International Journal of Human-Computer Studies, vol. 58, no. 2 (2003), pp. 215-239
Preprint

Making Use of Scenarios: A Field Study of Conceptual Design

MORTEN HERTZUM

Centre for Human-Machine Interaction, Risø National Laboratory, P.O. Box 49, DK-4000 Roskilde, Denmark.
email: morten.hertzum@risoe.dk

Abstract. Scenarios have gained acceptance in both research and practice as a way of grounding software-engineering projects in the users' work. However, the research on scenario-based design (SBD) includes very few studies of how scenarios are actually used by practising software engineers in real-world projects. Such studies are needed to evaluate current SBD approaches and advance our general understanding of what scenarios contribute to design. This longitudinal field study analyses the use of scenarios during the conceptual design of a large information system. The role of the scenarios is compared and contrasted with that of three other design artefacts: the requirements specification, the business model, and the user interface prototype. The distinguishing features of the scenarios were that they were task-based and descriptive. By being task-based the scenarios strung individual events and activities together in purposeful sequences and, thereby, provided an intermediate level of description that was both an instantiation of overall work objectives and a fairly persistent context for the gradual elaboration of subtasks. By being descriptive the scenarios preserved a real-world feel of the contents, flow, and dynamics of the users' work. The scenarios made the users' work recognisable to the software engineers as a complex but organised human activity. This way the scenarios attained a unifying role as mediator among both the design artefacts and the software engineers, whilst they were not used for communication with users. The scenarios were, however, discontinued before the completion of the conceptual design because their creation and management was dependent on a few software engineers who were also the driving forces of several other project activities. Finally, the software engineers valued the concreteness and coherence of the scenarios although it entailed a risk of missing some effective re-conceptions of the users' work.

Keywords : Conceptual design, Design artefacts, Scenarios, Task-based descriptions, Empirical studies

1 Introduction

During conceptual design, software engineers are typically faced with open-ended and poorly understood problems (e.g., Curtis, Krasner & Iscoe, 1988; Potts & Catledge, 1996; Rasmussen, Pejtersen & Goodstein, 1994). To cope with these conditions and reach closure on a common vision for the system they are designing, software engineers create a series of design artefacts as they go through requirements engineering, business modelling, and functional specification. One such design artefact is scenarios. Since the late 1980s, researchers in human-computer interaction have used scenarios in working with system requirements (e.g., Carroll, 1995, 2000; Jarke, 1998a; Jarke & Kurki-Suonio, 1998). In the late 1990s, scenarios have gained considerable practical popularity through the Use Case approach, which has become part of major methods for object-oriented software engineering (Jacobson, 1995; Jacobson, Booch & Rumbaugh, 1999). Scenarios appear to address a real problem and to fit into the design processes of practising software engineers. Nevertheless, Jarke (1998b, p. 153) observes that *"scenarios are possibly one of the least understood recent success stories in the information technology (IT) and management areas"*. This calls attention to a profound bias in the literature on scenario-based design (SBD). Despite numerous papers describing, discussing, and/or recommending SBD there is very little empirical material available on how practising software engineers have applied SBD in real-world projects and to what effect.

This study reports from a longitudinal field study of the conceptual-design stage of a software-engineering project. The project, estimated to last three years, concerns the complete redevelopment of a nation-wide information system for supporting municipal authorities in their administration of child support and alimony payments. This study investigates the first twelve months of the project, during which scenarios were one of the main design artefacts applied in getting to grips with the complexities of the application domain. The role of the scenarios in the design process is analysed by comparing and contrasting the scenarios with three other design artefacts that were also developed during the first year of the project: the requirements specification, the business model, and the user interface (UI) prototype.

This study is based on a single project which, however, shares several of its dominant features with numerous other projects. Three such features appear particularly noteworthy. First, the project is staffed with a multidisciplinary team where people from analysis and design, component development, customer support, human factors, and technical services work together throughout the development process. Second, many activities are performed concurrently to shorten development time and reduce the gap from a decision is made to its consequences become apparent. Third, even though the project is a redevelopment project the spread of domain knowledge among the involved people is very uneven. These features instigate an urgent need for the establishment of a common ground that conveys a sound understanding of the users' work and enables coordination and communication among a group of people with different responsibilities, competencies, and mindsets.

The next section gives a brief account of SBD including a review of previous empirical studies of scenario usage in real-world projects. Section 3 describes the empirical data collected from the project that is analysed in this study. Sections 4 and 5 introduce the project and the four main design artefacts produced during its first year. In Section 6 the four design artefacts are compared and contrasted, and the unifying role of the scenarios is analysed through an exposition of the descriptive, task-based accounts they provide of the users' work. Finally, Section 7 is about implications of the study for tools intended to support conceptual design.

