HUMAN-CENTERED SYSTEMS DEVELOPMENT AND USE INCONSISTENCIES

Claudine Toffolon
Université du Littoral, LIL, Calais, France,

Salem Dakhli
Université Paris-Dauphine, CERIA, Paris, France,

Keywords : Coordination, Deviation, Human-centered system, Inconsistency, Project space, Actor, Software engineering global model

Abstract : The framework we describe in this paper is composed of two parts. The first part provides a typology of deviations and inconsistencies which occur during human-centered systems development and use. This typology, based on four facets and three levels of abstraction (conceptual, detailed, technical), permits identifying other types of deviations and inconsistencies not considered in the literature. It may be useful to define methods and tools to manage and reduce deviations and inconsistencies in compliance with the organization's constraints, priorities and technical maturity. The second part consists in a coordination framework which permits reduction of deviations and inconsistencies inherent in human-centered systems.

1 INTRODUCTION

A human-centered system is one in which humans, supported by information technology, play a key role. Development and use of human-centered system involve actors, organizational structures, rules, procedures and computerized tools. Actors are humans who perform tasks in order to accomplish various categories of goals related to the business process supported by the human-centered system. Their role is characterized by three types of cooperation. On the one hand, they interact and cooperate among themselves (actor/actor cooperation). On the other hand, they interact and cooperate with the computerized tools (software, hardware, networks) which compose the human-centered system (actor/computerized tool cooperation). Finally, they interact with the organizational context composed of organization's internal and external constraints and priorities (actor/organizational context cooperation). Organizational structures, rules, procedures are instruments which permit actors either to interact among themselves or to cooperate with computerized tools and organizational context. Interactions and cooperation inherent in human-centered processes are either formal or informal. In recent years, many methods, techniques and tools have been proposed to support formal interactions. For example, workflow technology provides instruments to automate well defined sequences of actions performed by humans or machines. In particular, it automates formal procedures associated with actor/actor and actor/computerized tool cooperation. Either formal or informal interactions among actors or between actors and computerized tools may be sources of various unexpected deviations and inconsistencies. For example, since software engineering may be defined as a discipline of description (Jackson, 1995), a large number of descriptions are produced, exchanged, and used by the stakeholders of each software project according to the development process and the quality assurance standards and norms. Such descriptions include specifications, analysis and design models, program code, tests plans, schedules, change requests, process models, users manuals, style guides,… They are generally associated with formal interactions and cooperation procedures and often result in deviations and inconsistencies since establishing and maintaining consistency between these documents is difficult (Nuseibeh et al., 2000). By another way, informal interactions and cooperation either among

Toffolon C. and Dakhli S. (2004).
HUMAN-CENTERED SYSTEMS DEVELOPMENT AND USE INCONSISTENCIES.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 34-41
DOI: 10.5220/0002652500340041

actors or between actors and computerized tools are sources of various unexpected deviations and inconsistencies. Indeed, most processes governing these informal interactions cannot be completely specified in advance and once for all. A human-centered system inconsistency reflects a state of the software development or use process or a state of a software artifact resulting from this process, and generally originates from a set of deviations. Inconsistencies and deviations were defined by (Cugola et al., 1996) and (Bandinelli et al., 1994). (Fernström, 1993) have proposed formal definitions of these concepts. A wide range of deviations and inconsistencies arise during the development of human-centered systems. For example, requirements engineering generate many kinds of inconsistencies related to the multiplicity of information sources. By another way, the use of such systems often results in deviations and inconsistencies related to the gap between the cognitive, organizational and sociological processes driving the actors behaviour and the representation of these processes in the human-centered system in use. Many researchers and practitioners (Balzer, 1991) (Ghezzi et al., 1998) (Nuseibeh, 1996) have proven that deviations and inconsistencies are inevitable in particular in complex human-centered systems. Moreover, in such systems, removal of certain inconsistencies can cause others to pop up. By another way, inconsistencies may be useful for focusing on aspects of human-centered systems which need particular attention. Consequently, management of human-centered systems inconsistencies must be integrated in the human-centered systems development and use processes. We think that to be managed, deviations and inconsistencies must be identified and the issues related to understanding why they occur must be addressed. In this paper, we propose a framework which:

1. describes a typology of deviations and inconsistencies occurring during the human-centered systems development and use,
2. permits understanding the causes of human-centered systems deviations and inconsistencies,
3. may be helpful in building approaches and tools to cope with human-centered deviations and inconsistencies.

