A DECISION-ORIENTED MODEL OF SOFTWARE ENGINEERING PROCESSES

Claudine Toffolon
Littoral University

Salem Dakhli
Paris-Dauphine University

Abstract

Information Systems (IS) has become one of the most valuable assets of modern organizations where they play a critical role in supporting operational and decision processes. Nevertheless, despite the large part of organizations resources invested in information technology, development of information systems (IS) faces many problems recognized in the term "software crisis". In order to reduce the economic and social impacts of the software crisis, one widely acknowledged approach has been to improve software processes and software development methods supporting them. Nevertheless, as stressed by many authors, such solutions of the software crisis are partial and incomplete and present many weaknesses related to their technical orientation. In this paper, we present a decision-oriented model of software engineering process that integrates the multi-stakeholders nature of IS development, maintenance and use and aims at improving well-established software process models. Our framework models software engineering as nexus of decisions under uncertainty carried out by many stakeholders who behave according to principal-agent contracts.

Keywords: organizational actor, decision, contract, computer solution, project space

1 INTRODUCTION

With the ever-increasing penetration of information technology into the daily functioning of society, Information Systems (IS) has become one of the most valuable assets of modern organizations where they play a critical role in supporting operational and decision processes. Indeed, modern organizations need IS which provide them with instruments to leverage core business competencies, accelerate innovation and time to market, improve cycle times and decision making, strengthen organizational commitment, and build sustainable competitive advantage. Nevertheless, despite the large part of organizations resources invested in information technology, development of information systems (IS) faces many problems recognized in the term "software crisis" (Gibbs 1994) (Neumann 1995) (Pressman 2004). Among these problems are low developers productivity, a large number of software project failures, an inadequate alignment of IS with business requirements, and user resistance. Moreover, the "productivity paradox" (Solow 1987) (Dedrick et al. 2003) which states that the links between information technology and white-collar workers productivity have historically been weak, is partly related to the software crisis.

In order to reduce the economic and social impacts of the software crisis, one widely acknowledged approach has been to improve software processes and software development methods supporting them. A software process may be viewed as a nexus of development and maintenance activities associated with the phases of the software lifecycle (Cugola et al.

1998) (Osterweil et al. 2005). A software process model is an abstract representation of a process which presents a description of a process from a particular perspective. The waterfall model (Boehm 1976), the spiral model (Boehm 1988) and the Rational Unified Process (RUP) (Sommerville 2006) are examples of software process models. A software development method provides systematic and predefined guidelines for carrying out at least one complete activity of the software development process.

Nevertheless, as stressed by many authors, such solutions of the software crisis are partial and incomplete and present many weaknesses related to their technical orientation. In particular, well-established processes and methods neglect many important aspects of software notably economic, organizational, and human aspects (Abdel-Hamid et al. 1991) (Boehm 1988) (Fitzgerald 1996) (Fitzgerald 1998). To improve IS quality, many academics have stressed the multidimensional nature of IS and the multiplicity of the organizational actors involved in their development, maintenance, and use (Kling 1996) (Lyytinen 1987) (Toffolon et al. 2002). According to these authors, situated software development processes and methods taking into account all the aspects of software are required to build effective IS that meet modern organizations needs. We think that modeling software engineering as a decision-intensive process may help researchers and practitioners understand software complexity in order to reduce the software crisis negative impacts. In this paper, we present a decision-oriented model of software process that integrates the multidimensional and multi-stakeholders nature of IS development, maintenance and use and aims at improving well-established software process models. Our framework models software engineering as nexus of decisions under uncertainty carried out by many stakeholders who behave according to principal-agent contracts. It is based on the software global model (Toffolon et al. 2002). The rest of this paper is organized as follows. Section 2 presents a review of the literature related to decision modeling in software engineering. In section 3, we describe synthetically the theoretical foundations of our framework. Section 4 is dedicated to the detailed presentation of the proposed framework. In section 5, we conclude our paper by listing future research directions.