2 Scenario-based design (SBD)

Scenarios emerged as a methodology for strategic management in the late 1940s and their use has since spread to other areas. Jarke, Bui, and Carroll (1998) span three areas – strategic management, human-computer interaction, and software engineering – in their survey of how scenarios contribute to analysis and design. Their findings, summarised in Table 1, show some variation across the three areas but also point to a common underlying role of scenarios: to ground decisions in a sound and communicable understanding of the use situation. This role, which will be investigated and elaborated upon throughout this study, is also at the heart of Carroll's definition of the scenario concept:

> *The defining property of a scenario is that it projects a concrete narrative description of activity that the user engages in when performing a specific task, a description sufficiently detailed so that design implications can be inferred and reasoned about.* (Carroll, 1997, p. 385)

Insert Table 1 about here

2.1 A VARIETY OF SBD APPROACHES

One influential approach to SBD is that of Carroll and Rosson (1992). They treat design as a process that alternates between what users need and what technology has to offer. In this process designers respond to user requirements by building artefacts, such as scenarios or prototypes, which in turn present or deny possibilities to the users. Scenarios are introduced as an analytic way of capturing the use context. Thus, instead of, or in addition to, empirical techniques such as prototyping it is possible to develop scenarios that make the future use situation concrete and available for exploration. Contrary to empirical techniques, "s*cenarios have the important property that they can be generated and developed even before the situation they describe has been created*" (Carroll & Rosson, 1992, p.190). The strong focus on use is driven by a view of computer systems as rich and dynamic contexts for user activity. Consequently, this approach to SBD aims at opening up the use situation and treating its myriad of mundane details as credible and important expressions of the real-world situations the system is to match. This leads to scenarios that use text, storyboards, video, and other media to provide open-ended descriptions of the activities a user might engage in while pursuing a particular concern, combined with details about the setting in which the user is acting.

Other approaches to SBD have a different perspective on conceptual design or utilise scenarios for activities other than conceptual design. In addition to the SBD approach of Carroll and Rosson (1992), scenarios are, for example, used in:

- Object-oriented design, to embody the needs inherent in current or projected work practices – the Use Case approach (Jacobson, 1995; Jacobson et al., 1999).

- Requirements engineering, as a means to identify what the system is to achieve and how it fits into the users' work situation (Haumer, Pohl & Weidenhaupt, 1998; Hooper & Hsia, 1982; Kaindl, 2000).
- Participatory design, as a means of illustrating important design issues or possible designs in a way understandable to both users and designers (Bødker & Christiansen, 1997; Kyng, 1995).
- Documentation and training, as a means to bridge the gap between the system as an artefact and the tasks users want to accomplish using it (Kaminsky, 1992; Smith, Dowell & Ortega-Lafuente, 1999).
- Evaluation, as a means of defining the tasks against which the system is to be evaluated (Nielsen, 1995).

Depending on the purpose of using scenarios they will differ in contents, form, and lifecycle. Recent work on scenarios has emphasised that lifecycle issues, especially the need for scenario management, is an area that has not received sufficient attention, at least not in the literature (e.g., Jarke et al., 1998; Jarke & Kurki-Suonio, 1998; Weidenhaupt, Pohl, Jarke & Haumer, 1998). Jarke et al. (1998) identify four key research questions for SBD, and scenario management stands out as the overriding issue: (1) How do we deal with collections of scenarios? (2) How do we deal with coverage? (3) What aspects of scenarios are essential, and what are inconsequential? (4) What are boundary conditions for scenario applicability? The fourth question also points out that whereas the benefits of using scenarios have frequently been described, relatively little is known about when and at what cost software engineers can expect to reap these benefits.

2.2 ECOLOGICAL STUDIES OF SBD

Studies of how practising software engineers use scenarios in real-world projects are needed to evaluate current SBD approaches, provide directions for improving them, and enhance our general understanding of what scenarios contribute to design. Three such studies are reviewed below, and they may be the only ones of their kind. In addition, the literature contains studies in which researchers apply their SBD approach to illustrate the viability and flow of the analysis (e.g., Dzida & Freitag, 1998; Kazman, Abowd, Bass & Clements, 1996) or to design prototypes (e.g., Carroll, Rosson, Chin & Koenemann, 1998; Karat & Bennett, 1991). The qualities of these additional studies unsaid, they cannot replace studies of scenario usage in real-world projects.