To be complete, the human-centered systems deviations and inconsistencies description and analysis must take into account firstly, the conflicting interests and points of views of all the organizational actors involved in human-centered systems development and use, and secondly all the aspects of software engineering. Therefore, our framework rests on the software engineering global model (Toffolon et al., 2002) built using the economic agency theory (Alchian et al., 1972), the transactions costs theory (Williamson, 1989) and the software dimensions theory (Toffolon, 1999). The remainder of this paper is organized as follows. In section 2, we describe synthetically the related work. Section 3 presents the software engineering global model. In section 4, we analyze the causes of deviations and inconsistencies related to human-centered systems development and use, and describe synthetically a typology of these deviations and inconsistencies. Section 5 describes a coordination model which permits reducing and managing a subset of human-centered systems deviations inconsistencies. In section 6 , we conclude this paper by listing the principal applications of the proposed framework in software engineering.

2 RELATED WORK

Many authors have analyzed inconsistencies inherent in software engineering. In the remainder of this section, we list several important research papers related to our work. (Nuseibeh et al., 2000) have argued for “making inconsistency respectable” by sometimes avoiding or ignoring it, and more often using it as “a focus for learning and a trigger for further constructive development actions”. Nevertheless, their work is dedicated to inconsistencies related to descriptions associated with software engineering. (Finkelstein et al., 1994) present a technique for inconsistency handling in the View-Points framework (Nuseibeh et al., 1992). In their turn, (Grundy et al., 1998) describe an experience with building complex multiple-view software development tools that support diverse inconsistency management facilities. (Cugola et al., 1996) and (Bandinelli et al., 1994) have provided definitions of inconsistencies and deviations inherent in software engineering while (Fernström, 1993) have proposed formal definitions of these concepts. (Balzer 1991), (Ghezzi et al., 1998), and (Nuseibeh, 1996) have proven that deviations and inconsistencies are inevitable in particular in complex human-centered systems. Despite their richness, the papers listed above do not propose any typology of deviations and inconsistencies which take into account all the aspects of software as well as the conflicting interests and points of view of stakeholders involved in software projects. Such a typology is needed to define, for each inconsistency category, the most appropriate approach to manage it. The typology we propose in this paper permits us stressing that human-centered systems deviations and inconsistencies may be partly reduced and managed through the coordination process.

3 THE SOFTWARE ENGINEERING MODEL

Applying agency theory (Alchian et al., 1972) in analyzing information technology role in modern organizations demonstrates that software engineering is governed by a set of contracts among actors concerned with the software system to be developed or maintained. At a given time, each actor plays the role of consumer (principal) or producer (agent) under the contracts which link him to the other actors. So, human-centered system development and use are a nexus of contracts among different actors with conflicting interests and points of view. The discrepancies between the actors objectives are partly the source of software engineering inconsistencies and related agency costs. By another way, we notice that well established software development methodologies make a confusion between four businesses: the customer's business, the end user's business, the developer's business and the architect's business. To eliminate this confusion, we use the transaction costs theory (Williamson, 1989) to identify four different spaces representing respectively these four businesses:

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.

Besides, each software project is represented in the four spaces by a static part, a dynamic part and actors. In each space, project's dynamic part relates to the software engineering process, project's static part is composed of software artifacts resulting from this process, while project actors are human resources concerned with this project. Each actor may have two categories of roles: producer (agent) or consumer (principal) of software artifacts. A role played by a project's actor in one of the four spaces is either principal or secondary. In each space, it is possible that a project has many actors assuming secondary roles, but there can be only one project 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).

