A Context-Based Approach to the Development of Decision Support Systems

Alexandre Gachet[1], Ralph Sprague

Department of IT Management
University of Hawaii at Manoa
2404 Maile Way
Honolulu HI 96822, USA
gachet@acm.org
sprague@hawaii.edu

Abstract. After more than three decades of research, the Decision Support Systems (DSS) community is still looking for a unified and standardized DSS development methodology. In this paper, we argue that the existing solutions remain too distinct and project-specific because they fail to properly consider the context of the target DSS in their processes.
This paper describes a new approach, whose goal is to formalize the notion of context in the study of DSS development. The proposed approach is based on the concept of value-based software engineering. It is structured around two techniques called the benefits-realization approach and the realization feedback process. An example is provided to illustrate how an impractical, context-free DSS development life cycle can be turned into a context-based life cycle focusing on the most success-critical aspects of the target DSS.

1 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. Studies on DSS development conducted during the last fifteen years (e.g., Arinze 1991; Saxena 1992) have identified more than thirty 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.

In this paper, 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 et al. 1979, Stabell 1983), methodologies (e.g., Blanning 1979, Martin 1982, Sprague and Carlson 1982, Saxena 1991), cycles (e.g., Keen and Scott Morton 1978, Sage 1991), guidelines

[1] Research partly supported by Swiss NSF grant Nr. PBFR2-104340.

(e.g., for end-user computer), and frameworks, but often fail to explicitly describe the context in which the solution can be applied. Experience shows that the development process of a large strategic DSS for a multinational company, spanning many organizational units and getting connected to many transaction information systems, is much different from the development process of a small-scale logistics DSS for a local company.

We propose in this paper a new approach, whose goal is to formalize the notion of context in the study of DSS development. We believe that this approach can help the actors involved in the process of building a DSS to find the methodology best adapted to the context of the target system.

2 Context-Based DSS Development

A DSS is broadly considered as "a computer-based system that aids the process of decision making" (Finlay 1994). In a more precise 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) (Power 2002; Marakas 2003).

Many DSS development processes try to guide the development of the DSS with a sequence of steps resembling Table 1 (Sage 1991).

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 non-technical 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 and 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[2]. 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 and Murphy 1994), the wrapping approach (Comella-Dorda et al. 2000), the chicken little approach (Brodie and Stonebraker 1995), the butterfly approach (Wu et al. 1997), or the iterative re-engineering approach (Bianchi et al. 2003). Some authors also provide methods fostering the clear separation between the system part (called container) and the knowledge base part (called contents), in order to maximize reusability (Gachet and 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 which is not expected to have a long lifetime. 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 paper a new approach 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.

[2] One author worked on the development of a DSS for crisis management in the food supply sector, whose underlying models have been nurtured for more than twenty years.

3. 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 (Wallace et al. 1987; Sage 1991). 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 and Guo Huang 2003).

Two frequently used techniques in value-based software engineering are the benefits realization approach and the value-realization feedback process. The *benefits realization approach* (Thorp 2003) allows developers to determine and reconcile the value propositions of the project's success-critical stakeholders. The centerpiece of this approach is the results chain (Figure 1). This chain establishes a framework linking initiatives that consume resources, such as implementing a new DSS, to contributions (describing the effects of the delivered system on existing operations) and outcomes (which can lead either to further contributions or to added value, such as increased profit). A results chain links to assumptions, which condition the realization of outcomes. Once the stakeholders agree on the initiatives of the final results chain, "they can elaborate them into project plans, requirements, architectures, budgets, and schedules." (Boehm and Guo Huang 2003, p. 36)

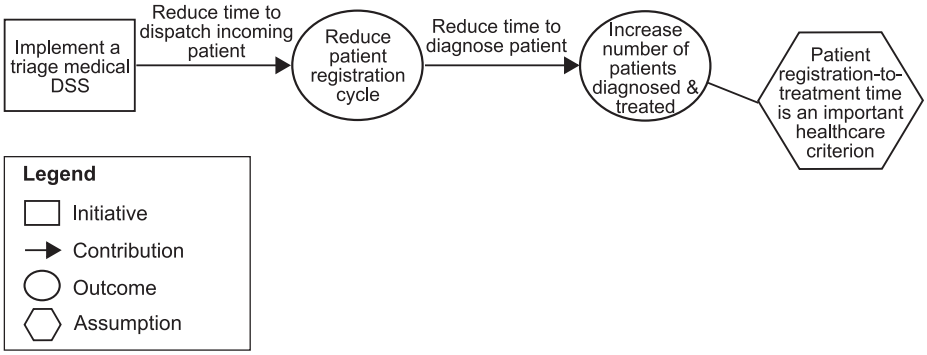

Figure 1. Benefits-realization approach results chain (adapted from Boehm and Guo Huang 2003)