Weidenhaupt et al. (1998) conducted a survey of current SBD practices in 15 projects covering a broad range of application domains and project sizes. Each project was visited for a half or a full day, and one or two project participants, mostly project managers, were interviewed. The main finding was that the use of scenarios varied greatly from project to project with respect to purpose, form, contents, as well as lifecycle. Some used narrative text to produce extensive descriptions of how the system interacted with its environment, and used these descriptions throughout the development process. Others used diagrammatic notations to produce dense descriptions of interactions among internal system components, and used these descriptions at a few clearly defined points in the development process. The most frequent purposes of using scenarios were to facilitate agreement and consistency (all 15 projects), to make abstract models concrete (13 projects), to enforce interdisciplinary learning (13 projects), and to reflect on static models (12 projects). In all the projects, the engineers viewed the creation, documentation, and evaluation of scenarios as a major effort because the scenarios evolved over time and this evolution had to be managed. The need for scenario management increased with the size of the projects, the complexity of the application domains, and as scenarios became increasingly pervasive artefacts used for manifold purposes throughout the system lifecycle. In most projects, the scenarios were out of date when system testing began and it was therefore not possible to derive test cases from the scenarios, though such a systematic approach to test case generation was considered highly desirable. Weidenhaupt et al. (1998) conclude by characterising scenarios as bridges that can provide links among various parts and stakeholders in the development process, but they also note that to do so effectively practitioners need better tool support and more guidance on when it pays to use scenarios.

Gough, Fodemski, Higgins, and Ray (1995) studied the use of scenarios (use cases) during the 18-month requirements-engineering phase of a project concerning the development of a medical control and monitoring system. The study was based on inspection of project documentation and five interviews with requirements engineers and customer representatives. The requirements engineers were first-time users of scenarios and received an introduction to scenarios at the beginning of the project. The scenarios were developed in two steps. First, scenario headings and brief descriptions were generated, modified, and reviewed to ensure that the full usage of the system was captured. Second, the scenarios were refined, written up in natural language, and reviewed. A typical scenario consisted of 3-4 pages of text in which the flow of activities was described in a standard, natural-language form and bulleted lists were used to indicate branching within the description. A total of 88 scenarios were developed, divided onto 31 representing the views of customers and 57 representing engineering views. The engineers felt that the use of scenarios was both creative and motivating, though time pressures and long debates about scope led to some demotivation. Four of the five people interviewed were in favour of continued use of scenarios; the last person felt they were no better than the short *shall* statements that made up the requirements specification. In addition, the formal reviews were considered an effective way of validating scenarios. The major shortcoming of the scenarios was that it was difficult to validate interactions among scenarios, especially for the customer representatives. Also, the purely textual format was considered inadequate; more graphics and flow charts were requested. Gough et al. (1995) conclude that the scenarios clearly succeeded in providing a means by which requirements engineers, customer representatives, and other stakeholders in the development process could work with system requirements.

Potts and Catledge (1996) studied the conceptual-design activities of an industrial software project for three months with follow-up observations and discussions over the following eight months. The aim of the study was to understand how patterns of collaboration and communication affect the convergence of the project on a common vision and a documented specification. The role of scenarios was one of the research foci, but the studied project was unbiased in the sense that it was not selected on the basis of a commitment to using scenarios. This is in contrast to the studies by Weidenhaupt et al. (1998) and Gough et al. (1995) where projects were only selected for study if scenarios were a major design artefact. Three findings stood out: (1) a painfully slow convergence on a common system vision, (2) repeated returns to certain issues on which the engineers failed to reach closure, and (3) a failure to ground the system architecture in concrete considerations of user needs. Scenarios were rare. Instead, requirements were stated generally and discussions about requirements were driven by general questions, rather than imagined usage. Also, the engineers focused on the static aspects of the system architecture such as where functions resided, rather than the dynamics such as when functions were invoked – a finding consistent with the infrequent use of scenarios. The scenarios that did arise were not named or referred to later. For example, during a meeting the engineers made and used a scenario but apart from temporarily clarifying some hitherto cloudy issues this discussion was never revisited and it was not reflected in the design artefacts. The near absence of scenarios did not lead to increased use of other design artefacts, rather the engineers carried contributions to the architecture in their heads. Potts and Catledge (1996) provide no data on why scenarios were virtually unused but, rather, conclude that scenarios seem to focus on issues that stood out as core problems in the conceptual-design process.

3 Method

The data collected for this study cover the first twelve months of a software-engineering project estimated to last three years. A longitudinal study was chosen to be able to span the conceptual design of a comprehensive, real-world project and experience how different activities and design artefacts contributed to the progress of the project. To follow the course of a project for twelve months it was necessary to restrict the data collection to selected points in the process. This led to choosing observation of project meetings as the primary means of data collection because they are themselves important project activities and they provide summary information about the period since the last meeting. All-together the data collection comprised: attendance at the two-day start-up seminar, observation of the project status meetings, which took place once every two weeks, interviews with eleven of the core project participants, and inspection of about a thousand pages of project documentation. As a precursor to the data analysis, the interviews and the observed meetings were recorded on tape and transcribed. This study is based on an analysis of the meetings and the various documents that concern the four main design artefacts, supplemented with data from the interviews.

