

Integrating requirements engineering and cognitive work analysis: A case study

Neil A. Ernst
Dept. of Comp. Sci.
University of Toronto
Toronto, Ontario, Canada
nernst@cs.utoronto.ca

Greg A. Jamieson
Dept. of Mech. and Ind. Eng.
University of Toronto
Toronto, Ontario, Canada
jamieson@mie.utoronto.ca

John Mylopoulos
Dept. of Comp. Sci.
University of Toronto
Toronto, Ontario, Canada
jm@cs.utoronto.ca

Abstract

Requirements engineering is a fundamental component of systems engineering. This paper describes how, and where, cognitive engineering (specifically the Cognitive Work Analysis (CWA) framework) could be applied in requirements engineering. We introduce an existing RE toolkit, the ι^* modeling framework, and compare it with the CWA framework. The paper concludes with an outline of opportunities to integrate the techniques of cognitive engineering with those from requirements engineering in the design of complex sociotechnical systems.

Introduction

This paper is an attempt to link research results in cognitive engineering (CE) with problems and research in the requirements engineering of software-based systems. Many complex systems have a large software component. Specifying software requirements – that is, the features and functionality a particular installation ought to have – is known as requirements engineering (RE). Although we focus on requirements in software, we believe the fundamental issues in requirements analysis are similar in all systems engineering – whether those systems are social, software, or hardware. Research in cognitive engineering has implications for improving the RE process in system design.

One framework for undertaking cognitive engineering is Cognitive Work Analysis (CWA). CWA, as described in Vicente (1999), takes an ecological perspective on work analysis, placing environmental constraints, including work domain limitations, at the forefront of analysis. The formative method of design CWA proposes has been applied in many areas, including domains such as procurement (Sanderson and Naikar, 2001) and chemical engineering (Jamieson and Vicente, 2001).

One problem in RE that CWA might be able to help with is in system definition. While most RE researchers agree that defining scope is essential (Jackson and Zave, 1993), there are few suggestions on how to determine this. Querying stakeholders only produces those requirements these stakeholders think they need, potentially ignoring other (implicit) requirements. CWA, by contrast with RE frameworks, places the structural, work domain constraints first, ensuring system scope and characteristics have primacy in the analysis.

Combining this contribution of CWA with RE ought to provide a useful synergistic approach to the engineering of software in particular, and systems in general. This paper will suggest where, and how, those combinations should be made.

In this paper we focus on one phase of RE, that of early stage requirements analysis, demonstrating how the work domain analysis component of CWA can enhance the requirements analysis stage of systems engineering. We first introduce both cognitive engineering and requirements engineering. We introduce the ι^* (pronounced i-star) analysis framework as a representative model, and discuss the parallels and distinctions between it and CWA, using a standard research example of a meeting scheduler system. Potential improvements in the RE methodology are highlighted. A brief survey of related work in this area is discussed in the following section. We conclude with some other areas of overlap between RE and CE that merit further study.

Is Software a Complex, Socio-technical System?

Before examining these related areas, it is important to assess whether CWA might be a useful framework for software-based systems. CWA is intended for the design of complex, socio-technical systems. Vicente (1999) lists eleven criteria that define such systems. In software-based systems, some of these criteria include social interaction coupled with different perspectives on the problem; a distributed, dynamic system and coupled subsystems; a degree of automation with uncertain quantitative data; and potential disturbances to the system. Many complex socio-technical systems have a software component, such as airplane cockpits and chemical plants. The CWA perspective is applicable to those software-based systems which meet a majority of these criteria.

Background: CE and RE

Cognitive engineering and software requirements engineering are active research areas. In both areas, there are many proposed frameworks, but insufficient empirical evaluation. This section introduces CWA and software requirements engineering in the research context.

Cognitive Work Analysis

Cognitive engineering frameworks are concerned with describing and analysing the structure of man-machine systems (which are structures for accomplishing goals, with both human and machine/computer components). One of the fundamental differences between the CWA framework and other techniques is its focus on the *structure* of a domain as the first step in the analysis. Other approaches, such as cognitive systems engineering (Hollnagel and Woods, 1999) or cognitive task analysis (Schraagen et al., 2000), are focused largely on the functions and actions a system (including an operator) must engage in. They tend to be procedural descriptions. CWA, by contrast, begins first with a declarative description of what the domain is, to ground further analysis. This is particularly important in complex domains: it is necessary to understand fundamental constraints, such as thermodynamics, gravity, and organizational mandates prior to engaging in analysis of functions in those

systems. This is akin to understanding the terrain before understanding what paths one can take therein.