We identify four actor's type: Customer (C), Architect (A), Developer (D), User (U). Each actor play different role in each space (Toffolon et al., 2002). An actor type is a concept that provides the specification of basic actors. Each actor type is associated with one business. In order to simplify the framework presentation, we assume that each basic actor belongs to only one actor type. For example, a maintainer of a software system in the construction space is a basic actor which belongs to the developer actor type.

The software dimensions theory (Toffolon, 1999) identifies ten dimensions which permit taking into account all the aspects of the software as well as all the conflicting interests and points of view of the project actors. Those ten dimensions concern altogether the software process and the artifacts produced by this process. The process' dimensions (cost dimension, delay dimension, technical dimension, communication dimension and organizational dimension) and the product's dimensions (functional dimension, human dimension, economic dimension, organizational dimension and temporal dimension) demonstrate that a same software may reflect many different realities which depend on the organizational, social and economic contexts of its use and exploitation. Each project space is associated with a subset of the ten software (Toffolon et al., 2002).

The correspondence between the four spaces originates from the iterative progress of the software development 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

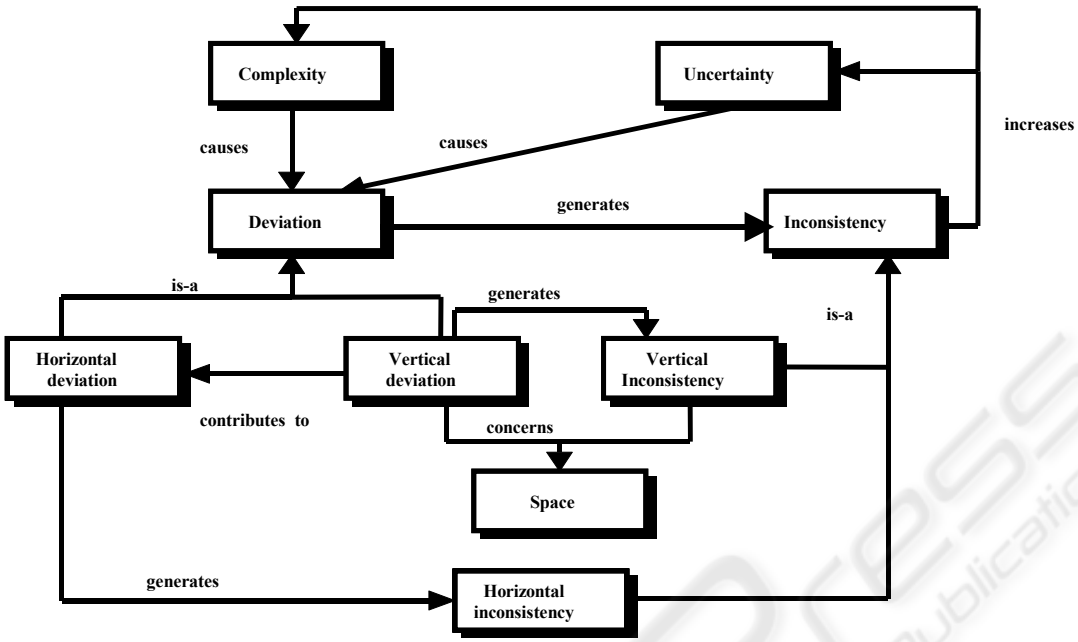

Figure 1: Software Engineering Inconsistencies Metamodel

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.