A total of 24 *project meetings* (52 hours) have been observed. The main purposes of these meetings, which gathered the entire project group, were to provide a forum for sharing information about the status of the project, maintaining awareness of the entire project, coordinating activities, discussing problems and progress, making decisions, and reviewing major project documents. During the meetings, I have been seated at the meeting table with the other people present. From their point of view, I have been invisible in that I was not to be spoken to and have myself remained silent. During the breaks, I have talked informally with people. In addition to the tape recording of the meetings, I have made notes about particularly interesting topics and copied the contents of the whiteboard when it was used. *Documents* handed out at the meetings were also handed out to me; the remaining project documents were available to me upon request. The documents, which provide evidence of the evolution and intermediate outcomes of the project, include among other things the project plan specifying subprojects and milestones, the project diary with brief notes about important events and decisions, and the final as well as several preliminary versions of the requirements specification, the business model, and the scenarios. The *interviews* provided an opportunity to talk with people about their individual experiences and concerns, and to dig deeper into issues and discussions that were merely hinted at during the meetings. The interviews, which lasted 1-1½ hour each, were loosely structured by a set of guiding questions prepared for the individual interview. The interviews progressed as conversations focused on the parts of the project in which the interviewee was involved.

4 The CSA project

The company where the field study took place is a large software house, which has developed and marketed a range of systems for use in municipal institutions. The studied project concerns a system to support municipal authorities in their handling of cases concerning child support and alimony (CSA). CSA cases come about when couples divorce and seek support to set up and enforce an arrangement of regular child-support or alimony payments. Several factors contribute to the complexity of CSA cases. First, whereas the handling of CSA cases is prescribed in detail in legislation, there is a large gap between these terse texts and the richness of real-world cases. The CSA system must both enforce the legislation and accommodate the variety of concrete cases. Second, CSA cases depend on a large number of circumstances in the divorcees' lives. These circumstances are likely to change on numerous occasions during the yearlong duration of a CSA case. Third, though CSA cases have their unique characteristics they should be modelled in ways that make it easy for municipal authorities – the users of the CSA system – to handle CSA cases alongside other types of cases. This includes that the CSA system should be technically integrated with several other systems. Finally, many divorcees are unhelpful or even hostile toward each other. Thus, the CSA system must, for example, provide municipal authorities with information about the current addresses of all persons involved in a CSA case but never reveal this information to the persons involved in the case.

The CSA project is to completely redevelop the company's existing CSA system, which has been in operation for almost two decades. During the first twelve months of the project, the period analysed in this study, the project was staffed with a project manager, eleven designers/developers, two service consultants, a methods & tools consultant, a usability specialist, and a secretary. The project manager and six of the designers/developers worked full time on the CSA project, the remaining ten persons were assigned to the CSA project on a part-time basis. This multidisciplinary group is comprised of people with an average of more than ten years of professional experience. Further, the majority of them have been extensively involved in the development and maintenance of the existing CSA system. When referred to as a group, the members of the CSA project will be termed CSA engineers, irrespective of their different backgrounds.

To accomplish their task, the CSA engineers have to interact with external stakeholders such as user representatives and the governmental bodies responsible for the administration of the laws regarding child support and alimony. Naturally, they also have to interact with management, marketing, the quality function, and other internal stakeholders in the software-engineering process. Whereas the existing CSA system contains substantial amounts of code that duplicate functionality from other systems made by the company, the new CSA system will distribute this functionality onto components that are to be developed by other project groups in the company. This adoption of component-based design means that the CSA engineers have to cooperate closely with a number of people outside the project to negotiate, settle, and follow up on component definitions and how the development of the components progresses (Hertzum, 2000). In addition to component-based design, the software-engineering methodology of the company prescribes the use of a CASE (Computer-Assisted System Engineering) tool, which assists the development process from business modelling to code generation.

5 The four main design artefacts

The four main design artefacts produced during the conceptual design of the CSA system were the requirements specification, the scenarios, the business model, and the UI prototype. Figure 1 shows how the activities involved in producing these four design artefacts were distributed over the first year of the project. Other design artefacts not treated here were, for example, the documentation plan and the information plan.

Insert Figure 1 about here

5.1 REQUIREMENTS SPECIFICATION

The first three months of the project concerned the requirements engineering and resulted in a written requirements specification. The purpose of the requirements specification was to make explicit what the system was to achieve and what constraints it had to operate within. Thereby, the requirements specification would provide both a checklist for the CSA engineers during the continued design process and a contract between the CSA engineers and the user representatives. The initial inputs to the requirements engineering were: (1) the records from the support centre where users report problems and get help from the service consultants, (2) seven interviews with users of the existing CSA system, (3) contributions by the CSA engineers, especially the service consultants, based on the domain knowledge they had acquired from their involvement in the development, maintenance, and operation of the existing CSA system, and (4) three meetings with the user representatives who were a group of prominent users of the existing CSA system. These inputs yielded 221 requirements, most of which were functional requirements. The requirements were maintained as individual entries in the requirements specification, which added a categorisation that grouped related requirements. The scenarios and business model were made after the requirements specification though they could have provided additional input. It was probably the redevelopment nature of the CSA project that made it practicable to create the design artefacts in this order.