The CWA framework (Vicente, 1999) is comprised of five stages of analysis, all of which serve to narrow down the potential design space for a particular problem. At each successive phase of analysis, constraints and affordances from previous phases act on the new focus of study. The phases may be of more or less importance in certain domains, and conducting a complete CWA does not presume that the stages are done strictly sequentially. Indeed, insights from one phase often inform earlier phases. The result of this process is an analysis of a potential system with a strong focus on environmental constraints, allowing for more successful resolution of unexpected (by users) and unanticipated (by designers) events (Vicente and Rasmussen, 1992). Here, we focus on one phase, the work domain analysis phase, and one technique for conducting the analysis, the Abstraction Hierarchy.

Software requirements engineering

Software requirements engineering identifies what will go into the design and implementation of a software-based system. Requirements are essential to understanding what to create; following the system implementation, they can then be used for validation. A software system has a purpose, and RE is “the process of discovering that purpose . . . and documenting these in a form that is amenable to analysis, communication, and subsequent implementation (Nuseibeh and Easterbrook, 2000, p. 34).”

Requirements engineering in the software context consists of several distinct phases. Requirements are first elicited from stakeholders using a variety of techniques, including standard surveys and questionnaires, interviews, and, more recently, ethnographic techniques. Following elicitation, requirements are modeled and communicated to stakeholders in the design process, typically in some graphical language. This leads to operationalized requirements, which serve to specify a functional product. The final phase consists of evolving and validating requirements in conjunction with some software system.

Early requirements analysis describes an early elicitation phase which sets the context for particular elicitation needs - answering the question *What needs to be elicited?* Since it focuses on domain definition, this phase aligns closely with the work domain analysis phase of CWA. Early requirements frameworks (which are typically goal-oriented), including the KAOS methodology (Dardenne et al., 1993) and the ι^* framework (Yu, 1997), emphasize the need for domain understanding before performing functional analysis.

Tools: ι^* and Abstraction Hierarchies

This section presents a simple example to illustrate the parallels and differences between cognitive engineering and requirements engineering.

Model Problem: the Meeting Scheduler

To compare the two approaches we’ve discussed, we use the example of a meeting scheduling tool, a model problem in software engineering. Model problems are used in software engineering research to act as examples that allow various researchers to compare