Once the benefits-realization approach is completed, stakeholders can monitor the development of the project using the *value-realization feedback process* (Figure 2). As explained by Boehm and Guo Huang (2003):

"The results chain, business case, and program plans set the baseline in terms of expected time-phased costs, benefit flows, returns on investment, and underlying assumptions. As the projects and program perform to plans, the actual or projected achievement of the cost and benefit flows and the assumptions' realism may become invalid, at which point the project team will need to determine and apply corrective actions by changing plans or initiatives, making associated changes in expected cost and benefit flows." (p. 37)

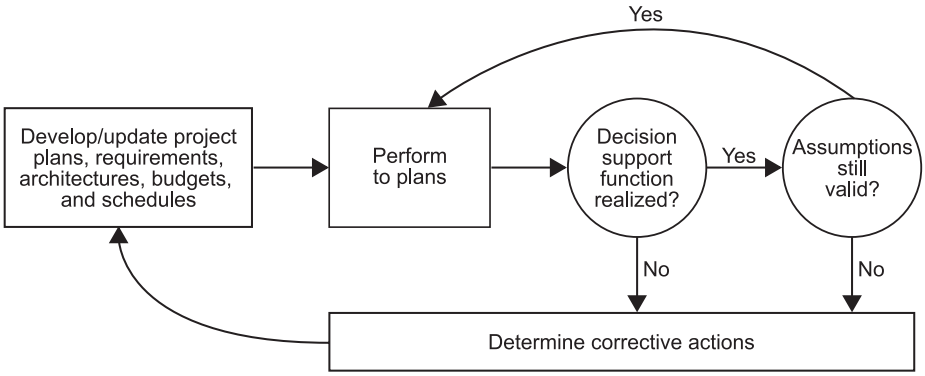

Figure 2. The realization feedback process (adapted from Boehm and Huo Huang 2003)

One obvious advantage of this feedback process is its ongoing consideration of the assumptions' validity. The development of an organizational DSS can take time and the project's plan can change several times during the whole process. It is therefore important to regularly monitor the process to be sure that the system still meets the local needs and helps answer the right questions (the popular "do the right thing" proposition). Otherwise, a DSS can be seen as very successful in terms of cost-oriented earned value, but a complete disaster in terms of actual organizational value.

The feedback process used in value-based software engineering focuses on value realization. Assessing the value of a transaction information system is a difficult task (Tillquist and Rodgers 2005). However, assessing the value of a DSS is even more difficult, since its main function (supporting decision-makers) leads to effects that are very difficult to measure in isolation from the complementary processes and activities in which the DSS is embedded. Even worse, measuring the value of a DSS during its development is almost impossible. Therefore, the feedback process that we propose in Figure 2 focuses more of the realization of the decision support function of the DSS ("do the thing right") rather than on an objective measure of value realization.

In the next section, we show how this context-based approach can be used for the development of a DSS, using the example scenario of a medical DSS for the triage of patients in an emergency room.

4. Modeling the Context-Based Development of a DSS

The basic idea is to use initiatives, contributions, outcomes, and assumptions to explicitly model the context-based elements of the DSS in the results chain. For the purpose of this paper, Figure 3 extends the results chain of Figure 1 by adding new initiatives explicitly dealing with the two issues we mentioned in Section 2, namely the user interface and the maintenance and legacy issues. These additional initiatives are shown with bold borders. Needless to say, a realistic and complete results chain for a triage medical DSS would be much more complicated than Figure 3, with many more initiatives, outcomes, and assumptions related to other value propositions. For the sake of simplicity, however, we decided to limit the scope of the results chain in order to improve its readability. The ultimate purpose of the figure is to show how the benefits realization approach allows DSS designers to dynamically identify the project's success-critical stakeholders and to determine their propositions in terms of decision support.