5.2 SCENARIOS

Scenarios are not a prescribed element of the design methodology of the company but when the requirements specification was completed the CSA engineers turned to scenarios as a means of describing the business the CSA system was to handle. Such opportunistic use of scenarios seems rather common (see Section 2) but also implied a rather uneven distribution of the CSA engineers' familiarity with the use of scenarios. The initial motivation for introducing scenarios was that the CSA engineers felt they needed a common worldview, which had not been provided by the requirements engineering. They started by developing a scenario for how municipal authorities handle a normal CSA case and then gradually extended it with the various exceptions and difficult cases. The scenarios consisted of step-by-step descriptions of the users' tasks but were sufficiently abstract to remain independent of any specific system attribute. The format was fairly schematic in that each scenario step was described in terms of three constituents: (1) the event that triggered this step of the scenario, (2) the business process carried out at this step, and (3) the sequence of elementary processes that were involved in performing the step. A total of 30 scenarios were developed, amounting to 69 pages of text. Each scenario described a central task, broke this task into steps, and the steps into elementary processes, which were at a level of detail appropriate to be transferred to the business model (see Figure 2, for an example of what scenario steps looked like). About six months into the project the CSA engineers ceased to develop and update the scenarios because they had to be documented and maintained outside the CASE tool and because an upcoming project deadline demanded all the CSA engineers' resources. Shortly after the discontinuation of the scenario development the scenarios were out of date and the CSA engineers stopped using them.

Insert Figure 2 about here

5.3 BUSINESS MODEL

The business modelling was initiated a month before the completion of the requirements specification and lasted six months. In the early stages, the business modelling ran in parallel with the scenarios, which came to be considered the better way of getting a systematic view of what constitutes CSA work. Consequently, the CSA engineers gave priority to the scenarios and used them to identify the elementary processes, which would otherwise have been identified during the business modelling. The business modelling, which became the dominant project activity when the scenarios were discontinued, was about identifying the elementary processes that constitute CSA work and the events that trigger them. The purpose of the business modelling was to provide the founding description of CSA work in a format suitable for input into the CASE tool. This made the business modelling considerably more system-oriented than the requirements engineering and scenario development in that it had to take into account how different ways of representing the acquired understanding of CSA work affected the outputs that could subsequently be generated with the CASE tool. The business model consisted of an enumeration of the events and elementary processes involved in CSA work, supplemented with definitions of key business concepts and a brief, high-level description of how the CSA system related to the company's other systems. The business model was the result of numerous discussions among the stakeholders involved in the development of the new CSA system but these discussions were normally driven by the requirements specification, scenarios, or UI prototype, rather than by the business model itself.

5.4 UI PROTOTYPE

The CSA engineers refrained from walking through the scenarios with the user representatives. While such a walkthrough would validate the scenarios, the CSA engineers found that it would take too long and be of little use and interest to the user representatives. The CSA engineers wanted to show the user representatives a prototype of the new system, not a schematic description of CSA work. The UI prototype was developed to visualise how the requirements were turned into a design and to support the user representatives in articulating how various aspects of their tasks made certain designs useful and others inappropriate. Like the scenarios but contrary to the requirements specification and business model, the UI prototype was not a mandatory element of the design methodology of the company. The first two versions of the UI prototype were printouts of draft screens, which were presented to the group of user representatives to stir discussion. The third and fourth version of the UI prototype were computer-based and evaluated by having test users solve a number of set tasks while thinking out loud. Gradually, more and more facilities were incorporated in the prototype, but its coverage (horizontal scope) remained partial. Compared to the paper versions of the UI prototype, the computer-based versions were more refined, with a granularity (vertical scope) that spanned overall screen layout and increasing amounts of detail regarding dialogue flow and specific screen elements. This evolution was reflected in the purpose of the UI prototype in the sense that the successive versions were progressively more focused on validating the proposed design and correspondingly less concerned with generating new ideas.

6 The unifying role of the scenarios

The scenarios were developed over a period of three months and during these three months they attained a pivotal position in the CSA project. As discussed below the scenarios supported the CSA engineers in establishing a common ground that was easy to relate to and provided them with a much-needed understanding of the contents and flow of the users' work. Figure 3 summarises the following discussion by illustrating how the scenarios, as the only one of the four design artefacts, provided a descriptive account of CSA work at the level of the users' tasks.

Insert Figure 3 about here

6.1 A TASK-LEVEL ACCOUNT

In the CSA project, as in the majority of the SBD literature, it is a prominent characteristic of scenarios that they are pitched at the level of the users' tasks. For example, the CSA engineers started by making a scenario for the creation of a straightforward CSA case. Subsequently, other scenarios were added and the creation scenario was extended with various exceptions. Some of the circumstances that complicate CSA cases are of an administrative nature, such as when the person obliged to pay CSA lives in, or moves to, another country with different CSA regulations. However, a large number of the difficult cases are caused by the emotional and deeply personal nature of the situations that give rise to CSA cases. Furthermore, a number of the persons involved in CSA cases have other severe problems in addition to their CSA case. Thus, the CSA engineers developed scenarios specifically for the CSA administration necessary to handle the situation in which a divorced couple's child is taken into foster care.