4 THE SOFTWARE ENGINEERING INCONSISTENCIES

In this work, we use the four project spaces to give more detailed definitions of software engineering deviations and inconsistencies by introducing the concepts of intra-space and inter-spaces deviations and inconsistencies (Toffolon et al., 1998). An intra-space (vertical) deviation is an event which causes a discrepancy either between the real and anticipated behavior of the process supporting one of the four project spaces, or between the real and anticipated behavior of a software artifact resulting from this process. An intra-space (vertical) inconsistency may be generated by an intra-space deviation and describes either the state of the process supporting one of the four project spaces, or the state of a software artifact resulting from this process. In the same way, an inter-spaces (horizontal) deviation is an event which disturbs interactions between two spaces. An inter-spaces (horizontal) inconsistency may be generated by an inter-spaces deviation, and describes either the state of inter-spaces interactions or the state of software artifacts concerned with these interactions. Vertical deviations and inconsistencies are related; on the one hand, to the activities of processes supporting the four project spaces and software artifacts they build; and on the other hand, to the communication problems associated with these activities. Horizontal deviations and inconsistencies depend on vertical deviations and inconsistencies which can worsen them since the software artifacts resulting from the process supporting a given space are used in the information flows exchanged between this space and the three other project spaces. Vertical and horizontal deviations and inconsistencies are interdependent since, in each project space, the actor who plays the principal role is at the same time producer of software artifacts and consumer of artifacts coming from the three other project spaces.

Consequently, the software engineering deviations and inconsistencies constitute a spiral whose progression can be compared with a succession of chain reactions which explain the iterative character of the software development process, and the inadequacy of the conventional lifecycle. Figure 1 illustrates the software engineering inconsistencies metamodel.

The definitions provided above give only a general view of software engineering deviations and inconsistencies. In particular, to cope with human-centered systems inconsistencies, we must take into account aspects related to human actors involved in such systems development and use. The typology we

propose in this work aims to bridge the gap between on the one hand, human-centered systems deviations and inconsistencies and on the other hand, tools built to reduce their impacts. In that way, we consider that deviations and inconsistencies of human-centered systems are either formal or informal. Formal deviations (vs. inconsistencies) occur if a formal procedures or rules related to software processes or software artifacts are broken. Informal deviations (vs. inconsistencies) are related either to informal aspects of software development and use processes, or to software artifacts. The distinction between formal and informal deviations (vs. inconsistencies) is important since tools which permit coping with them are different. Indeed, it is possible to reduce formal deviations (vs. inconsistencies) impacts by defining tools based on existing formal procedures. By another way, tools needed to cope with informal deviations (vs. inconsistencies) are dependant on many factors like the nature of the deviation, the organization's maturity and the existing communication and coordination know-how. Consequently the *degree of formalism* is the first facet of the typology we propose. We note that we use the expression " *degree of formalism* " to stress that each human-centered activity is a mix composed of formal and informal tasks.

Secondly, we focus on the type of cooperation where deviations and inconsistencies occur. So, we distinguish the actor/actor and the actor/tool deviations (vs. inconsistencies). The first category splits into communications and coordination deviations (vs. inconsistencies) while the second is composed of actor/tool and actor/context deviations (vs. inconsistencies). Actor/tool deviations and inconsistencies originate notably from the inadequacy of the computerized tool with the actor cognitive process. They may result in inefficient contribution of actors to the organization's business processes. We note that the analysis of the tool/tool deviations (vs. inconsistencies) is beyond the scope of this paper which focus on the human aspects of human-centered systems deviations and inconsistencies. Consequently, the *nature of cooperation* is the second facet of the proposed typology.

The third facet of this typology, called "*localization*", relates to the project spaces where the deviations (vs. inconsistencies) occur. This facet reflects the vertical (vs. horizontal) nature of human-centered deviations and inconsistencies.

The second and third facets play a critical role in determining the appropriate tools to reduce deviations and inconsistencies impacts. Indeed, they permit taking into account the characteristics of the software project spaces as well as the actors and processes concerned with the deviations and inconsistencies. In addition to actors directly involved in their development and use, human-centered systems interact indirectly with other organizational actors while cooperating with the organizational context and the external environment. Deviations and inconsistencies related to computerized tool/organizational context cooperation reflect notably human-centered system inadequacy with the organization's structure. Such deviations and inconsistencies are generally difficult to detect and their impacts are observable notably through the organization's business processes pitfalls. Because of the economic, organizational and social importance of these impacts, we consider that the *organizational aspects* constitute the fourth facet of the deviations and inconsistencies typology we propose in this work. This typology is composed of three abstraction levels: a conceptual level, a detailed level and a technical level. At the conceptual level, a deviation (vs. inconsistency) of a human-centered systems may be analyzed on the basis on the four facets described above i.e. (*degree of formalism, nature of cooperation, localization, organizational aspects*). The detailed level permits describing the characteristics of a given deviation (vs. inconsistency) four facets. This description includes notably the actors, the artifacts, the processes, the rules and the spaces concerned with a given deviation (vs. inconsistency). At the technical level, techniques, methods and tools which permit coping with deviations and inconsistencies are described. The next section provides a synthetic description of a coordination model which permits reduction of human-centered systems deviations and inconsistencies which result from actor/actor interactions and cooperation.

