece of software can be considered a DSS is a matter of judgement.

Sage probably does not classify the Apprentice as an expert system, because it does not replace the engineer. However, hardly any systems exist that have replaced or even substituted human experts; virtually all such systems function supportively.

From a practical point of view, there is not much point in this discussion. Today, 10 years after the publication of [RW90], it seems that the project has almost been abandoned. There are very few references on the web, and none suggests that any progress has been made since 1990; and even then, it was in experimental stage.

4 Other associates systems

The other systems explored by Sage are also in experimental stage. The pilot's associate is software that enables the pilot to monitor system status, assess the situation and plan his mission. Mission planning is of interest to other areas as well, and relevant associates systems provide support in analysis and selection for weapons, resources and targets, consideration of environmental factors, distance, time, and fuel calculations, information exchanges with other units, and others. Program manager's associates would provide support for configuration management, cost analysis, risk management, quality assurance and others.

References

[Fre00] S. French, *Decision analysis and decision support systems*, 3rd draft edition, 2000.

[RW90] C. Rich and R. C. Waters, *The Programmer's Apprentice*, ACM Press, 1990.

[Sag93] A. P. Sage, "Associates systems for decision support", *Information and decision technologies*, 19:165–184, 1993.