Some of the scenarios were triggered by other scenarios; others by external events such as the death of the person obliged to pay CSA. The scenarios define about 140 events that impact on the flow and treatment of CSA cases. The events are an example of a refinement that was largely absent in the early versions of the scenarios but added as the scenarios grew in scope and detail. The events were an important means of representing relations among the scenarios. Further, the scenarios were the only design artefact aimed at providing an account of the interaction among the various events and activities that constitute CSA work. To begin with, these accounts were fairly unstructured sequences of very concrete incidents. Later, the accounts were expressed in terms of more and more refined concepts but they remained descriptions of courses of activities:

> *We have developed the scenarios because we have felt that to be able to grasp what the system is to be capable of it is necessary that you can imagine some courses of activities. We call these courses of activities scenarios. When you, in the beginning of the process, imagine a course of activities then you start with somebody knocking on the door, you say "hello", she enters, you take a piece of paper and do like this and like that. What happens along the way is, for one, that we get an outline of the business model. So when we continue thinking about the course of activities we have to ask ourselves: What is actually happening here? What are the constituents of this? […] What we need the scenarios for is to gradually describe some courses of activities in more and more detail.* (Project meeting middle of month 5)

The courses of activities that delineated the scenarios were the users' tasks. The tasks bound events and activities together into threads where the meaning and consequences of individual events and activities were made clear by the surrounding events and activities. This way the courses of activities – the tasks – made up a set of interwoven threads. The tasks provided a structure and, thereby, some handles for interacting with the gradually more and more refined descriptions. While the descriptions got more and more refined, the structuring of the descriptions into tasks maintained a fairly constant level of detail. The CSA engineers found that the tasks (threads) provided a unitary way of gaining access to all the details related to a specific issue:

What the scenarios show us at the moment is: "Given this situation, what is the course of activities we imagine?" There you have the green threads down through the description. You pull a thread and check whether it holds together all way through. (Project meeting beginning of month 5)

When the project manager decided to discontinue the scenarios her major reasons were that creating and, especially, keeping them up to date had become very time-consuming, and that the consumed resources were needed for other activities in order to meet an upcoming deadline. An important factor in this decision was that very few of the CSA engineers knew CSA work well enough to produce the more fine-grained parts of the scenarios. These two or three CSA engineers were, however, also the driving forces of several other project activities. The decision to discontinue the scenarios was instigated by the scarcity of these especially qualified persons, rather than by a general shortage of project staff. While the discontinuation of the scenarios certainly freed the especially qualified CSA engineers for other activities in the short term, these key CSA engineers expressed concerns about the longer-term consequences of the decision. They considered the scenarios superior to the other design artefacts in terms of providing the CSA engineers, as a group, with an understanding of the flow of the users' work:

I'm worried that the part containing the processes with events won't be that visible anymore. [In the business model] I may be able to go down and read some elementary processes, which belong to different entities, but then your are a little further away from catching sight of a coherent process than you were with the scenarios we had before. (Project meeting beginning of month 8)

The discontinuation of the scenarios meant that none of the completed design artefacts contained a task-level account of the users' work. Levels not covered by design artefacts may still exist as knowledge possessed by individual CSA engineers. Specifically, the key CSA engineers will probably be able to make sense of the users' work and bridge the gaps between the design artefacts without the scenarios, as they to a considerable extent carry the design in their heads anyway. The other CSA engineers will probably need the scenarios to reason and argue about the relationships between, on the one hand, the users' work domain with its overall objectives and, on the other hand, the individual requirements and elementary processes recorded in the requirements specification and business model. In reasoning and arguing about these things the scenarios provided an intermediate level of description that was both an instantiation of the domain objectives and a context for the subtask activities. Furthermore, engineers are not supported in maintaining a shared understanding of levels not covered by the design artefacts.

Analytically, it is possible to distinguish between a domain (the total work setting in which people perform the tasks relating to a specific subject area; e.g., CSA work), a task (a unit of work that is reasonably self-contained and sufficiently complex to be perceived by the involved people as an undertaking in and of itself; e.g., creating a new CSA case), and a subtask (an element of a task and not in itself perceived as a self-contained unit of work; e.g., determining the income of the person obliged to pay CSA). This distinction between domain, tasks, and subtasks serves to illustrate that the scenarios provide an intermediate level of description, but it also captures substantial differences in the scope of the four design artefacts (see the horizontal axis of Figure 3):