5 THE COORDINATION FRAMEWORK

As stressed above, human-centered actor/actor deviations and inconsistencies associated with actor/actor interactions and cooperation are either formal or informal. For example, deviations and inconsistencies related to descriptions (Nuseibeh et al., 2000) belong to this category since descriptions are produced, used, and exchanged by the software project stakeholders. They may be formal if such descriptions are associated with formal methodology or quality assurance procedures. Nevertheless, such formal deviations and inconsistencies are often associated with informal deviations and inconsistencies resulting notably from misinterpretations of terms and ideas used by descriptions issued from the software development

process. In particular, the degree of precision and formality of a description is variable, and depend upon the goals and the constraints of the software development process phases and activities. Secondly, descriptions associated with this process are often characterised by a high degree of volatility. Finally, since software descriptions are undertaken by humans actors, they may be ill-formed or self-contradictory. By another way, various informal deviations and inconsistencies may result from informal interactions between the software project stakeholders during the software project lifecycle. Such interactions are among the main characteristics of many iterative software development lifecycles like the spiral model (Boehm, 1988). There are two categories of actor/actor interactions and cooperation: the producer/consumer cooperation, and the sharing resources cooperation. The producer/consumer cooperation occurs during the realization of a contract between two stakeholders. The sharing resources cooperation occurs when two stakeholders use common resources (software, tools, procedures, standards,…) while carrying out the software engineering process tasks. For example, software developers share the same software engineering environment, the same human resource may be at the same time software designer and software programmer within the same software project. By another way, producer/consumer relationship between two project actors belonging to the same project space means that artifacts produced by one project actor are consumed by another project actor belonging to the same category. For example, there is a producer/consumer relationship between the software programmer and the software designer who belong to the developer category. Indeed, to carry out its coding tasks, the programmer consumes design artifacts built by the designer. Consequently, actor/actor deviations (vs. inconsistencies) category splits into two sub-categories: producer/consumer deviations (vs. inconsistencies) and sharing resources deviations (vs. inconsistencies). The actor/actor deviations and inconsistencies may have important negative impacts on the software development process and on the software system issued from it. Therefore, a coordination process is needed in order to reduce the impacts of formal and informal deviations and inconsistencies inherent in actor/actor cooperation. The coordination process in software engineering permits managing dependencies between the stakeholders involved in the software development and maintenance processes. The coordination process rests on formal and informal organizational models which determine the distribution of roles among interdependent stakeholders. So, it aims to answer the question: who do what? when? According to the global software model, since dependencies between stakeholders result in contracts, the coordination process in software engineering describes the formal and informal organizational procedures needed to carry out these contracts. Each organizational procedure points to one or many operational procedures. An operational procedure is a set of ordered activities associated with a stakeholder's role. In other words, an operational procedure describes the concrete actions undertaken by a stakeholder involved in a contract. To manage vertical and horizontal dependencies, we propose a coordination process composed of two sub-processes. The vertical coordination sub-processes aims at managing vertical dependencies while the horizontal coordination sub-process permits managing horizontal dependencies. Each coordination sub-process is composed of two layers. The first layer relates to the informal coordination activities while the second layer corresponds to the formal coordination activities which are supported by formal organizational procedures stored in a repository, called the coordination repository. Since the coordination process in software engineering supports the software development and maintenance process, it must be integrated to this process. So, the software engineering coordination process depends on the lifecycle model of the software development and maintenance process it supports. Besides, according to the software engineering model presented above and the software engineering process described in (Ghezzi et al., 1998), the software engineering coordination process lifecycle is based on five spirals: four vertical spirals and one horizontal spiral. The horizontal coordination spiral supports the horizontal coordination sub-process and permits communication, artifacts exchange and navigation between the four project spaces. The four vertical coordination spirals, which support the four vertical coordination sub-processes associated with the four project spaces, are:

- The problem coordination spiral associated with the problem spiral which supports the problem space,
- The architecture coordination spiral associated with the architecture spiral which supports the solution space,
- The construction coordination spiral associated with the construction spiral which supports the construction space,
- The operation coordination spiral associated with the problem spiral which supports the operation space.

The five coordination spirals are interdependent. So, representing these processes using a lifecycle based on the spiral model is a difficult task. Firstly, each coordination spiral has two aspects: formal and

informal. Secondly, coordination tasks and software development tasks are intertwining. Thirdly, the horizontal coordination process between two project spaces is preceded by a nexus of software development and vertical coordination tasks and generates new coordination and development tasks taking place in the two spaces. Finally, since the software development process is iterative, the coordination processes supporting this process are also iterative. So, a clear representation of the software engineering coordination process requires the description of the meta-lifecycle supporting the five coordination spirals.

The horizontal and vertical coordination sub-processes rest on a meta-life cycle called **(INFO)** composed of four phases: Initialization, Negotiation, Formalization an Operation. The Initialization and the Operation phases permit shifting between the coordination activities and the software engineering activities while the Negotiation and the Formalization phases support the two layers of the coordination horizontal and vertical sub-processes.

The Initialization phase

The Initialization phase consists in identifying the scope of coordination problem to be solved

The Negotiation phase

In our framework, the first layer of each coordination sub-process is supported by a Negotiation phase. It consists to cope with coordination topics which are not carried out by the formal procedures stored in the coordination repository. The Negotiation phase consists in eliminating conflicts and discrepancies between producers and consumers of artifacts needed to build a software system.

The Formalization phase

The formalization phase supports the formal layer of the coordination sub-process. This phase rests on formal organizational procedures needed to carry out contracts among project actors.

The Operation phase

The Operation phase links the coordination activities to the software engineering activities. It rests on the operational procedures associated with the formal organizational procedures supporting the coordination sub-process layer.

6 CONCLUSION

The framework we describe in this paper provides basic instruments to cope with deviations and inconsistencies of human-centered systems. In particular, the proposed typology permits defining methods and tools to manage inconsistencies in compliance with the organization's constraints, priorities and technical maturity. So, it is compliant with the Simon's Bounded Rationality Principle (Simon, 1983). By another way, the typology of inconsistencies and deviations proposed in this work permits identifying other types of deviations and inconsistencies not considered in the literature. These deviations (vs. inconsistencies) must be managed since they are sources of uncertainty and are not compliant with many software quality principles like traceability and continuous improvement of the software development process. A future research direction consists on the one hand, to propose a more formal description of the proposed typology of human-centered deviations and inconsistencies, and on the other hand, to use the proposed typology to build management and reduction approaches which take into account all the facets of human-centered deviations and inconsistencies. Since it may be beneficial to avoid or defer the elimination of some deviations and inconsistencies, such approaches should include evaluation instruments of the cost and the outcome of reduction and management of deviations and inconsistencies.

The use of the coordination process to cope with the actor/actor inconsistencies and deviations associated with human-centered systems permits us identifying many practical problems. Firstly, a formal procedure language is needed. The main advantage of such a language consists in minimizing the number and the negative effects of deviations and inconsistencies resulting from procedure misinterpretations , and on the other hand, it improves the transition between software engineering activities and coordination activities by mapping organizational procedures and operational procedures. Finally, in order to optimize the outcome of coordination process in deviations and inconsistencies management and reduction, the transition from the Negotiation phase to the Formalization phase need to be described formally. Indeed, this transition is related to procedure reuse, procedure creation and procedure configuration management.