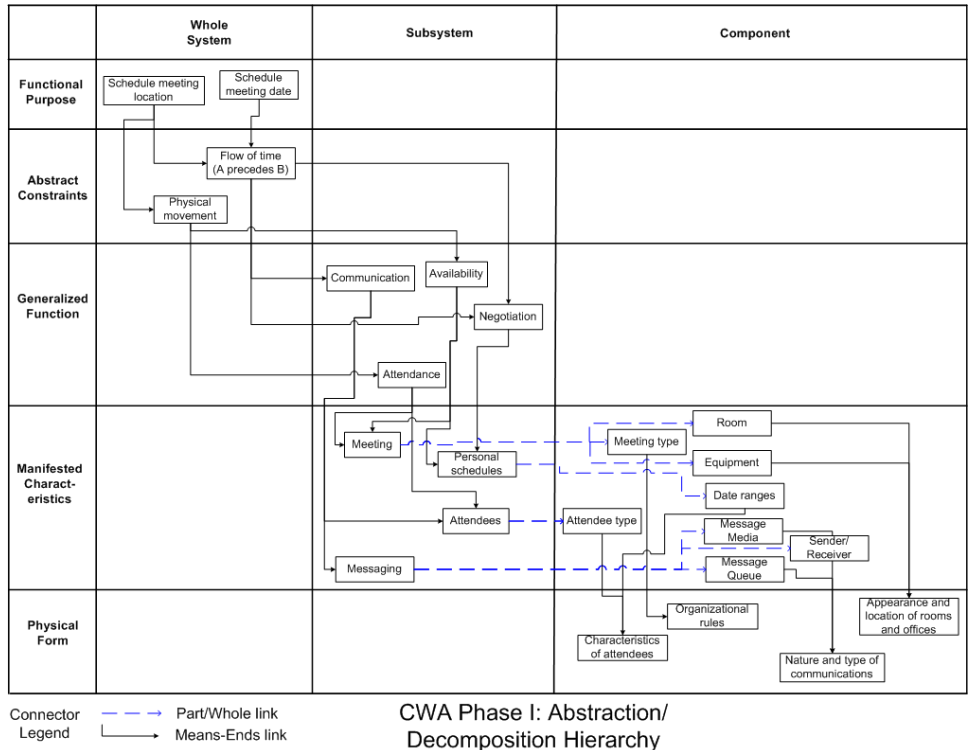


Figure 1: Abstraction-Decomposition Hierarchy for the meeting scheduler domain

solutions and techniques for addressing these problems (for this and other problems, see <http://www-2.cs.cmu.edu/~ModProb/index.html>). The meeting scheduler is a system for scheduling meetings of various participants. It should reduce the headaches currently associated with this process. For a given meeting, there is an initiator and various participants (some of whom may be preferred). Participants define personal exclusion sets (*can't-attend* dates), and preference sets (*would-prefer* dates). The software then determines the set of dates that satisfy those constraints in various ways – typically with feedback from the various stakeholders. Meeting rooms also have availability constraints.

Work domain analysis of the meeting scheduler system

This domain is a complex one due to factors such as uncertainty, numerous stakeholders, and potential disturbances. Complexity would increase as more participants were added, or different components were automated (for example, digital personal assistants). Tools from the CWA framework can be used to analyze it. The Abstraction Hierarchy (AH) is the preferred tool to analyse work domain constraints, and one for the meeting scheduler is shown in Fig. 1.

An AH is a framework for conducting work domain analysis. It captures the structural constraints on the work domain as well as the functional *capabilities* (potential, not realized) of that domain. Abstraction hierarchies consist of levels of system abstraction linked by means-ends links (the *y-axis*), along with decomposition levels linked by part-whole links (the *x-axis*). The conceptual structure of an AH can be varied, as noted in Vicente (1999).

For example, Hajdukiewicz et al. (2001) redefine AH labels in the medical domain. More detail can also be added, such as information requirements. It should be noted that an AH is not intended to capture Actor assignments; that is, any Actor (human or otherwise) could have responsibility for any aspect of the AH - the actual assignments are design specific, and informed by later phases of analysis.

A conceptually difficult aspect of the AH is that it captures declarative aspects of a domain via means-ends relationships. Many software design methodologies start with procedural descriptions – such as task models, use cases, and scenarios. However, in CWA these are explicitly **not** included in the AH. The analysis focuses on the thing being acted on (in this case, the domain of scheduling meetings), and not the goals an Actor has in acting on that domain (Vicente, 1999, p. 162). The abstraction levels are:

1. *Functional purpose* – The work domain’s purposes are to schedule meetings, both in time and space.

2. *Abstract constraints* – Abstract constraints are structural properties of the domain that act as abstract restrictions, such as physical laws. In this domain, these are the notions that attendees can only occupy one place at a given instant, and that events happen sequentially, that is, time has a unidirectional flow. For example, participant A must be able to travel to meeting M’ after meeting M. A more detailed analysis may specify these as logical statements.

3. *Generalized functions* – These are the higher-order functions the domain offers, or needs to achieve. Here, these include the notions of **communication**, **availability** (rooms and participants), **negotiation** (for preferred dates, etc.), and **attendance** (confirm, reject).

4. *Manifested characteristics* – The characteristic functions the domain has to provide. These elements refine the generalized functions (i.e., they are the means of accomplishing generalized functions of the domain). Some of the aspects are **personal schedules**, which constrain the functions of negotiation and availability; **meetings**, which afford attendance and availability; **messaging**, constraining communications. These constructs are decomposed into their constituent parts: for example, **messaging** has components **media**, **queues**, and **sender/receiver**.

5. *Physical form* – The notion of physical form is often vexing in primarily software-based systems. Few aspects of a software system manifest themselves physically, at least in the sense of the appearance and location of controllers that appear in AHs for physical work domains. The domain of interest, though, does, notably in the placement of meeting rooms and individual characteristics of the participants (e.g., disabilities, intentions, desires). We have extended this level of abstraction to reflect the notions of organizational rules and the nature and type of permitted communications: for example, IMAP email, Outlook with meeting request abilities, and so on. These are not strictly physical forms, but are the forms that constrain the functions they are linked to (and these functions may vary).

The result of this process is an abstraction hierarchy which captures the structural means-ends links that hold between components in the domain. It should reflect a series of mental models of the domain at each layer of abstraction.