- Nearly all the requirements in the requirements specification are at the subtask level. This is noteworthy but may partly be a consequence of the *re*design nature of the project. The existing system embodies a lot of knowledge about the domain and the users' tasks. This embodied knowledge has a large impact on the requirements specification, which explicitly states that if nothing is noted about a facility the new CSA system should provide the same functionality as the existing system. This way, the domain and tasks are only described indirectly by a general reference to the existing system, and the requirements specification mainly adds to and subtracts from this description at the subtask level (Hertzum, 2002).
- The business model consists of an overall description of the domain and a comprehensive enumeration of events and elementary processes. The domain description aims at positioning the CSA system within the company's framework for systems regarding social benefits. The enumeration of events and elementary processes focuses on providing the founding input for the CASE tool and must be at the subtask level in order not to constrain the ways in which CSA work can subsequently be composed in the CASE tool. The two parts of the business model serve quite different purposes and they are not related through an intermediate task level or in any other way.
- The UI prototype is concerned with how CSA work is presented to the user through the interface, that is, with the actual screens and the dialogue flow of the system. During the development and discussions of the two initial, paper versions of the UI prototype the dialogue flow was a minor aspect of the prototype. So, while the dialogue flow gives some information at the task level this information was not incorporated into the UI prototype until its third version and partly based on input from the scenarios. Hence, the UI prototype has primarily been concerned with the contents and layout of screens and, thereby, with details at the subtask level.

This means that out of the four design artefacts it is only the scenarios that attempt to provide a task-level description of CSA work.

Rasmussen et al. (1994) warn that a description of tasks is bound to be incomplete because it is impossible to anticipate all the events and user needs that will show up in future use situations. Several advocates of SBD acknowledge this incompleteness as an inescapable limitation of scenarios. For example, the task-artefact cycle (Carroll, Kellogg & Rosson, 1991) shows that tasks can only be specified relative to known or envisioned artefacts. This relativity of tasks greatly complicates the use of scenarios and other task-based descriptions as a defining input to the design of new artefacts. Nevertheless, the CSA engineers seem to consider the task focus a major asset of the scenarios. By focusing on the users' tasks the scenarios support the design of a CSA system that does not redefine CSA work to an extent where it is no longer recognisable to the users. The technological platform on which the existing CSA system is based was fixed 15-20 years ago, and the technological developments that have taken place since then are huge. Therefore, the CSA engineers are concerned that it is at least as important to acknowledge the current organisation of CSA work and a smooth transition from the existing to the new CSA system as it is to enhance the system with various facilities that add to its functionality by exploiting new technological possibilities.

Several of the CSA engineers, especially the service consultants, talk about a risk of being too visionary and emphasise that the typical user of the CSA system is not technologically advanced. The scenarios become valuable because they support the CSA engineers in both focusing on the complexity of CSA work and acknowledging how it is handled through the current organisation of work. The scenarios may miss some clever re-conceptions of CSA work but the CSA engineers find that there is room for substantial improvement of the CSA system even without radically re-conceptualising CSA work.

6.2 A DESCRIPTIVE ACCOUNT

The scenarios are schematised descriptions of the courses of activities that constitute CSA work. By characterising the scenarios as descriptive accounts of how CSA work is accomplished it is emphasised that the scenarios preserve a real-world feel of the contents and flow of CSA work and that they are grounded in the dynamics of this work. The real-world feel reflects that the scenarios aim to unravel CSA work by describing what people concretely do, that is, almost without replacing real-world events and activities with abstracted entities. For example, actors are not abstracted away to arrive at functions that have not yet been allocated to either the system or its users. Likewise, the relations among events and activities are not abstracted away to produce lists of elementary entities that have been detached from the way in which CSA work is currently perceived and organised. The grounding in the flow of CSA work means that the scenarios are rich in the information needed in the day-to-day management of CSA cases, such as how activities are sequenced, what triggers them, and when they trigger other activities. This means that the scenarios make the users' work recognisable to the CSA engineers as a complex but organised human activity.

Each scenario consists of a chronological progression of activities. Typically, CSA work progresses continually for only brief intervals of time; then further activity does not occur until, for example, the person obliged to pay CSA disregards this obligation or the person entitled to receive CSA remarries. Consequently, most of the steps in the scenarios are triggered by events. This can be contrasted with essential use cases (Constantine & Lockwood, 1999), in which use situations are modelled by a sequence of user intentions and associated system responsibilities. The events define information that must be provided before further progress can be made, or they lead to the execution of subtasks that are only relevant when certain conditions occur (see Figure 2). This results in scenario descriptions that preserve the real-world ordering of the activities involved in performing a task and also delineate the events or circumstances that affect whether and when the various activities are performed. The CSA engineers perceived the scenarios as quite coherent descriptions of CSA tasks and considered this a valuable and distinguishing feature of the scenarios. As an example, one of the CSA engineers responsible for the UI prototype explicitly acknowledged that the scenarios provided a structured description from which to generate the UI prototype:

> *I think they [i.e., the scenarios] were good in the sense that they told us what notifications we needed. They told us what system-generated letters we needed. They told us the course of activities users go through. That was useful in figuring out how to put a dialogue together.* (Project meeting beginning of month 8)