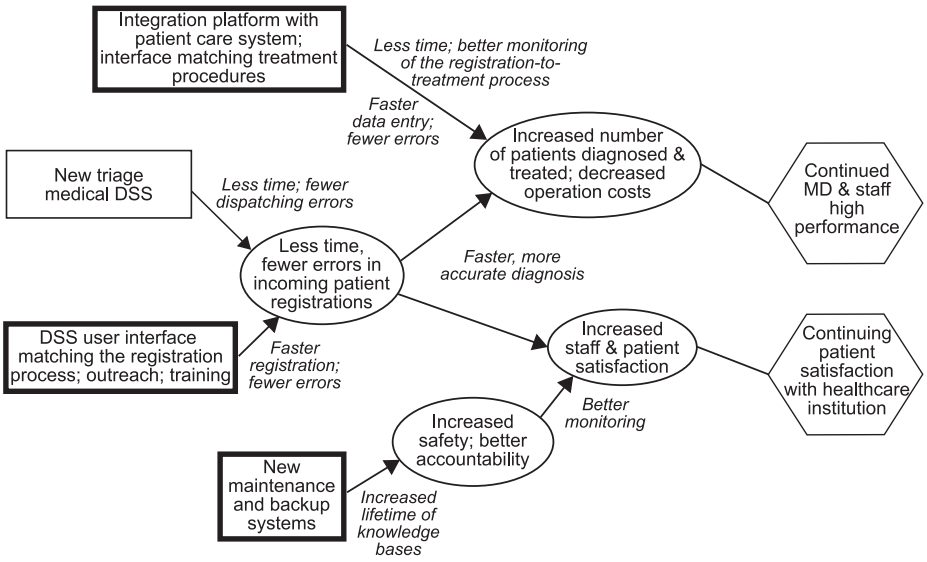

Figure 3. Extended results chain for a triage medical DSS. Stakeholders and contextual information are explicitly considered

Figure 3 shows that MDs, nurses, staff, and patients are considered as important stakeholders in the initiatives. Their satisfaction and acceptance of the new system is deemed critical for the DSS success. Expected outcomes such as the increased staff and patient satisfaction, or initiatives focusing on user and system interfaces acknowledge this fact.

The results chain also expresses the extra precautions that need to be taken to ensure patient safety and avert the possibility of costly medical malpractice lawsuits.

The initiative to tightly integrate the triage DSS with the rest of the patient care system are necessary to ensure patients are being given the drugs corresponding to the diagnosis. The initiative to include maintenance and backup systems illustrates the necessity to safeguard the data and knowledge bases of the entire infrastructure, for increased safety and better accountability.

Once the initiatives of the results chain are set, the DSS designers can turn them into plans, requirements, architectures, budgets, and schedules. At that stage, traditional DSS development processes and methodologies can be used with a context-based flavor. For example, Table 2 extends Table 1 (Sage's DSS design and development life cycle) with contextual information. The original life cycle proposed by Sage comes with long, context-free checklists for each of the seven phases, encompassing all kinds of issues, such as system objectives, user commitment, frequency of use, expected effectiveness improvement, planning horizons, leadership requirements, training, political acceptability, institutional constraints, management support, value of information, hardware and software requirements, functional performance, to name a few. We maintain that these context-free checklists are impractical, because they bury the DSS designer under considerations that are not necessarily relevant in the context of the target DSS. Using results chain, on the other hand, the DSS designer can extract the contextual issues relevant to his task and associate them to the correct steps of the process.

Table 2. Extended context-based development lifecycle

1. Identify requirements specifications based on contextual issues
• *User interface requirements from the end-user perspective (nurses, MDs, etc.)*
• *Data integration requirements (global patient care system)*
• *Workflow requirements (treatment procedure)*
• *Maintenance and backup requirements*
• *...*
2. Preliminary conceptual design
• *Determine the appropriate design and implementation approach for the DSS*
• *Identify the specific input/output formats to satisfy user interface requirements*
• *Determine specific hardware and software requirements*
• *Identify conceptual specifications for the transactional and backup databases*
• *...*
3. Logical design and architectural specifications
• *User interface design and early prototyping*
• *Process modeling for the registration-to-treatment procedures*
• *Specification of a distributed architecture adapted to the required integration with the patient care system*
• *Data modeling and strategic design of the maintenance and backup architecture*
• *...*
4. Detailed design and testing
• *Testing of the user interface with end users*

- *Testing of the process model with regard to the DSS integration in the patient care system*
- *Testing of the architecture (resilience, reliability, scalability)*
- *Use of failure scenarios to test the maintenance and backup system*
- *...*

5. Operational implementation

- *Production of the components of the operational DSS*
- *Implementation of the distributed functionalities with integration in the patient care system*
- *Implementation of the maintenance and backup system and integration with the DSS*
- *Selection and training of a test group*
- *...*