ι^* analysis of the meeting scheduler system

ι^* is used in early-phase RE for domain understanding. It models intentional actors in the domain, their dependencies (requirements for action) and rationale (why they act, what they

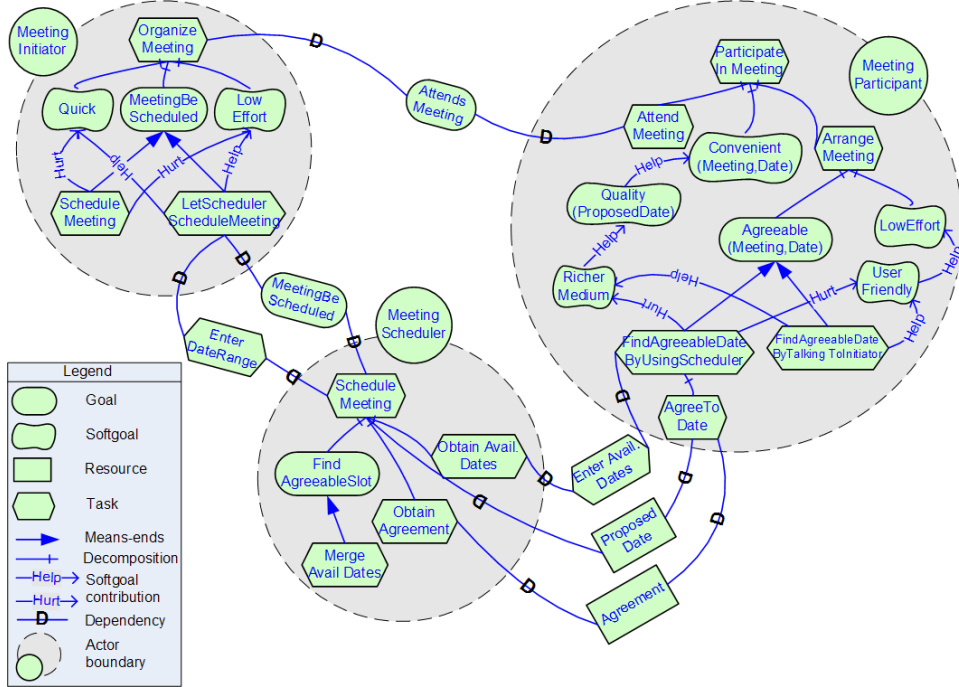


Figure 2: Strategic Rationale diagram of the Meeting Scheduler (adapted from Yu (1997))

require before acting). This is often referred to as goal-oriented requirements engineering. ι^* – the name refers to the notion of distributed intentionality of software systems – consists of two main views of a system. *Strategic Dependency* (SD) graphs show intentional relationships among Actors, which explain ‘why’ they take actions, or the preconditions for intentional action. *Strategic Rationale* (SR) graphs deconstruct individual Actors, examining the internal rationale, or goals, for each Actor. Actors in ι^* have the same definition as in CWA – a human user or automation.

ι^* syntax is largely graphical, although there is work to formalize the language for reasoning support. The constructs include Actors (circles), related by typed arcs representing dependencies. There are four types of dependency: *resource* (e.g., dates, rooms); *task* (e.g., enter information); *goals* (e.g., attend meeting); and *soft-goals*, which are qualitative goals, that is, goals with no clear criteria for satisfaction (e.g., fast execution times). Strategic Rationale graphs add the relationships of *task-decomposition*, *means-ends*, and *contribution*. Contribution is a relationship between some task and a softgoal, indicating that fulfillment of the task positively contributes to satisficing that softgoal (Simon, 1967).

We use these concepts to analyse the meeting scheduler domain. We leverage existing work which has examined this problem in detail. Yu (1997) presents an ι^* model for the meeting scheduler problem (Fig. 2).

The Strategic Rationale model of Fig. 2 provides insight into why particular dependencies exist. Actors are represented by dashed circles containing the various intentional elements associated with them: goals, tasks, and resources. For example, in this model we can see what is motivating the meeting initiator. The task of organizing the meeting is decomposed into the goal of `MeetingBe Scheduled` and the softgoals of `LowEffort` and `Quick`. These softgoals are negatively impacted (shown with contribution links) by the subtask of

ScheduleMeeting. The task `LetSchedulerScheduleMeeting`, however, partially satisfies the `LowEffort` softgoal. This task in turn depends on the `MeetingScheduler` Actor. Means-ends links can show issues that need to be resolved (Yu, 1997). For example, we can now reason about other means of accomplishing tasks, and assess the contributions these have to Actor concerns – for example, `LowEffort`.

