8-2010

Evaluation of the Goal-Oriented Requirements Engineering Method KAOS

Frank Zickert
University of Frankfurt, mail@frankzickert.de

Evaluation of the Goal-Oriented Requirements Engineering Method KAOS

Frank Zickert
University of Frankfurt
mail@frankzickert.de

Roman Beck
University of Frankfurt
rbeck@wiwi.uni-frankfurt.de

ABSTRACT

Software engineering is a complex task. But although there is no silver bullet that guarantees accomplishing this task, appropriate methods can support the engineer by addressing the characteristics that make it complex. The objective of this paper is to evaluate whether and how the goal-oriented requirements engineering method KAOS addresses these characteristics of complex tasks and thereby, whether it effectively supports software engineering. For serving this purpose, we conduct a literature analysis, which discloses core concepts underlying to the KAOS method, and we apply KAOS in two software development projects, which provide insights into KAOS in use. Our results show that KAOS, despite of some shortcomings, addresses all characteristics, but that applying it can be work intensive. Consequently, while KAOS supports software engineering, provided support must be weigh up against invested work.

Keywords (Required)

KAOS, system engineering, task complexity.

INTRODUCTION

Software engineering is a complex problem (Benbya and McKelvey, 2006) in which "the hardest single part … is deciding precisely what to build. … No other part of the work so cripples the resulting [software] system if it is done wrong. No other part is more difficult to rectify later" (Brooks, 1986). But although there is no silver bullet for software engineering, methods which support the creation of software have improved in the past (Berry, 2002).

The purpose of this paper is to evaluate whether and how the goal-oriented requirements engineering (RE) method KAOS supports software engineering. For serving this purpose, we derive propositions from the perspective of software engineering as complex problem and evaluate them using two software development projects within the financial industry.

This evaluation of KAOS contributes to researchers in RE and practitioners interested in using KAOS. To researchers in RE, we point out with regard to which characteristics of complex problems KAOS supports software engineering and with regard to which it requires improvement. To practitioners, who are interested in using KAOS, insights into its effectiveness provide information on what to expect from this method and what not to expect.

The remainder of this paper is as follows. In section two, we briefly present the KAOS method. After that, we depict our literature analysis (section three) and derive our propositions (section four). In section five, we describe the empirical case studies, which serve as basis to evaluate KAOS (section six). Finally, we present our conclusions in section seven.

REQUIREMENTS ENGINEERING USING KAOS

KAOS builds upon the notion of goal orientation (van Lamsweerde, 2004) that is a major stream in the field of RE (Anwer and Ikram, 2006). This stream builds upon the premise that "in designing software systems, requirement engineers aim to 'improve' organisational situations which are seen as problematic" (Kavakli and Loucopoulos, 2003). Goals address these problems and thereby, set the objectives for changes (Loucopoulos and Kavakli 1995). While goal orientation completely covers the process building software, KAOS is one of the most important methods, whose focal points are supporting requirement acquisition (Dardenne, van Lamsweerde and Fickas, 1993), specification (Darimont and van Lamsweerde, 1996), and verification (Ponsard, Massonet, Rifaut, Molderez, van Lamsweerde and Tran Van, 2007).

In KAOS, the software under construction is captured as an instance of a conceptual meta-model where abstractions such as goals, requirements, operations, agents, or entities are semantically linked (van Lamsweerde, Darimont and Massonet, 1995). Moreover, KAOS provides a formal assertion layer (Darimont and van Lamsweerde, 1996) for inferring specifications from

requirements (van Lamsweerde and Willemet, 1998) and reasoning about goal satisfaction (Letier and van Lamsweerde, 2004).

The conceptual meta-model of KAOS comprises four models that are iteratively prepared: (1) a goal model in which goals to be achieved by the software are described; (2) an object model in which objects involved in the software are described; (3) an agent model in which responsibilities are assigned to agents; and (4) an operation model in which input-output relationships among operationalizations of requirements and identified objects are described (Letier and van Lamsweerde, 2002b; van Lamsweerde 2001).