6. Evaluation and modification

- *Preparation of an evaluation methodology and of specific evaluation tests for each critical aspect of the DSS (user acceptance, system integration, architecture resilience and scalability)*
- *Execution of the test procedure with the test group, for each critical aspect of the DSS*
- *...*

7. Operational deployment

- *Final design and implementation of the DSS*
- *Training of all user groups*
- *Continuous monitoring of postimplementation realization of the DSS*
- *...*

Instead of focusing on traditional performance issues, the context-aware DSS designer can focus on the most salient and relevant aspects of the target system. In our example DSS for medical triage, the requirements specification phase should focus on user interface requirements (to guarantee the system acceptability) and on data integration and workflow activities (to prepare a smooth integration with the global patient care system). Maintenance and backup considerations are traditionally postponed until the last step of the process (operational deployment). In our example, however, we clearly identified maintenance and backup as critical value propositions for the DSS. Therefore, it is wise to move the corresponding requirements to the first step of the process. Traditional methodologies described in the DSS literature, such as functional mapping (Blanning 1979), decision graph (Martin 1982), or descriptive and normative modeling (Stabell 1983) can be used to identify requirements specifications.

The primary goal of the preliminary conceptual design phase (step 2) is "to develop conceptualization of a prototype that is responsive to the requirements identified in the previous phase" (Sage 1991). In our example, this consists in determining the appropriate design and implementation approach for the DSS (for example, the evolutive approach based on prototyping; Courbon et al. 1979); identifying input and out-

put formats to satisfy the user interface requirements; determining specific hardware and software requirements; and working on the conceptual specification of the transactional and backup databases.

Whereas the first two phases are conceptual in nature and produce documents, the third phase (detailed design and architectural specifications) should result in a usable prototype that end users can get familiar with. Confronted with a running system for the first time, end users can update their mental models about the target DSS and refine their requirements during early tests (step 4). Even though the realization feedback process (check back to Figure 2) should be executed after each phase, its execution is particularly important here. With a prototype at hand, it becomes possible to determine if the expected decision support functionalities are likely to be realized by the final DSS ("do the thing right"), and if the assumptions defined in the results chain are still valid ("do the right thing"). The feedback process reminds the stakeholders that the actual or projected achievement of the decision support functionalities and the assumptions' validity may become invalid, in which case the DSS designers will need to iterate to an earlier phase and apply corrective actions by changing initiatives and outcomes in the results chain.

If no corrective action is needed, the various DSS components are implemented and aggregated into a complete system ready to be tested and evaluated by a test group (step 5). During the evaluation phase, each critical aspect of the DSS is subjected to a specific set of tests conducted with the test group. For example, user interface testing can be conducted using methodologies described in the human-computer interaction (HCI) literature. This phase (step 6) is again an important milestone for the realization feedback process, since it represents the last chance to determine corrective actions before the operational deployment and final implementation of the DSS. Once the DSS is deployed, all user groups (i.e., nurses, MDs, and administrative staff) need to be properly trained. To guarantee the expected lifetime of the system, continuous monitoring of postimplementation realization must be provided. If necessary, modifications must be retrofitted in the DSS. In other words, the realization feedback process still has to be regularly executed, even after the seven steps of the life cycle.

5 Concluding Remarks

In this paper, we have described a new approach to formalize the notion of context in the study of DSS development. The proposed solution relies on the concept of value-based software engineering. It provides DSS designers and builders with the appropriate techniques to explicitly consider the context of the target DSS during the entire development process.

The first technique is the benefits realization approach, which uses diagrams called results chains to determine and reconcile the value propositions of the project's success-critical stakeholders. The results chain establishes a framework linking initiatives to contributions and outcomes. The results chain also links to assumptions, which condition the realization of outcomes.

The second technique is the realization feedback process, which regularly monitors the realization of the expected decision support functionalities and the validity of the assumptions. If the decision support functionalities are not realized, or the assumptions' realism becomes invalid, the DSS designers need to to determine and apply corrective actions by changing plans or initiatives.

We then provided an example to illustrate how a somewhat impractical, context-free DSS development life cycle can be turned into a context-based life cycle focusing on the most success-critical aspects of the target DSS, without overwhelming the DSS designers with long checklists that are not necessarily relevant in the context of the target system.

The inability of the DSS community to come up with unified and standardized methods to develop decision support systems is a recurring topic that has kept researchers and practitioners busy for the past three decades. We strongly believe that our approach finds a partial answer to the problem, by explicitly acknowledging the fact that DSS can be widely different in their goals, scope, depth, lifetime, and costs. Rather than looking for an elusive context-free, one-size-fits-all solution, we propose a context-based set of tools able to formalize the DSS environment and context before choosing the appropriate development methodology. It is our hope that the approach described in this paper provides a vehicle for researchers and practitioners to develop better and more successful DSS.

References

Arinze, B. (1991). "A Contigency Model of DSS Development Methodology." Journal of Management Information Systems 8(1): 149-166.

Bateman, A. and J. Murphy (1994). Migration of Legacy Systems, School of Computer Applications. Dublin: Dublin City University: Dublin.

Bianchi, A., D. Caivano, V. Marengo and G. Vissagio, "Iterative reengineering of Legacy Systems", IEEE Transactions on Software Engineering, vol 29(3), pp. 225-241, March 2003.

Blanning, R. W. (1979). "The functions of a decision support system." Information and Management 2(September): 71-96.

Boehm, B. and L. Guo Huan (2003). "Value-Based Software Engineering: A Case Study." Computer 36(3): 33-41.

Brodie, M. and M. Stonebraker (1995). Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. San Francisco: Morgan Kaufmann Publishers Inc.

Comella-Dorda, S., K. Wallnau, R. C. Seacord and J. Roberts (2000). "A Survey of Legacy modernisation approaches", Carnegie Mellon University, Software Engineering Institute, pp. 1-20, April 2000.

Courbon, J.-C., J. Drageof and J. Tomasi (1979). "L'approche évolutive." Informatique et Gestion 103(Janvier-Février).

Finlay, P. N. (1994). Introducing decision support systems. Oxford, UK Cambridge, Mass., NCC Blackwell; Blackwell Publishers.

Gachet, A. and P. Haettenschwiler (2005) "Development Processes of Intelligent Decision Making Support Systems: Review and Perspective", in Gupta, J., Forgionne G., and Mora, M. (eds) Intelligent Decision-Making Support Systems (I-DMSS): Foundations, Applications and Challenges, Springer-Verlag, London [forthcoming]

Holzinger, A. (2005). "Usability Engineering Methods for Software Developers." Communications of the ACM 48(1): 71-74.

Keen, P. G. W. and M. S. Scott Morton (1978). Decision support systems : an organizational perspective. Reading, Mass., Addison-Wesley Pub. Co.

Marakas, G. M. (2003). Decision support systems in the 21st century. Upper Saddle River, NJ, Prentice Hall.

Martin, M. P. (1982). "Determining Information Requirements for DSS." Journal of Systems Management(Decembre 1982): 14-21.

Sage, A. P. (1991). Decision support systems engineering. New York, Wiley.

Saxena, K. B. C. (1991). Decision support engineering: a DSS development methodology. 24th Annual Hawaii International Conference on System Sciences (HICSS'91), Los Alamitos, CA, IEEE Computer Society Press.

Saxena, K. B. C. (1992). DSS Development Methodologies: A Comparative Review. 25th Annual Hawaii International Conference on System Sciences (HICSS'92), Los Alamitos, CA, IEEE Computer Society Press.

Sprague, R. H. and E. D. Carlson (1982). Building effective decision support systems. Englewood Cliffs, N.J., Prentice-Hall.

Stabell, C. B. (1983). A Decision-Oriented Approach to Building DSS. Building Decision Support Systems. J. L. Bennett. Reading, MA, Addison-Wesley: 221-260.

Thomas, C. and N. Bevan (1996). Usability Context Analysis: A Practical Guide. Teddington, UK, National Physical Laboratory.

Thorp, J. (2003). The information paradox : realizing the business benefits of information technology. Toronto, ON, McGraw-Hill Ryerson.

Tillquist, J. and W. Rodgers (2005). "Using Asset Specificity and Asset Scope to Measure the Value of IT." Communications of the ACM 48(1): 75-80.

Turban, E. (1995). Decision support and expert systems : management support systems. Englewood Cliffs, N.J., Prentice Hall.

Wallace, R. H., J. E. Stockenberg and R. N. Charette (1987). A unified methodology for developing systems. New York, NY, Intertext Publications : McGraw-Hill.

Wu, B., D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade and D. O'Sullivan (1997), "The Butterfly Methodology: A gateway free approach for migrating Legacy Information Systems", In Proceedings of the ICECCS97, Villa Olmo:Italy.