Discussion: Improvements for RE from CWA

Differences between ι^* and the AH are attributable to different perspectives on the system design task. The AH is clearly positioned as but one phase of a complete system analysis. ι^* , on the other hand, is a stand-alone early requirements tool for software-based systems design and organizational modeling. In this section we discuss the implications of the previous sections, highlighting some areas where CWA notions can improve the requirements engineering process.

Set the context

The ι^* tools are intended for organizational analysis, providing context and groundwork for the software development process. Several researchers mention the importance of this. Finkelstein (1994) mentions the need to provide context to the RE process. Nuseibeh and Easterbrook (2000) mention the importance of groundwork and context, and also note that requirements gathering and software implementation do not occur in a vacuum. However, they merely note that RE has different meaning in different projects, without defining CWA style notions such as work domain constraints or social organization constraints. ι^* does a good job of providing some contextual setting for further requirements phases, explaining who the relevant Actors are, how they depend on each other, and what motivations drive them. Furthermore, ι^* shows the artifacts that exist in the domain, albeit implicitly. Resource dependency links indicate what components of the domain are important to Actors (and thus to the system itself). However, this context is provided implicitly, without explicit linkages to later phases of engineering.

The work domain analysis phase of CWA, on the other hand, is solely concerned with context. The AH clearly identifies what structures exist, contextualizing the analysis to come. It omits Actors deliberately, with the assumption that constraints provided by Actors are less important to a complex sociotechnical system than those from the domain itself. We suggest that this notion of explicit domain constraints (without Actor involvement, at this stage) can improve the requirements modeling process by providing a concrete contextual setting.

Cascading constraints

In general, ι^* models only implicitly show constraints. In ι^* we see similar concepts to those in the CWA framework, such as the importance of well-defined system boundaries. However, these concepts are buried within the process, and don't have the same significance as in CWA. Similarly, the notion of cascading constraints – that each subsequent phase inherits constraints from the preceding phases – doesn't appear in ι^* . This may be due to

the relative maturity of the CWA framework in this case. Clearly constraints from early requirements analysis should influence later phases of software design, but currently no one framework ties these together. CWA can provide useful insights into how to create such an overarching methodological framework. Explicitly stating domain constraints may be useful at an earlier phase to the ι^* framework, allowing these domain constraints to act as input to the SD and SR diagrams. For example, such a catalogue could identify the resource or information dependencies for use in a SD diagram.

Action and structure means-ends

Perhaps the most significant difference in the two tools is that between structure and action. ι^* conflates structure and action: in the same diagram, we see tasks and resources. The SR model is strongly procedural and task-focused. In ι^* , tasks act on resources. In CWA, though, tasks are examined in a subsequent phase. This is because *what* tasks are performed, and *how*, depends entirely on the resources and structure of the work domain. To wit: the work domain is ‘an unavoidable bedrock of constraint (Hajdukiewicz and Vicente, 2004, p. 532)’.

The reason for this distinction is that ι^* is explicitly actor-oriented, whereas, at least conceptually, the AH does not have a specific Actor in mind. Actors are best defined by what they seek to accomplish. Involving actors in a domain model necessarily invokes the tasks they are linked with. Goals are higher level processes than tasks and structures in the AH: e.g., the goal of attending a meeting becomes the structural component of attendance, which constrains a higher-level goal of scheduling a location for a meeting. In this way, the AH removes references to ‘who’ is involved in the domain, and takes an action-neutral approach to the analysis. Domains have goals, or purposes for which they were created; actors act to fulfil those purposes. Actors are part of the domain in CWA, not above it. In ι^* Actors have primacy and act *on* the domain. Again, this has advantages – we understand the ‘why’ – and disadvantages – the ‘what’ is obscured. CWA here brings clarity to the potential confusion between the two in the RE process.

Conclusions

In this paper we examined two distinct research areas: cognitive engineering of complex systems, and requirements engineering of software-based systems. We outlined the underpinnings of each, using two frameworks as exemplars: cognitive work analysis (CWA), and ι^* . We then used a model problem from software engineering and analysed it with portions of each framework – work domain analysis from CWA, and Strategic Rationale diagrams from ι^* . From that, we outlined three areas where CWA might improve requirements engineering. One is to use Work Domain Analysis (WDA) to inform domain and organizational models, such as ι^* , of the resource constraint relationships. The WDA therefore acts as a means to explicitly establish context, which is of great importance in RE. Secondly, cascading constraints would help with later phases of the system development. Finally, the explicit distinction between the action and the structure of a work domain are very useful in clarifying domain properties.

We see several areas worth further investigation. One, there may be applications of CWA to other requirements engineering frameworks aside from ι^* , such as the KAOS goal modeling language (Dardenne et al., 1993, p. 53). Second, formal reasoning is commonly used in RE. Applying such techniques to CWA models may help to reveal gaps and inconsistencies in the model.

It is important to align software engineering (including software requirements engineering) and systems engineering. As Nancy Leveson writes,

“many of the problems arising in our attempts to build complex systems are rooted in the lack of integration of software engineering, system engineering, and cognitive engineering (Leveson, 1997, p. 130).”

More work is needed to align the various formalisms and frameworks in all three disciplines. Such work is ongoing, including the Systems Engineering Center of Excellence research projects¹, and preliminary analyses of software engineering techniques for systems engineering, such as object-oriented design (Doyle and Pennotti, 2005).

We think CWA orientation towards structure can greatly aid in requirements analysis. For example, most RE research agrees on the idea of properly defining scope and system boundary. However, no further step is taken to place this step on its own – that is, to define the structural constraints themselves. Instead, after the scope is defined, the next phase is to query stakeholders about their desires and requirements. This is a descriptive analysis paradigm (Vicente, 1999), and consequently, the only requirements one will get from a stakeholder are those they think they need. The advantage of starting with work domain analysis first is that while stakeholder views are essential, the analyst maintains a higher-level perspective on the problem, using their views as but one data point to construct a work domain model of the entire system. Requirements engineering frameworks should incorporate similar notions.

Acknowledgements

Many thanks to Steven Deal for his insightful comments on an earlier version of this paper.

Biographies

Neil A. Ernst is a Ph.D. candidate in software engineering at the University of Toronto. His research interests include requirements engineering, information visualization, and human factors.

Greg A. Jamieson is Assistant Professor of Mechanical and Industrial Engineering at the University of Toronto. His research interests include developing the Cognitive Work Analysis framework and extending its domains of application.

John Mylopoulos is Professor of Computer Science at the University of Toronto. His research interests include information modelling techniques, information system design and requirements engineering.

¹<http://www.incose.org/secoe>

References

- Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, April 1993.
- Laurence Doyle and Michael Pennotti. Systems engineering experience with UML on a complex system. In *Conference on Systems Engineering Research*, Hoboken, NJ, USA, March 2005.
- A. Finkelstein. Requirements engineering: a review and research agenda. In *First Asia-Pacific Software Engineering Conference*, pages 10–19, Tokyo, Japan, 1994.
- J. R. Hajdukiewicz, K. J. Vicente, D. J. Doyle, P. Milgram, and C. M. Burns. Modeling a medical environment: an ontology for integrated medical informatics design. *Int J Medical Informatics*, 62(1):79–99, June 2001.
- John R. Hajdukiewicz and Kim J. Vicente. A theoretical note on the relationship between work domain analysis and task analysis. *Theoretical Issues in Ergonomics Science*, 5(6): 527–538, November 2004.
- Erik Hollnagel and David D. Woods. Cognitive systems engineering: New wine in new bottles. *International Journal of Human-Computer Studies*, 51(2):339–356, August 1999.
- M. Jackson and P. Zave. Domain descriptions. In *International Symposium on Requirements Engineering*, pages 56–64, San Diego, CA, 1993.
- Greg A. Jamieson and Kim J. Vicente. Ecological interface design for petrochemical applications: supporting operator adaptation, continuous learning, and distributed, collaborative work. *Computers & Chemical Engineering*, 25(7-8):1055–1074, August 2001.
- Nancy G. Leveson. Software engineering: stretching the limits of complexity. *Communications of the ACM*, 40(2):129–131, February 1997.
- Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA, May 2000.
- P. M. Sanderson and N. Naikar. Evaluating design proposals for complex systems with work domain analysis. *Human Factors*, 43(4):529–542, 2001.
- Jan M. Schraagen, Susan F. Chipman, and Valerie J. Shalin, editors. *Cognitive Task Analysis. Expertise: Research and Application Series*. Lawrence Erlbaum Associates, New Jersey, June 2000.
- H. A. Simon. Motivational and emotional controls of cognition. *Psychology Review*, 74(1): 29–39, January 1967.
- Kim J. Vicente. *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work*. Lawrence Erlbaum Associates, New Jersey, April 1999.
- K.J. Vicente and J. Rasmussen. Ecological interface design: theoretical foundations. *IEEE Transactions on Systems, Man and Cybernetics*, 22(4):589–606, 1992.
- Eric S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *International Symposium on Requirements Engineering*, pages 226–235, Annapolis, Maryland, 1997.