2 LITERATURE REVIEW

As stressed by many authors, decision-making is crucial for guiding the development of IS needed by modern organizations. (Davis et al. 1988) propose a framework which provides instruments for analyzing the similarities and differences among alternate life cycle models, helping software engineering researchers describing the probable impacts of a new life cycle model; and supporting software practitioners decide on an appropriate life cycle model to utilize on a particular project or in a particular application area. (Cardenas-Garcia et al. 1991) note that few models and fewer tools have been developed to help the project manager in the decision-making process supporting development activities. Such process permits aiding the project managers in picking one design strategy over another and determining which design will best meet management's objectives with respect to cost, schedule, and functionality. Moreover, the decision-making is critical for productivity improvement through software reuse. Indeed, it helps the software project team in evaluating previously written components in order to determine whether they meet users current needs, whether they need to be modified, or whether they should be ignored and a new component designed. These authors propose a framework that addresses these issues by describing a design- evaluation mechanism and a prototype implementation of that mechanism that can aid the software manager in making such decisions. This framework includes an evaluation mechanism for comparing design attributes with an underlying utility function model for determining the

appropriateness of software prototyping. (Toffolon et al. 2002) propose an evaluation method of software prototyping which permits determining the value of information provided by software prototyping and determine the optimal number of iterations associated with prototypes development. (Davis 2003) analyses the requirements triage process of determining which requirements a software product should satisfy given the time and resources available. He points out that among the most important activities of this process is the selection activity which consists in determining a subset of requirements that optimizes the probability of the software product's success in its intended market, whether commercial or internal, relative to the resource constraints. According to (Sullivan et al. 1999), the theories that support earlier approaches to software engineering economics cannot account for the value of flexibility since they are based on static net present value. They use real options theory to build a framework within which they analyze software design concepts. This framework interprets important design principles in real options terms and discusses the validity of the approach for software; impediments to quantitative application; and how qualitative options thinking can nevertheless improve design decision-making. (Falessi et al. 2006) stress that individual and team decision-making have crucial influence on the level of success of every software project. They remark that even though several studies were already conducted, which concerned design decision rationale documentation approaches, a few of them focused on performances and evaluated them in laboratory. To deal with this weakness of existing studies, they propose a technique to document design decision rationale, and evaluate experimentally the impact such a technique has on effectiveness and efficiency of individual and team decision-making in presence of requirement changes. The main results of this study conducted as a controlled experiment, show that, for both individual and team-based decision-making, effectiveness significantly improves, while efficiency remains unaltered, when decision-makers are allowed to use, rather not use, the proposed design rationale documentation technique. (Tockey 2005) observes that modern organizations do not maximize the return on their investments in information technology because most software professionals do not know how to consider the business aspects of their software decisions. Many software professionals do not even know that doing so is important. This author considers that business consequences should play a critical role in all software technical choices, from choosing which projects to do, selecting software development processes, choosing algorithms and data structures, all the way to determining how much testing is enough.

These studies are examples of researchers and practitioners work related to the role of decision-making in software engineering. They agree that to be effective, the software development process must be supported by a decision process. Nevertheless, despite their important contribution to the software engineering discipline, they are of limited scope since they do not take into account the organizational and the human aspects of software engineering.

3 THEORETICAL FOUNDATIONS

According to the Leavitt's model modified by (Stohr et al. 1992), an organization is composed of five interconnected elements (structure, tasks, people, production technology, IT) which interact with environment (Figure 1).

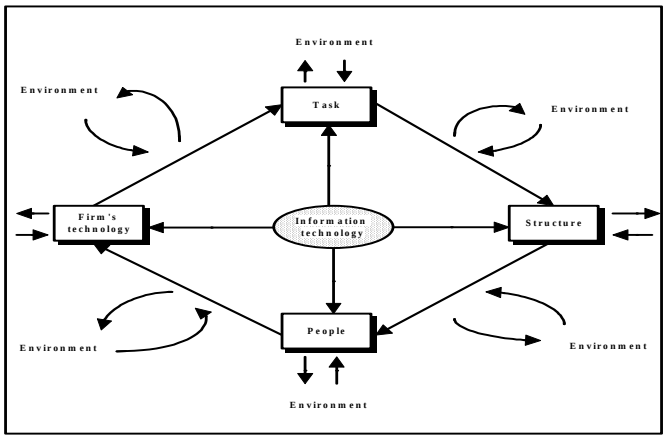


Figure 1: The Leavitt Model

This model provides us with a static view of organizations. In order to take into account the behavioral aspects of the organization's components, (Toffolon 1996) improve this model by using the economic agency theory (Alchian et al. 1972) and the transaction costs theory (Coase 1937) (Williamson 1981) to describe the information flows sent and received by the organizational entities. She identified three categories of organizational entities (actors, resources, tasks) which exchange three types of information flows related to decisions, contracts and products. The organization's actors are people involved in operational or decision-making processes. They are either producer or consumer of goods and services within the organization. Actors are linked by organizational contracts related to goods and services and information flows exchanged. They use resources to carry out the organization's business processes composed of tasks. Resources are either technology or organizational structure (rules, protocols,…).

Well known definitions of IS often stress their technical aspects while neglecting totally or partly their organizational, social and economic aspects. To take into account the main aspects of IS, we consider their role in supporting modern organizations operational and decision processes. According to the model presented previously, carrying out contracts linking organizational actors is strongly dependent on information they exchange. IS may be viewed as a set of tools providing information needed by organizational actors to carry out operational and business processes related to roles they play within organization. In particular, at least three aspects must be considered while analyzing IS. These aspects include actors concerned by IS, businesses exercised by these actors and relationships between them. In that way, the economic agency theory permits identifying organizational actors concerned with IS and assimilating relationships between them to contracts. Therefore IS are governed by a nexus of informational contracts linking organizational actors with conflicting interests and points of view. At a given time, each organizational actor plays the role of consumer (principal) or producer (agent) of goods and services under the contracts which link him to the other organizational actors. The transaction cost theory permits identifying businesses and tasks of actors involved in IS development and use. On the basis of this theory, (Toffolon et al. 2002) identified four types of actors associated with four businesses: the customer, the architect, the developer and the end-user. Each organizational actor involved in IS development or use performs contracts linking him to the other organizational actors within a specific project space where he plays the principal role. The four spaces associated with organizational actors businesses are:

❶ **The problem space** where are defined the customers and users problems and their organizational solutions. This space represents the customer's business.
❷ **The solution or architectural space** where are defined the computer solutions of the customer/user's problems. This space represents the architect's business.
❸ **The construction space** where these solutions are implemented. This space represents the developer's business.
❹ **The operation space** where are evaluated the software's usability from the user's perspective as well as its contribution to the organization's competitiveness. This space represents the end user's business.

An organizational may have two categories of roles: producer (agent) or consumer (principal) of software artifacts. A role played by an organizational actor in one of the four spaces is either principal or secondary. In each space, it is possible that there are many actors assuming secondary roles, but there can be only one organizational actor involved in a principal role; moreover, an actor can play a secondary role in many spaces, but a principal role only in one (every actor plays the principal role in some space). According to (Toffolon et al. 2002),

- In the problem space, the customer plays the principal role while the user and the architect play secondary roles.
- In the solution (architectural) space, the architect plays the principal role while the customer and the developer play secondary roles.
- In the construction space, the principal role is played by the developer and the secondary roles are played by the architect and the end user.
- In the operation space, the principal role is played by the user and the secondary roles are played by the developer and the customer.

According to the software engineering global model presented in the previous section, software engineering refers to the set of activities carried out by organizational actors concerned with computerization while realizing agency contracts linking them, in order to build software solutions. Software engineering takes place at two levels: horizontal (inter-spaces) level and vertical (intra-spaces) level. The horizontal level is concerned with those activities related to transitions between two project spaces and results in exchange of software artifacts by actors who play principal roles in one space. Activities realized at the horizontal level depend on outputs issued from software engineering activities belonging to the vertical level and carried out within each project space. The horizontal software engineering activities take place according to an iterative process designated by the acronym « **PACO »** (**P**roblem-**A**rchitecture-**C**onstruction-**O**peration): the definition of a computer solution of an organizational problem permits the transition from the problem space to the solution space, the implementation of this solution expresses the transition from the solution space to the construction space, the installation of the software artifacts built in the construction space results in the transition from this space to the operation space, the description of problems and needs generated by the use of the software installed permits the transition from the operation space to the problem space. The human interface between two spaces is carried out by the project's actors who play a principal role at once in these two spaces. The vertical software engineering activities are based on process models related to organizational context, technical maturity, and know-how within each project space. For example, vertical software engineering in the construction space may be supported by a mix of many process models like the waterfall model, the spiral model, the prototyping lifecycle, and agile processes. The

vertical software engineering activities are either project management activities or software development activities.

4 THE DECISION-ORIENTED MODEL OF SOFTWARE ENGINEERING PROCESSES

Computerization consists in providing computer-based solutions to support organizations operational and decision processes. Such solutions are composed of three parts: hardware, networks and software artifacts (specifications, models, programs, documentation) called software solution and issued from the software development process. Software engineering is the conduct of software process that is, applying means to produce and maintain a solution to a real-world problem in the software domain. Software engineering has been defined by many authors and institutions (Nauer et al. 1969) (IEEE Std 610.12-1990) (Pressman 2004). In particular, (IEEE Std 610.12-1990) defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software", that is, the application of engineering to software. According to (Pressman 2004), software is composed of three parts: instructions, data structures and documents. Instructions are computer programs that when executed provide desired function and performance. Data structures enable the programs to adequately manipulate information. Documents describe the operation and use of the programs. Software can also be seen as an interface between the problem space and the computer. In the rest of this section, we demonstrate why decision-making is inherent in each software engineering activity prior to presenting a decision-oriented model of this discipline.