Firstly, goals are acquired by incrementally asking "how" and "why" questions (van Lamsweerde, 2000; Yu and Mylopoulos, 1998). They are captured in the goal model that is represented as hierarchical graph using AND/OR-decompositions of discrete high-level goals down to precise leaf-level requirements (van Lamsweerde et al., 1995). Figure 1 provides an exemplary goal model in KAOS.

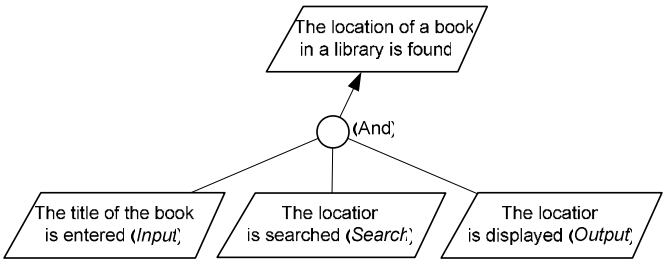

Figure 1. KAOS goal model

After the goals are acquired, the object and the agent models are prepared. The object model collects objects, attributes, and relationships among them while in the agent model, agents are assigned responsibility for achieving the goals (Letier and van Lamsweerde, 2002a). Agents and objects are used when the operation model is prepared. In this last step, operations (represented as ovals) that describe the behavior of the system in specific situations are derived from requirements (van Lamsweerde and Willemet, 1998). Situations that operations work in are determined by events (arrowed rectangle) that cause the operation and entities (rectangles) that serve as information input. Each operation must be performed by an agent (hexagon). Figure 2 provides an example of an operation model.

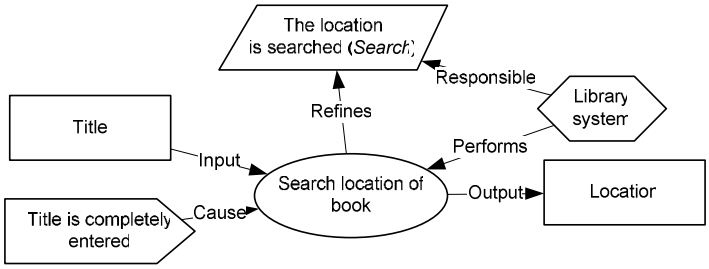

Figure 2. KAOS operation model

When KAOS models are prepared, the formal assertion layer that is written in a real-time temporal logical way (Koymans, 1992) can be used to formally assure that goals are correctly refined into sub-goals, requirements, and operations. For instance, the goal of *the location of a book in a library is found* can be defined as follows:

Goal Achieve[*the location of a book in a library is found*]
Concerns *Title*, *Location*
RefinedTo *Input, Search, Output*
InformalDefinition The location of a book is found if it is displayed
FormalDefinition $\forall$ t: *Title*, l: *Location*: *Output* (l) $\wedge$ *SearchedTitle*(t) $\wedge$ *IsLocationOfBook*(l, t) $\Rightarrow$ o *BookFound*(t)

The goal definition describes what must be done to achieve the goal. The used operator o is a classical operator for temporal referencing and means "in the next state" (see Manna and Pnuel (1992) for an in depth description of temporal logic). Thus, *finding the location of a book in a library* is achieved, if a location (l) is output that is the location of the book with the searched title (t).

LITERATURE ANALYSIS

The KAOS method as presented was not spontaneously developed, but evolved as the result of research conducted over a period of more than 10 years now. Therefore, we claim that modeling concepts were not randomly added but included for the purpose of supporting engineers in building software. This literature analysis aims at disclosing these modeling concepts.