REFERENCES

Alchian A.A. and Demsetz H., 1972, "*Production, Information Costs and Economic Organization*", American Economic Review, Vol. 62, No.5, pp. 777-795.

Balzer R., 1991, "*Tolerating Inconsistencies*", in Proceedings of ICSE'13 Conference, Austin, Texas, USA, May 13-17, 1991, pp. 158-165.

Bandinelli S., Di Nitto E. and Fuggetta A., 1994, "*Policies and Mechanisms to Support Process Evolution in PSEE*", In Proceedings of the 3rd IEEE International Conference on the Software Process, Reston, VA.

Boehm B.W., 1988, " A Spiral Model of Software Development and Enhancement ", Computer, Vol. 21, No.5, pp. 61-72.

Cugola G., Di Nitto E., Fuggetta A. and Ghezzi C., 1996, "*A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems*", ACM Transactions on Software Engineering and Methodology, Vol. 5, No.3, pp. 191-230.

Fernström C., 1993, "*State Models and Protocols in Process-centered Environment*", in Proceedings of the 8th ACM International Software Process Workshop, Wadern, Germany.

Finkelstein A., Gabbay D., Kramer J. and Nuseibeh B., 1994, "*Inconsistency Handling in Multi-Perspective Specifications*", IEEE Transactions on Software Engineering, Vol. 20, N°8, pp. 569-578.

Ghezzi C. and Nuseibeh B., 1998, "*Managing Inconsistencies in Software Development*", Guest Editorial, Transactions on Software Engineering, Vol. 24, No.11, pp. 906-907.

Grundy J.C., Hosking J.G. and Mugridge W.B., 1998, "*Inconsistency Management for Multi-view Software Development Environments*", IEEE Transactions on Software Engineering: Special Issue on Managing Inconsistency in Software Development, Vol. 24, No.11.

Jackson M., 1995, "*Software Requirements & Specifications : a lexicon of practice, principles and prejudices*", Addison-Wesley.

Nuseibeh B. and Finkelstein A., 1992, "*View-Points: A Vehicle for Method and Tool Integration*", Proceedings of the 5th workshop on Computer-Aided Software Engineering (CASE'92), July 6-10th, 1996, Montreal, Canada, IEEE Computer Society Press, pp. 50-60.

Nuseibeh B., 1996, "*To Be or Not To Be: On Managing Inconsistency in Software Developmen*t", in the Proceedings of the 8th International Workshop on Software Specifications and Design (IWSSD'8), Schloss, Velen, Germany, pp. 164-169.

Nuseibeh B., Easterbrook E. and Russo A., 2000, "*Leveraging Inconsistency in Software Development*", Computer, Vol. 33, No. 4, pp. 24-29.

Simon H.A., 1983, "*Models of Bounded Rationality*", (2 volumes), MIT Press, Cambridge.

Toffolon C., 1999, "*The Software Dimensions Theory*", in the Proceedings of ICEIS'99 Conference, Setubal, Portugal, published by KLUWER ACADEMIC PUBLISHERS in "Enterprise Information Systems", Selected Papers Book, Joaquim Filipe (Ed.).

Toffolon C. and Dakhli S., 1998, "*A Framework for Software Engineering Inconsistencies Analysis and Reduction*", In the Proceedings of the 22nd Annual International Computer Software & Applications Conference (IEEE-COMPSAC'98), Vienna, Austria, August 1998, IEEE Computer Society Press, pp. 220-227.

Toffolon C. and Dakhli S., 2002, "The Software Engineering Global Model", in the Proceedings of the 20th Annual International Computer Software and Application Conference (IEEE-COMPSAC'02), Oxford, England, August 26-29, 2002, IEEE Computer Society Press.

Williamson O.E., 1989, "*Transaction Cost Economics*", in "R. Schmalensee and R. Willig (Eds): Handbook of industrial organization", Vol. 1, North-Holland.