4.1 The Decision-Oriented Nature of Software Engineering Activities

At each iteration of the (**PACO**) process, an organizational actor is linked by two agency contracts to the organizational actors who play secondary roles within his own space. He is the principal part of one contract and the agent part of the other contract. For example, the architect is the principal part of the agency contract linking him to the developer who implements the computer solution architecture in the solution space. By another way, he is the agent part of the agency contract linking him to the customer who asks him to define a computer solution in order to support a business process. Besides, each organizational actor transforms a set of software artifacts (inputs) he received from a principal and delivers the transformed software artifacts (outputs) to an agent who will transform it again according to the software development process. Each software engineering activity includes decisional and operational tasks. In particular, horizontal software engineering activities and vertical software engineering activities related to project management are decision-intensive while the proportion of decision tasks in vertical software engineering related to software development activities increases with level of the lifecycle stage to which they belong. Besides, upper vertical software engineering activities are more decision-oriented than lower ones. Indeed, information needed to accomplish upper vertical software engineering activities is often ill-structured, incomplete, uncertain and originates from many internal and external sources. Information needed to accomplish lower software engineering activities is often structured, complete, and originates from technical well-known sources. Horizontal software engineering activities and vertical upper vertical software engineering activities rely in a great part upon tacit knowledge while vertical lower software engineering activities uses detailed explicit

knowledge. For example, project planning is a software engineering management activity while requirements engineering is a vertical upper software development activity. These two activities are decision-intensive software engineering activities. Project planning consists in developing a project plan which includes defining project goals and objectives, specifying tasks or how goals will be achieved and what resources are needed, and associating budgets and timelines for completion. Project planning is a set of vertical software engineering tasks accomplished before starting the software development activities. At this stage, there is not enough information to support fine-grained effort and duration estimation and help organizational actors reason about the cost and schedule implications of their computerization objectives. To deal with uncertainty resulting from lack of information, the project manager usually negotiates cost/schedule/quality tradeoffs while setting project budgets and schedules. Consequently, project planning is a decision-intensive activity. Requirements engineering follows project planning and may be considered as the beginning of software development activities. This activity is difficult to realize because of the immaterial nature of software artifacts. Moreover, at this stage little is known about the final software product under consideration and communication between the software problem side (the problem space) and the software solution side (the solution space) is often difficult. Therefore, software architects who define the computer solution usually build prototypes to resolve high-risk issues involving user interfaces, software and system interaction, performance, or technological maturity. Such prototypes, needed to reduce uncertainty inherent in these issues, provide information to determine alternative options related to global and detailed architectures and support the software architects in deciding what is the most appropriate option and making a decision to move forward with development. By another way, information provided by software prototyping activities induces more complexity which makes the software project more difficult to achieve and manage. Indeed, more information related to the end-users requirements may increase software artifacts complexity in particular when such information describes more needs and constraints. This in turn increases software project complexity. On the one hand, more complex tasks are often needed to develop more complex software artifacts. On the other hand, more information about software requirements may increase pressures on software engineering management activities. Figure 2 illustrates the tradeoff between complexity and uncertainty associated with software prototyping activity.

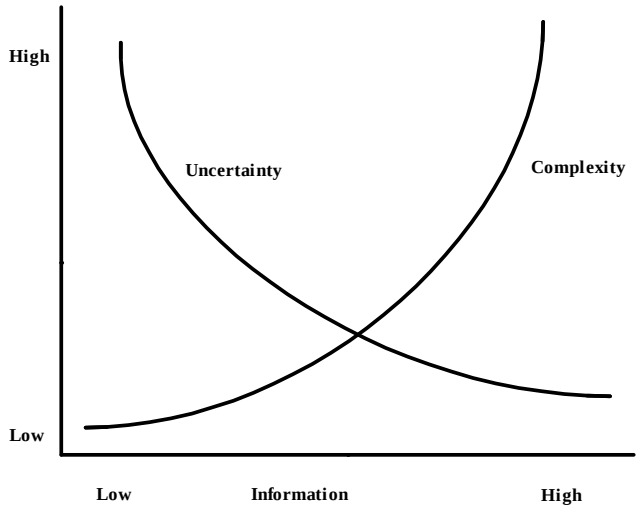
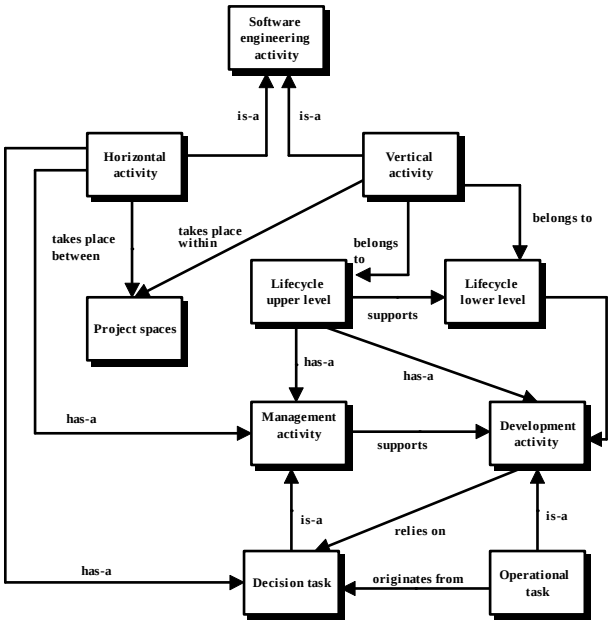


Figure 2: The tradeoff between uncertainty, complexity and information

Generally, the reduction and the control of such a complexity is associated with a decision-intensive activity which consists in estimating the value of information issued from software prototyping activity, and determining the optimal number of iterations during the prototyping process (Toffolon et al. 2002). Figure 3 presents a meta-model activities which illustrates that software engineering is a nexus of intertwined, concurrent and mutually supportive decision and operational activities. In the next sub-section, we describe synthetically the decision-making behavior of the organizational actors involved in software artifacts development, maintenance, and use.

Figure 3: Software engineering activities metamodel



4.2 The Decision-Making Behavior of Software Engineering Stakeholders

According to (Toffolon et al. 2002), software engineering is a nexus of contracts between the stakeholders of the future software product under development. During the progress of the software engineering process, these stakeholders behave as decision-makers in an uncertain universe of discourse. As stressed by (Sullivan et al. 1999), the software engineering process and the software artifacts issued from this process emphasize structural concerns instead of value-added ones. These authors analyze the software engineering discipline through three issues: the process, the product, and the timing of commitments to design decisions. They demonstrate that decisions criteria associated with process and product are largely structural while criteria related to the timing of commitments to design decisions rely on informal rules of thumbs. According to these authors, the structural decision-making criteria do not constitute the best foundations of the decision-making process supporting the software processes. Instead, they argue that value added by software engineering is a more useful foundation of the decision-making behavior of the computerization stakeholders. Since value added by software engineering results from flexibility to make risky decisions related to software artifacts development, maintenance and use, flexibility is modeled as a real option which consists in the right without an obligation to make an investment contingent on a future outcome. Despite we agree with these authors on the contribution of real options theory to understanding of the decision-making process that supports software engineering, we think that a deeper analysis of decisions related to software engineering is needed. This analysis which consists in determining the nature and the characteristics of such decisions, is based on the concept of essential and accidental characteristics of software engineering (Brooks 1987). Complexity and risk are examples of essential characteristics of software engineering while uncertainty and complication are examples of accidental characteristics of this discipline. The effectiveness of software engineering relies on the management of the essential characteristics and the reduction of the accidentals characteristics impacts. For example, software engineering risks is managed while uncertainty is reduced. Risk management consists in analysis and evaluation of software risks and selection of the most appropriate preventive and curative actions to avoid their potential losses. There are many approaches that permits uncertainty reduction. Software informative prototyping is an example of such an approach. In the same way, there are many development approaches and methods aimed at managing software engineering complexity. Such methods are either structured (e.g. SA/SD), systemic (e.g. MERISE), or object-oriented (e.g. OOSE, OMT). In order to take into account the essential or accidental nature of software engineering characteristics, we distinguish two categories of decisions: decisions to manage essential characteristics and decisions to reduce the negative impacts of accidental characteristics. Since uncertainty is the most important accidental characteristic which pervades software engineering, we restrict the second category of decisions to those decisions aimed at uncertainty reduction. The differences between these two categories of decisions are important. Firstly, the value of decisions belonging to the first category result from flexibility provided by the second category. Indeed, either building a computer solution or developing an informative prototyping are irreversible decisions. Nevertheless, since an informative prototype costs less than a complete software system, it may be helpful to wait and gather more information through prototyping before developing the final computer system. Secondly, decisions to manage essential characteristics of software engineering take place in a risky universe of discourse. Therefore, they differ from decisions to reduce uncertainty which, by their nature, are made in uncertain universe of discourse. The relationships between decisions to manage essential characteristics of software engineering