This CSA engineer was concerned that the discontinuation of the scenarios would deprive her of valuable information about the various aspects of CSA work. This concern was partly an appreciation of the scenarios and partly instigated by the common impression among the CSA engineers that the other design artefacts, especially the business model, did not provide them with an equally good tool for understanding CSA work. What the CSA engineers lost with the scenarios was a design artefact that aimed at describing the users' work as tasks consisting of a structured sequence of interrelated activities. The CSA engineers specifically pointed out that the coherence – the threads – was lost when a scenario was transferred to the business model; that is, when the constituent events and elementary processes were entered into the CASE tool:

> *Once it [i.e., a scenario] has been entered into [the CASE tool] you can't say "I want to pull the green thread, which is deaths, and see what happens". That's not possible. In [the CASE tool] it's not possible to say: "What will happen given this situation?"* (Project meeting beginning of month 5)

Contrary to the scenarios, the requirements specification and the business model can best be characterised as extensive lists that enumerate large amounts of separate details, whereas the UI prototype aims at demonstrating the new design rather than describing CSA work (see the vertical axis of Figure 3):

- The requirements specification lists 221 requirements but provides no information about how these requirements impact on each other. The requirements are categorised by user group and general system function (Hertzum, 2002). This places requirements concerning the same function close to each other but makes no attempt at providing a coherent description where the interplay between the requirements is laid out and shown to form a consistent whole. It is, for example, left entirely to the reader of the requirements specification to determine whether it contains conflicting requirements.
- The business model makes up a similarly fragmented account of CSA work. In the business model the flow of CSA work is discarded in favour of a complete enumeration of the events and elementary processes that enter into CSA work. This enumeration involves only the identification of the individual events and processes; there is little in the way of a scaffolding for relating the details to each other. Rather, events and elementary processes are the decomposed building blocks from which CSA work will subsequently be composed in the CASE tool.
- The UI prototype demonstrates how the CSA engineers have transformed their understanding of CSA work into screen layouts and, near the end of the conceptual design, dialogue flows. This demonstration reflects the requirements and other user input back to the user representatives in terms of a proposed design. Thereby the UI prototype moves a step beyond the other design artefacts, which focus on providing the CSA engineers with a detailed, design-oriented understanding of the users' work and requirements toward a CSA system.

It should be noted that the scenarios were developed as a tool for the stakeholders internal to the CSA project. This project-internal use of scenarios could be considered unnecessary in a redevelopment project where several of the project participants have substantial domain knowledge from the development, maintenance, and service of the existing CSA system. In practice, the redevelopment nature of the CSA project may be the circumstance that made it possible for the CSA engineers to postpone the establishing of a common ground until after they had completed the requirements specification. The descriptive nature of the scenarios made them accessible to all CSA engineers and meant that the scenarios were not biased toward, or owned by, a subgroup of CSA engineers responsible for a specific part of the project. Further, all CSA engineers considered it natural to relate their work to the users' tasks, which were the common referent of the scenarios. This can be illustrated by some of the uses to which the scenarios were put. As previously discussed, the scenarios generated a number of the events and elementary processes, which made up the business model, and they were a defining input in the development of the dialogue flow of the UI prototype. In addition, the scenarios were, for example, pivotal to a joined effort to establish the status of the project after six months had elapsed. In establishing the status of their project the CSA engineers preferred the scenarios as their base representation and determined outstanding issues by relating their other design artefacts to the scenarios:

> *[Two CSA engineers] have gone over it [i.e., the business model] and asked whether what is described in the scenarios can be found in the business model – yes or no. The outcome of that was a list of outstanding issues. Then [a third CSA engineer] and I have gone over the requirements specification and matched it against the scenarios, and that has produced a list of outstanding issues. Finally, I have started looking at the places in the scenarios where it says "to be clarified", "ask the user representatives", "ask the service consultants" – that kind of things. […] When that is completed we can look at status.* (Project meeting beginning of month 7)

Apart from bridging gaps between the design artefacts the scenarios mediated among the CSA engineers by providing a common ground the CSA engineers could relate their own work to and use in maintaining a feel for what the others were working on. This way the scenarios attained a central position as the design artefact that bound the design process together. In the words of Star and Griesemer (1989) the scenarios provided the CSA engineers with a design artefact that was "*both adaptable to different viewpoints and robust enough to maintain identity across them*" – a boundary object.

Naur (1995) suggests that the core of organised knowledge is best considered to be coherent description, as opposed to rules and other formalisations. This view discards any distinction between base-level descriptive accounts and higher-level analytic accounts that extend mere descriptions with explanations of the described phenomena and insights into their structure. What makes certain descriptions stand out is their coherence. Coherence is a composite quality which involves, among other things, the use of ways of description that fit the concrete situation and a narrative element that lends flow to the description. Further, