Context in Decision Support Systems Development

Alexandre Gachet
University of Hawaii at Manoa, USA

Ralph Sprague
University of Hawaii at Manoa, USA

INTRODUCTION

Finding appropriate decision support systems (DSS) development processes and methodologies is a topic that has kept researchers in the decision support community busy for the past three decades at least. Inspired by Gibson and Nolan's curve (Gibson & Nolan 1974; Nolan, 1979), it is fair to contend that the field of DSS development is reaching the end of its expansion (or contagion) stage, which is characterized by the proliferation of processes and methodologies in all areas of decision support. Studies on DSS development conducted during the last 15 years (e.g., Arinze, 1991; Saxena, 1992) have identified more than 30 different approaches to the design and construction of decision support methods and systems (Marakas, 2003). Interestingly enough, none of these approaches predominate and the various DSS development processes usually remain very distinct and project-specific. This situation can be interpreted as a sign that the field of DSS development should soon enter in its formalization (or control) stage. Therefore, we propose a unifying perspective of DSS development based on the notion of context.

In this article, we argue that the context of the target DSS (whether organizational, technological, or developmental) is not properly considered in the literature on DSS development. Researchers propose processes (e.g., Courbon, Drageof, & Tomasi, 1979; Stabell 1983), methodologies (e.g., Blanning, 1979; Martin, 1982; Saxena, 1991; Sprague & Carlson, 1982), cycles (e.g., Keen & Scott Morton, 1978; Sage, 1991), guidelines (e.g., for end-user computer), and frameworks, but often fail to explicitly describe the context in which the solution can be applied.

BACKGROUND

A DSS is broadly considered as "a computer-based system that aids the process of decision making" (Finlay, 1994). Sprague uses a definition that indicates key components of the DSS architecture. A DSS is a "computer-based system which helps decision makers confront ill-structured problems through direct interaction with data and analysis models" (Sprague, 1980). In a more detailed way, Turban (1995) defines it as "an interactive, flexible, and adaptable computer-based information system, especially developed for supporting the solution of a non-structured management problem for improved decision making. It utilizes data, provides an easy-to-use interface, and allows for the decision maker's own insights." This second definition gives a better idea of the underlying architecture of a DSS. Even though different authors identify different components in a DSS, academics and practitioners have come up with a generalized architecture made of six distinct parts: (a) the data management system, (b) the model management system, (c) the knowledge engine, (d) the user interface, (e) the DSS architecture and network, and (f) the user(s) (Marakas, 2003;Power, 2002).

One section this article, Key Terms, briefly defines nine DSS development methodologies popular in the DSS literature. A typical methodology is represented by the steps in Table 1.

Table 1. Phases of the DSS design and development life cycle (Sage, 1991)

1. Identify requirements specifications
2. Preliminary conceptual design
3. Logical design and architectural specifications
4. Detailed design and testing
5. Operational implementation
6. Evaluation and modification
7. Operational deployment

The exact number of steps can vary depending on the aggregation level of each phase. Moreover, steps are usually sequenced in an iterative manner, which means the process can iterate to an earlier phase if the results of the current phase are not satisfactory. Even though these processes are useful from a high-level perspective, we argue that they poorly support the DSS designers and builders to cope with contextual issues. The next paragraphs provide a couple of examples to illustrate this argument. The first example is related to the user interface. The DSS community widely recognizes that the user interface is a critical component of a DSS and that it should be designed and implemented with particular care. But how critical is this component? On the one hand, if we consider a DSS that is intended to be used by a wide range of nontechnical users (for example, a medical DSS for the triage of incoming patients in an emergency room that will be used by nurses and MDs working under pressure), then the user interface is indeed the single most critical component of the DSS, at least from a usability/acceptability point of view. In this context, the human-computer interaction (HCI) literature tells us that usability must definitely be considered before prototyping takes place, because the earlier critical design flaws are detected, the more likely they can be corrected (Holzinger, 2005). There are techniques (such as usability context analysis) intended to facilitate such early focus and commitment (Thomas & Bevan, 1996). On the other hand, if we consider a highly specific DSS that will be handled by a few power-users with a high level of computer literacy (sometimes the DSS builders themselves), then the user interface is less critical and usability considerations can be postponed until a later stage of the development process without threatening the acceptability of the system. This kind of decision has an impact on the entire development process but is rarely considered explicitly in the literature.

The second example deals with the expected lifetime of the DSS. On the one hand, some DSS are complex organizational systems connected to a dense network of transaction information systems. Their knowledge bases accumulate large quantities of models, rules, documents, and data over the years, sometimes over a few decades. They require important financial investments and are expected to have a long lifetime. For a computer-based system, a long lifetime inevitably implies maintenance and legacy issues. The legacy information systems (LIS) literature offers several approaches to deal with these issues, such as the big bang approach (Bateman & Murphy, 1994), the wrapping approach (Comella-Dorda, Wallnau, Seacord, & Roberts, 2000), the chicken little approach (Brodie & Stonebraker, 1995), the butterfly approach (Wu et al., 1997), and the iterative re-engineering approach (Bianchi, Caivano, Marengo, & Vissagio, 2003). Some authors also provide methods fostering the clear separation between the system part and the knowledge base part, in order to maximize reusability (Gachet & Haettenschwiler, 2005). On the other hand, some DSS are smaller systems used to deal with very specific—and sometimes unique—problems, that do not go past the prototyping stage, that require minimal finances, and use a time-limited knowledge base. Maintenance and legacy issues are less salient for these systems and their development follows a different process.

We describe in the coming sections of this article a unifying approach to DSS development allowing DSS designers to explicitly take these contextual aspects into considerations in order to guide the development process of a DSS. This new approach is based on the concept of value-based software engineering.

VALUE-BASED SOFTWARE ENGINEERING

Suggesting that the DSS community never considered the context of a DSS prior to its development would be unfair. Several authors acknowledge that a systems design process must be specifically related to the operational environment for which the final system is intended (Sage, 1991; Wallace et al., 1987). For example, Sprague and Carlson (1982) explicitly specified in their "DSS action plan" a phase consisting of steps to develop the DSS environment. The purpose of this phase is to "form the DSS group, articulate its mission, and define its relationships with other organizational units. Establish a minimal set of tools and data and operationalize them." (p. 68). Nevertheless, how these tasks should be carried out is not specified. In this section, we propose an approach allowing DSS designers to model contextual value propositions and perform feedback control of a DSS project. This approach is inspired by the concept of value-based software engineering (Boehm & Guo Huang, 2003).

Two frequently used techniques in value-based software engineering are the benefits realization approach