We conducted the literature analysis by following the framework of Levy and Ellis (2006). Search for relevant literature involved three steps, a keyword search, a backward search, and a forward search. We started with a keyword search on the term "KAOS" in three major literature databases ACM Digital Library, IEEE Xplore, and Elsevier (ScienceDirect). Based on title, abstract, and keywords of the results, we decided on whether we found the source relevant for KAOS. A thereon based backward search aimed at disclosing the origin of KAOS (Webster and Watson, 2002). We found 75 publications that directly related to the KAOS method. The subsequent forward search builds upon the five most cited publications, which we chose, since there is a gap between #5 (324 citations) and subsequently ranked articles (#6: 247 citations, #7: 220 citations) as depicted in table 1. We found additional 13 publications with KAOS as their major topic. We conducted another backward search to assure that we have not missed any major source or concurrent stream of KAOS-related research. Since we did not find any new major author or publication, we consider having reached an appropriate stopping point for our literature analysis (Leedy and Ormrod, 2005).

Authors	Year	Title	Citations	Rank
Dardenne, van Lamsweerde and Fickas	1993	Goal-directed requirements acquisition	1231	1
van Lamsweerde	2001	Goal-Oriented Requirements Engineering: A Guided Tour	759	2
van Lamsweerde	2000	Requirements engineering in the year 00: a research perspective	417	3
van Lamsweerde, Darimont and Letier.	1998	Managing Conflicts in Goal-driven Requirements Engineering	387	4
van Lamsweerde and Letier	2000	Handling Obstacles in Goal-Oriented Requirements Engineering	324	5
Darimont and van Lamsweerde	1996	Formal refinement patterns for goal-driven requirements elaboration	247	6

Table 1. Major KAOS sources (based on citation count)

We analyzed the literature with regard to which modeling concepts of the KAOS method they describe. Repeated reference of the same keywords in the literature, such as *refinement* or *verification*, allowed identification of the most important concepts. Moreover, we recognized similarities among given reasons on why different concepts were included in KAOS.

In papers, which concerned topics such as refinement, patterns, or derivation of requirements from goals, the question of interest mainly is how to refine goals into requirements and operations. These papers aim at guiding the engineer on coping with decomposition of a high-level goal into more detailed requirements and operations.

Similarities were also found among papers, which for example mention verification, validation, or scenarios. These do not answer the question of how a goal can be decomposed but they provide techniques for reasoning about whether decompositions satisfy superordinate goals. Most of these papers employ the formal method of KAOS including temporal logic.

The third identified class comprises papers mentioning keywords, such as inconsistencies, obstacles, or conflicts. These are concerned with providing support on how to cope with situations when refinements do not satisfy superordinate goals, because they conflict with satisfaction of other goals.

Table 2 illustrates these three classes of the KAOS method.

Class	Modeling concepts
Hierarchical decomposition	Refinement, patterns, architecture, operationalization, derivation, acquisition,
Goal-based reasoning	Verification, validation, interaction analysis, formal method, correctness, scenario, temporal logic, animation, model checking, runtime behavior
Obstacle management	Inconsistency, obstacle, conflict, divergence

Table 2. Core concepts of the KAOS method.

THEORETICAL PERSPECTIVE

This evaluation of KAOS is based on whether the identified modeling concepts are appropriate for coping with the complex problem of software engineering. The theoretical context for this evaluation is given by the perspective of task complexity of which Gill and Hicks (2006) provide a comprehensive literature review.

Complex tasks exhibit specific characteristics that make them complex. According to Campbell (1988) these are (a) the presence of multiple ways of solving the problem, (b) the presence of uncertain links among ways and outcomes (c), the presence of multiple desired outcomes that must be achieved, and (c) the presence of conflicting interdependence among the outcomes. Table 3 summarizes these characteristics of task complexity. For successfully performing complex tasks, the task-doer has to cope with these characteristics. Although using a tool cannot change these characteristics, which are inherently associated with the given task, it can dramatically impact the performance when the task is carried out by the task-doer (Gill and Hicks, 2006). Therefore, by appropriately addressing these characteristics, a method can support the task-doer to perform the task effectively.