and flexibility suggests that using real options theory to model such decisions may help capturing the value added by these kind of decisions. By another way, using real options is not appropriate to model decisions related to reduction of uncertainty inherent in software engineering. Indeed, the value added by such decisions is not related to flexibility but to the value brought by the investment in acquiring information permitting to move from uncertain states of nature to risky states of nature. Besides, decisions to reduce uncertainty are often considered as a prior stage that precedes decisions to manage software engineering essential characteristics. For instance, the Boehm's spiral model (Boehm 1988) recommends software prototyping to reduce uncertainty inherent in software engineering, prior to software solution building. At a micro-level, the decision-making process which supports software engineering may be modeled as follows: at à given period, an organizational actor involved in computerization may make a decision to manage a essential characteristic of software engineering or a decision to reduce uncertainty. The later decision permits moving from an uncertain universe of discourse to a risky universe of discourse and results in at least one decision to manage essential characteristics. For example, at the first period, user requirements are generally composed of complete well-understood requirements and imprecise requirements. To take into account the later category of requirements in a future version of a computer solution, a software engineer may proceed by building immediately a new component to implement well-understood requirements and an informative prototype to reduce uncertainty inherent in less-understood requirements. Well-understood requirements increase the software solution complexity and to implement them relies on a decision to manage essential characteristics of software engineering. By another way, prototyping less-understood requirements results in a new flow of well-understood requirements that may be implemented by the future version of the computer solution. The following decision tree (Figure 4) illustrates the progress of the software engineering decision process. As we stressed previously, each organizational actor plays either the role of a principal or the role of an agent in the two agency contracts linking him to the organizational actors who play a secondary role within its own space. For example, the architect plays the principal role in the solution while the customer and the developer play secondary roles in this space. On the one hand, the architect is the agent part of the agency contract linking him to the customer who asks him to define a computer solution in order to support a business process. On the other hand, the architect is the principal part of the agency contract linking him to the developer who implements the computer solution architecture in the solution space. Besides, each organizational actor transforms a set of software artifacts (inputs) he received from a principal and delivers the transformed software artifacts(outputs) to an agent who will accomplish further transformations prescribed by the software development process. The following decision tree (Figure 4) illustrates the progress of the software engineering decision process during a iteration. It represents the decision-making behavior of an organizational actor who contributes to computer solution construction by acting in its own space. According to this schema, the contribution of an organizational may take place either in a risky universe of discourse or in an uncertain universe of discourse. Besides, as a decision-maker, each organizational actor selects one of the following actions: cancel his contribution to the software project, solve immediately the problem submitted by an organizational actor who plays the principal role, or wait in order to gather more information about this problem and solve it later. Solving the submitted problem now takes place within a risky universe of discourse while gathering more information takes place within an uncertain universe of discourse. Information gathered allows organizational actors to move from the later to the

former universe of discourse where they may decide to build or buy the most appropriate solution of the problem to be solved.

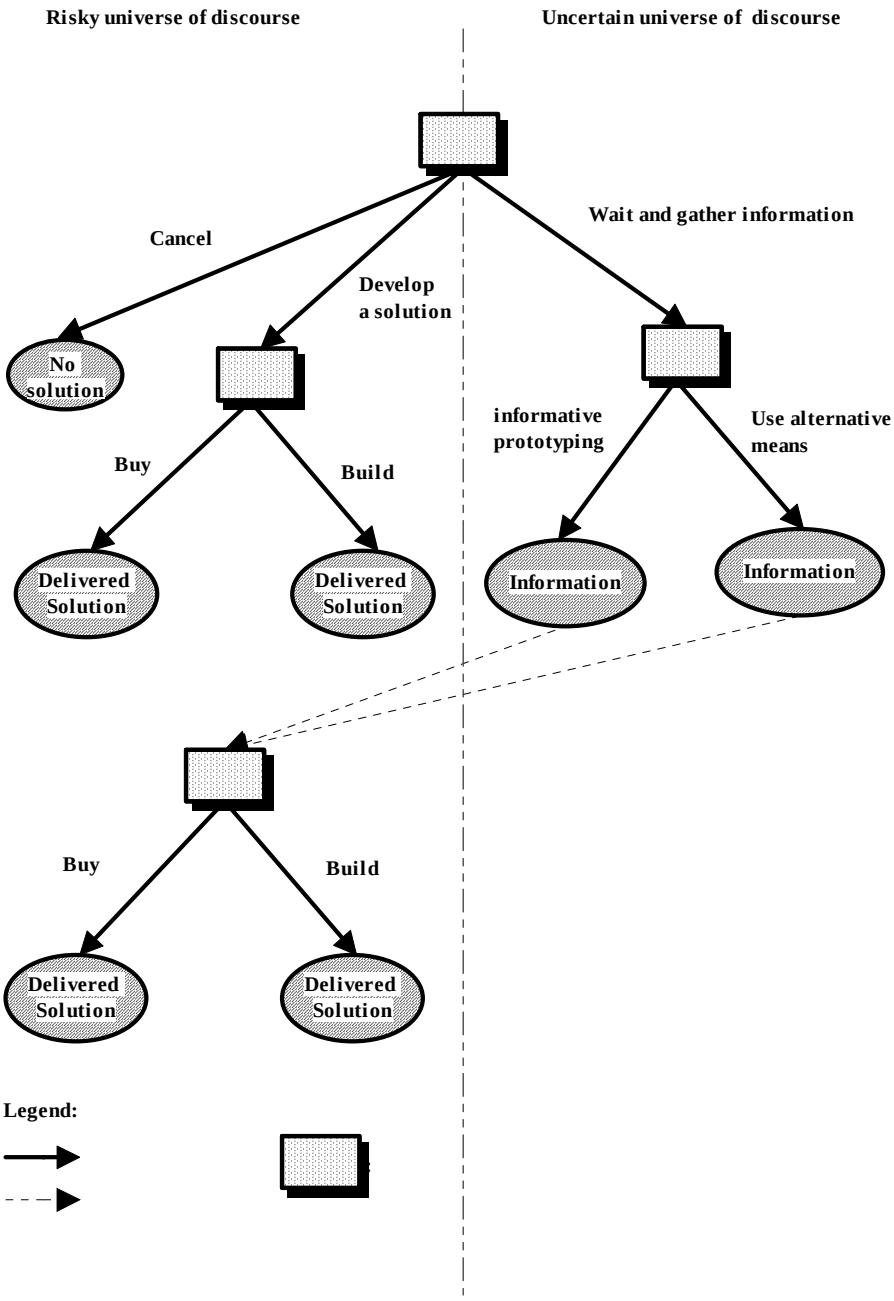


Figure 4: the software engineering decision tree

The following table (Figure 5) presents synthetically the main decisions made by the four types of organizational actors involved in the software development process.

Organizational actor	Project space	Decisions made (uncertain universe of discourse)	Decisions made (Risky universe of discourse)
Customer	Problem space	• Freeze system specifications immediately • Wait and build system specification later	• Gather information by prototyping uncertain requirements • Gather information by alternative means
Architect	Solution space	• Define immediately the computer solution • Wait and define software architecture later • Buy or build in-house	• Gather information by prototyping uncertain aspects of software architecture
Developer	Construction space	• Implement immediately the software solution • Wait and implement later the software solution	• Gather information by prototyping uncertain aspects related to implementation
End user	Operation space	• Use immediately the software system • Wait for training before using the software system • Ask for a new version now • Wait until identifying all the use problems	• Get more information through training • Get more information through communities of practice

Figure 5: Organizational actors decisions

5 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

In this paper, we have presented a decision-oriented model of software process that integrates the multi-stakeholders nature of IS development. The advantages of this framework may be summarized as follows. Firstly, it models software engineering as a nexus of agency contracts between organizational actors. Secondly, it stresses that at each iteration of the software engineering process, an organizational behaves either as principal or as an agent while carrying out the agency contracts linking him to those actors who play a secondary role in his own project space. Finally, this framework demonstrates that the contribution of each organizational actor to a software project is triggered by two categories of decisions related to the management of software essential and accidental characteristics. Decisions to manage accidental characteristics are sources of value-added despite they may result in more complex

software systems by adding an amount of information to be processed. On the one hand, they help understanding stakeholders requirements. On the other hand, they make decisions to manage essential characteristics more flexible. However, further work has to be accomplished to evaluate the value-added by flexible software engineering decisions and analyze the impact of the information gathered on the decision-making context. These impacts may be important and result in modifications of set of states of nature and heir probability distribution.

References

Abdel-Hamid T., Madnick S.E.: "Software Project Dynamics: An Integrated Approach", Prentice-Hall-Englewood Cliffs, NY 07632, 1991.

Alchian A.A., Demsetz H.: "Production, Information Costs and Economic Organization", American Economic Review, 62 (5), No. 5, december 1972, pp. 777-795.

Boehm B.W.: "A Spiral Model of Software Development and Enhancement", IEEE Computer, 21(5), May 1988: 61-72.

Boehm B.W.: "Software Engineering", IEEE Transactions on Computers, Vol.C25, No.12, 1976, pp. 1226-1241.

Brooks Jr. F.P: "No Silver Bullet-Essence and Accidents of Software Engineering", Computer, 20(4), April 1987: 10-19.Cardenas-Garcia S., Zelkowitz M V.: "A Management Tool For Evaluation of Software Designs", IEEE Transactions on Software Engineering, 17 (9), September 1991, pp. 961-971.

Coase R.: "The Nature Of The Firm", *Economica*, 4, pp. 386-405.

Cugola G., Ghezzi C.: "Software processes: a retrospective and a path to the future", Software Process: Improvement and Practice, 4(3), 1998: 101-123.

",Davis A.M., Bersoff E.H., Comer E.R.: "A Strategy for Comparing Alternative Software Life Cycle Models", IEEE Transactions On Software Engineering, 14 (10), October 1988, pp. 1453-1461.

Dedrick J., V. Gurbaxani V., K.L. Kraemer K.L.: "*IT and Economic Performance: A Critical Review of the Empirical Evidence*", ACM Computing Surveys, 35 (1), March 2003.

Falessi D., Cantone G., Becker M.: "Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation", in the Proceedings of the ACM/IEEE International Symposium on International Symposium on Empirical Software Engineering, Rio de Janeiro, Brazil, 2006, pp. 134-143

Fitzgerald B.: "An Empirically-Grounded Framework for the IS Development Process", Information & Management, 34, 1998: 317-328.

Fitzgerald B.: "Formalized Systems Development Methodologies: A Critical Perspectives", Information System Journal, 6(1), 1996: 3-23.

Gibbs W.: « Software's Chronic Crisis », Scientific American, September 1994:72-81.

IEEE Standard Glossary of Software Engineering Terminology 610.12-1990. In IEEE Standards Software Engineering, 1999 Edition, Volume One: Customer and Terminology Standards. IEEE Press, 1999.

Kling R.: "Computerization and Controversy: Value Conflicts and Social Choices", Academic Press, San Diego, 2nd edition, 1996.

Leon J. Osterweil, Carlo Ghezzi, Jeff Kramer, Alexander Wolf: Editorial. ACM Trans. Softw. Eng. Methodol. 14(4): 381-382 (2005)

Lyytinen K.: "Different Perspectives on Information Systems: Problems and Solutions", ACM Computing Surveys, 19(1), March 1987: 5-46

Nauer P., Randell BG. (eds.): "*Software Engineering: A Report on a Conference sponsored by NATO Science Committee*", NATO, 1969.

Neumann, P.G.: "Computer Related Risks", ACM Press, New York,1995.

Pressman R.S.: " Software Engineering: A Practitioner's Approach (Mcgraw-Hill Series in Computer Science) (Hardcover) 6th Edition, 2004

Pressman R.S.: " Software Engineering: A Practitioner's Approach (Mcgraw-Hill Series in Computer Science) (Hardcover) 6th Edition, 2004

Solow R.: "We'd Better Watch Out", New York Times Book Review, July 12, 1987, p. 36.

Sommerville I: "Software Engineering", 8th edition, Pearson Education, 2006

Stohr E.A. et Konsynski B.R.: "Information Systems and Decision Processes", IEEE Computer Society Press, 1992.

Sullivan K.J., Chalasani P., Jha S., and Sazawal V.: "Software design as an investment activity: A real options perspective", in *Real Options and Business Strategy: Applications to Decision-Making*, L. Trigeorgis, Consulting Editor, Risk Books, December 1999.

Tockey S.: "Return on Software: Maximizing the Return on Your Software Investment", Addison-Wesley, 2005.

Toffolon C. and Dakhli S.: "The Software Engineering Global Model", in the Proceedings of the COMPSAC'2002 conference, August 26-28, 2002, Oxford, United Kingdom.

Toffolon C.: "L'Incidence du Prototypage dans une Démarche d'Informatisation", Thèse de doctorat, Université de Paris-IX Dauphine, Paris, Décembre, 1996.

Williamson O.E.: "The Modern Corporation: Origins, Evolution, Attributes", Journal of Economic Literature, 19 (12), December 1981, pp. 1537-1568.