Complexity characteristics	Short description
Multiple ways	In complex tasks, there is not a singular way of how the task must be done, but there are multiple ways, which may or may not result in the desired outcome. Multiple ways increase complexity, since the task-doer must decide, which way to follow.
Uncertain linkages	A task is complex, if the linkage between a way and its outcome is ambiguous. As a result, the way of achieving the goal cannot be planned upfront, because information does not become available until the task is performed.
Multiple outcomes	Complex tasks have multiple outcomes that require attention. Typically, each outcome entails a separate information processing stream, which increases the amount of information that has to be processed concurrently.
Conflicting interdependence among outcomes	A task is complex if achieving one desired outcome conflicts with achieving another desired outcome. As a consequence, ways that achieve both outcomes are less obvious and thus more difficult to find.

Table 3. Complexity drivers (according to Campbell 1988)

KAOS addresses the complexity characteristic of multiple ways by employing the concept of problem decomposition (Simon, 1996) for refining discrete high-level goals down to precise leaf-level requirements (van Lamsweerde et al., 1995). Decomposition is a suited technique for reducing the dimensions of a problem and thereby decreasing the number of ways that have to be considered (Marengoa and Dosi, 2005). For example, refinement of the goal of *finding the location of a book* into *input*, *search*, and *output* as shown in figure 1 allows the engineer to separately concentrate on satisfying each sub-goal while excluding the dimensions related to the other sub-goals. When designing the *input* interface, the engineer does not need to consider how the *search* works. However, when refining goals, it has to be verified that the composition of sub-goals satisfies the higher-level goal (Dardenne, Fickas and van Lamsweerde, 1991; De Landtsheer, Letier and van Lamsweerde, 2004). Therefore, we state proposition 1: KAOS addresses the complexity characteristic of multiple ways.

Goal-based reasoning addresses the complexity characteristic of uncertain linkages, since it provides information on the outcome of respective goal refinements. This information is valuable for decisions on whether to continue with the current refinement or to select another one, which might better satisfy the higher-level goal. Since verification using the formal assertion layer is work intensive, KAOS provides refinement patterns that can be applied to refine specific types of goals, such as *Achieve X*, or *Maintain X* (Darimont and van Lamsweerde, 1996; Letier and van Lamsweerde, 2002b). Patterns are pre-verified refinement strategies, whose usage represents an economic way of assuring that the refinement of a goal is valid. Therefore, we state proposition 2: KAOS addresses the complexity characteristic of uncertain linkages.

To some extent, goal-based reasoning also addresses the complexity characteristic of multiple outcomes. Since a goal model is not limited to a singular top-level goal, it can be verified whether decomposed sub-goals satisfy multiple goals. However, since satisfaction of each goal must be separately verified, resulting work is even higher. Proposition 3: KAOS addresses the complexity characteristic of multiple outcomes.

The KAOS method addresses the complexity characteristic of conflicting interdependence among tasks by explicitly including obstacles into the models (van Lamsweerde, Darimont and Letier, 1998; van Lamsweerde and Letier, 1998). Thereby, inconsistencies and conflicts can be made explicit and the engineer can verify potential resolutions of the obstacle (van Lamsweerde and Letier, 2000). However, applying the KAOS method does not prevent from inconsistencies. In fact, systematic techniques are requested that generate better designs which do not end up in inconsistencies (Letier and van Lamsweerde, 2004). Nevertheless, we state proposition 4: KAOS addresses the complexity characteristic of conflicting interdependence among outcomes

CASE STUDIES

Our empirical evaluation of KAOS builds upon two case studies, which we conducted in software development projects within the information technology division of a large financial institution. We followed the guidelines of Kitchenham and Pickard (1995) for conducting multiple case research for method and tool evaluation. In the cases, we prepared KAOS models concurrently to the requirement engineer, who did not use a formal RE method but adhered to the default process used in the institution. This process generally followed the waterfall model (Royce 1970). Based on functions as requested by the internal customer, that is the Germany-based retail customer division, the engineer elaborated requirements and derived design specifications. Both, requirements and specifications were written in natural language and collected in respective documents. Besides the engineer, other stakeholders, such as project manager, technical specialists, and test analysts were involved, who reviewed and commented the documents.

Data was collected as follows. While the engineer, who was the same in both projects, analyzed the problem and designed the software, we prepared KAOS models based on the same information that was available to the engineer. While the engineer already worked in this environment for several years, one researcher served as on-site expert and was already involved in several other projects within this environment. Information parity in the projects was ensured by interrogations with the engineer, attendance at meetings, access to e-mails, and access to documents. Despite of situations, in which major decisions about the design were made, insights we gathered from the KAOS models were not returned to the engineer. These situations were particularly beneficial for the analysis, since they disclosed whether the engineer had any problems in designing the software, which could have been mitigated by using KAOS. Information exchange started with an inquiry of the engineer's opinion on the situation, any preferences with regard to the system design, and the suggested next steps. After having recorded this data, we returned information provided by the KAOS models. We particularly paid attention to gaps between the exchanged information and whether information we returned led to any change of the engineer's mind regarding the previously inquired opinion, preferences, or suggestion. These gaps served as basis for identifying differences between the default case and application of the KAOS method.

In the first project, an existing front-end system was integrated with a recently built payment processing system. Before, payment order entering was already offered at the front-end, but these orders were transferred to a legacy processing system. The reason for this re-engineering project was the intended deactivation of the legacy system.

There were two important situations in this project, in which stakeholders controversially discussed which of alternative system designs had to be used. First, the engineer designed a web-service interface between the front-end and the processing system, whereas the technical analyst responsible for the processing system insisted on using a routing system. Second, a problem occurred about how to check bank codes for correctness already on the front-end. While the engineer favored download of bank code data, the vendor responsible for the development suggested an online interface to check bank codes on demand.

In both situations, decision documents prepared by the engineer were simple Power Point slides with yes/no statements about goal satisfaction, whereas the KAOS models provided logical reason whether goals about the software under construction were satisfied in the discussed alternatives. In the first situation, evaluation of goal satisfaction using KAOS matched with the engineer's perception for all four discussed goals. In the second situation, the engineer was doubtful about satisfaction of one out of three goals, whereas the KAOS models provided clear information on satisfaction of all goals.

However, in both situations, not only goals concerning the software under construction, such as functional or architectural goals, were discussed, but also resulting project effort was an important factor. In fact, project effort was the decisive factor in the first situation, in which an architectural goal was consciously left unsatisfied, because it would have generated too

much effort. While the engineer was able to roughly estimate resulting effort, the KAOS models did not provide this information.

In the second project, a server system had to be extended with a function that allows the selective deactivation of connected terminals. Since these terminals are programmed for highest possible availability, they autonomously re-activate. The challenge in this project was the purposeful adjustment of the terminal re-activation routines. While it had to be assured that a terminal did not autonomously re-activate once it had been deactivated, it was as critical that reliability of the activation routines remained unaffected in default cases, in which terminals were supposed to re-activate.

The most critical situation in this project occurred, when a major flaw in an assumption about the existing re-activation routines was discovered. Up to this point, both the requirement specifications written by the engineer and the KAOS model suggested a well-formed design. But at the time when the engineer refined this assumption, it turned out that the re-activation routines worked slightly but essentially different than assumed. The KAOS models did not disclose this flaw either, since it resided inside an assumption.

After the flaw had been discovered, KAOS obstacle management techniques allowed resolving the flaw with additional adjustments at the re-activation routines. However, the engineer rejected this resolution, since associated changes would have been fundamental and thus, potentially affecting the reliability of the re-activation routines. The problem was solved by using a different approach, which involved another set of high-level requirements. Consequently, large parts of the requirements specifications and the KAOS models had to be rewritten.

RESULTS

Observations support our introductive claim that engineering software requirements is a complex problem. The engineer did not spend most of the time for proceeding in a well defined task, but rather had to search for a solution in an obscure environment, where there was no obvious way to achieve the goal. In fact, in both projects, the system could have been built in different ways, which resulted in discussions among stakeholders. While the engineer accounted for these ways by preparing decision documents, KAOS supported concurrent consideration of alternatives by using OR-relationships. But although concurrent consideration of different alternatives results in additional work for preparing the models, it addresses the complexity characteristic of multiple ways, wherefore we find support for proposition 1.

The projects also exhibited the second complexity characteristic of uncertain linkages. For instance, while the engineer was doubtful about satisfaction of a goal in the first case, the KAOS models provided clear information on satisfaction of all goals. Moreover, decision documents used in the projects were generally limited to yes/no statements, whereas KAOS models provided detailed and verified proof of goal satisfaction. Thus, we find support for proposition 2: KAOS addresses the complexity characteristic of uncertain linkages.

In our cases, multiple outcomes were requested by stakeholders. These included but were not limited to goals that the software under construction had to achieve. Stakeholders were also concerned about the resulting effort for building the software, which is reasonable, since software engineering "is not just about solving problems [but] about solving problems with economical use of resources, including money" (Shaw, 1990). KAOS models, however, represent the software under construction (van Lamsweerde, 2001) and thus are unable to represent goals about resulting effort, which is not a result of the software but a result of the process of building the software (Abdel-Hamid and Madnick, 1991). We therefore find limited support for proposition 3: KAOS addresses the complexity characteristic of multiple outcomes.

Finally, in the second case, a wrong assumption had been made in the requirements specifications, which resulted in an unworkable situation that had to be reworked when the flaw was discovered. Since the flaw resided within an assumption, it was not discovered using KAOS either. Nevertheless, KAOS obstacle management techniques helped mitigating the discovered problem and suggested a solution, which however was not pursued by the engineer. Under the premise that even KAOS models are only as good as information put into them, we find support for proposition 4: KAOS addresses the complexity characteristic of conflicting interdependence among outcomes.

We find KAOS addressing each characteristic of a complex task. However, we also identified a shortcoming of the KAOS method with regard to addressing multiple outcomes. Table 4 summarizes the results of our evaluation of whether and how KAOS addresses the characteristics of a complex task. Indeed, while KAOS addresses all characteristics, work has to be spent for using it. While preparation of the semantic models is effortless, applying the formal assertion layer is very work intensive. Moreover, when modeling obstacles, scenarios, and various alternatives, the resulting work rapidly increases, even if only semantic models are prepared.

Complexity characteristics	Modeling concepts used in KAOS	Observations in the cases
Multiple paths	- Hierarchical decomposition	- OR-relationships. Concurrent consideration and discussion of alternative solutions to the problem
Uncertain linkages	- Goal-based reasoning (verification) - Hierarchical decomposition (OR-relationships)	- Satisfaction of all goals were verified by using KAOS, whereas the engineer had no information about satisfaction of one out of seven goals
Multiple outcomes	- Goal-based reasoning (concurrent verification of high-level goals)	- Goals about the software under construction were considered - Resulting effort could not be assessed using KAOS, but it was an important factor in the cases
Conflicting interdependence among outcomes	- Obstacle management (resolving inconsistencies)	- KAOS supported finding a resolution for the wrong assumption

Table 4. How KAOS addresses complexity factors

CONCLUSIONS

This paper provides an evaluation of the RE method KAOS. It builds upon a literature analysis and two case studies in which the KAOS method is compared to not using a formal RE method. Our results show that KAOS employs useful concepts for accomplishing the complex task of software engineering, but that using this method becomes laborious if all of its provided functions are used. Thus, any improvement reducing the work effort required for preparing the models would be most beneficial to the KAOS method.

Moreover, we have identified a shortcoming of KAOS. The goal-based reasoning in KAOS is limited to goals about the software under construction. It does not account for other important factors, such as the effort required for building the engineered software. Moreover, any KAOS model is only as good as the input that is used for preparing it. Thus, although KAOS supports the engineer in solving the complex task of software engineering, it is no silver bullet whose application guarantees success. Whether and to which extent KAOS is used remains the decision of the engineer, who has to weigh up gained support against invested work.

This evaluation of KAOS contributes to researchers in RE and practitioners interested in using KAOS. To researchers in RE, we provide insights into the characteristics of complex problems that KAOS addresses and with regard to which it requires improvement. To practitioners, we provide insights into what to expect from using KAOS and what not to expect. These insights are the basis for a reasoned decision about whether to use KAOS or not.

While we consider the synthesis of both literature-based identification of the KAOS modeling concepts and empirical evaluation of their appropriateness as the major strength of this evaluation of KAOS, this work itself has some limitations. Firstly, in our literature analysis, we focused on the major concepts underlying to KAOS. While this focus discloses the core of the KAOS method, it might not account for all extensions that have recently been made to KAOS. In fact, when regarding the authors of KAOS-related literature, we see that particularly in the last few years the number of different authors rapidly increased, whereas before, KAOS was mainly addressed by the group of persons who were involved in its development. With an increasing number of researchers interested in KAOS, we expect more concepts to become integrated into it. An analysis of recent improvements of the KAOS method may disclose new fields of its application.

Secondly, our empirical analysis is limited to two software development projects within the same financial institution. There may be other problems in software engineering in other organizations in which KAOS is more or less useful. However, since we investigated software engineering in two different contexts, a multi-stakeholder re-engineering project and a single-stakeholder software enhancement project, we are confident that the empirical insights are a valuable source for this evaluation of the KAOS method.

REFERENCES

1. Abdel-Hamid, T. and Madnick S. (1991) Software project dynamics, Englewood Cliffs, Prentice-Hall.
2. Anwer, S. and Ikram, N. (2006) Goal oriented requirement engineering: A critical study of techniques, *Proceedings of the 13th Asia Pacific Software Engineering Conference*, December 6-8, Kanpur, 121-130.
3. Benbya, H. and McKelvey, B. (2006) Toward a complexity theory of information systems development, *Information Technology & People*, 19, 1, 12-34.
4. Berry, D. (2002) The inevitable pain of software development: Why there is no silver bullet, *Proceedings of the 9th International Workshop on Radical Innovations of Software and Systems Engineering in the Future*, October 7-11, Venice, Italy, 2002, 50-74.
5. Brooks, F. (1986) No silver bullet, in Kugler, H.-J. (Ed.) *Information Processing '86*, Elsevier Science Publishers.
6. Campbell, D. (1988) Task complexity: A review and analysis, *The Academy of Management Review*, 13, 1, 40-52.
7. Dardenne, A., Fickas, S. and van Lamsweerde, A. (1991) Goal-directed concept acquisition in requirements elicitation, *Proceedings of the 6th international workshop on Software specification and design*, Como, Italy, 14-21.
8. Dardenne, A., van Lamsweerde, A. and Fickas, S. (1993) Goal-directed requirements acquisition, *Science of Computer Programming*, 20, 1, 3-50.
9. Darimont, R. and van Lamsweerde, A. (1996) Formal refinement patterns for goal-driven requirements elaboration, *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, 21, 6, 179-190.
10. De Landtsheer, R., Letier, E. and van Lamsweerde, A. (2004) Deriving tabular event-based specifications from goal-oriented requirements models, *Requirements Engineering*, 9, 2, 104-120.
11. Gill, T. and Hicks, R. (2006) Task complexity and informing science: A synthesis, *Informing Science Journal*, 9, 1-30.
12. Kavakli, E. and Loucopoulos, P. (2003) Goal Driven Requirements Engineering: Evaluation of Current Methods, *Proceedings of the 8th CAiSE/IFIP Workshop on Evaluation of Modeling Methods in Systems Analysis and Design, EMMSAD 2003*.
13. Kitchenham, B. and Pickard, L. (1995) Case studies for method and tool evaluation, *IEEE Software*, 12, 4, 52-62.
14. Koymans, R. (1992) Specifying message passing and time-critical systems with temporal logic, *Lecture Notes in Computer Science*, 651, Springer-Verlag, Berlin, Germany.
15. Leedy, P., and Ormrod, J. (2005) Practical research: Planning and design, Prentice Hall, Upper Saddle River, NJ.
16. Letier, E. and van Lamsweerde, A. (2002a) Agent-Based Tactics for Goal-Oriented Requirements Elaboration, *Proceedings of the 24th international conference on Software Engineering*, Orlando, Florida, 83-93.
17. Letier, E. and van Lamsweerde, A. (2002b) Deriving operational software specifications from system goals, *SIGSOFT Software Engineering Notes*, 27, 6, 119-128.
18. Letier, E. and van Lamsweerde, A. (2004) Reasoning about partial goal satisfaction for requirements and design engineering, *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, Newport Beach, CA, 53-62.
19. Levy, Y. and Ellis, T. (2006) A systems approach to conduct an effective literature review, *Support of Information Systems Research*, 9, 181-212.
20. Loucopoulos, P. and Kavakli, E. (1995) Enterprise Modelling and the Teleological Approach to Requirements Engineering, *International Journal of Intelligent and Cooperative Information Systems*, 4, 45-79.
21. Manna, Z. and Pnuel, A. (1992) The temporal logic of reactive and concurrent systems, Springer-Verlag, Berlin, Germany.
22. Marengoa, L. and Dosi, G. (2005) Division of labor, organizational coordination and market mechanisms in collective problem-solving, *Journal of Economic Behavior & Organization*, 58, 303-326.
23. Ponsard, C., Massonet, P., Rifaut, A., Molderez, J., van Lamsweerde, A. and Tran Van, H. (2007) Early verification and validation of mission critical systems, *Formal Methods in System Design*, 30, 3, 233-247.
24. Royce, W. (1970) Managing the development of large software systems, *Proceedings of IEEE Wescon,* 1-9.
25. Shaw, M. (1990) Prospect for an engineering discipline of software, *IEEE Software*, 7, 15-24.
26. Simon, H. (1996) The sciences of the artificial, MIT Press, Cambridge, MA.

27. van Lamsweerde, A. (2000) Requirements engineering in the year 00: a research perspective, *Proceedings of the 22nd international conference on Software Engineering*, Limerick, Ireland, 5-19.
28. van Lamsweerde, A. (2001) Goal-oriented requirements engineering: A guided tour, *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, 249-263.
29. van Lamsweerde, A. (2004) Goal-oriented requirements engineering: A roundtrip from research to practice, *12th IEEE International Requirements Engineering Conference*, September 6-10, Kyoto, Japan, 4-7.
30. van Lamsweerde, A., Darimont, R. and Letier, E. (1998) Managing conflicts in goal-driven requirements engineering, *IEEE Transactions on Software Engineering*, 24, 908-926.
31. van Lamsweerde, A. Darimont, R. and Massonet, P. (1995) Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt, *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*, March 27-29, 194-203.
32. van Lamsweerde, A. and Letier, E. (1998) Integrating obstacles in goal-driven requirements engineering, *Proceedings of the 20th international conference on Software Engineering*, Kyoto, Japan, 53-62.
33. van Lamsweerde, A. and Letier, E. (2000) Handling obstacles in goal-oriented requirements engineering, *IEEE Transactions on Software Engineering*, 26, 10, 978-1005.
34. van Lamsweerde, A. and Willemet, L. (1998) Inferring declarative requirements specifications from operational scenarios, *IEEE Transactions on Software Engineering*, 24, 12, 1089-1114.
35. Webster, J., and Watson, R. (2002) Analyzing the past to prepare for the future: Writing a literature review, *MIS Quarterly*, 26, 2, 13-23.
36. Yu, E. and Mylopoulos, J. (1998) Why goal-oriented requirements engineering, *Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality*, June 8-9, Pisa, Italy.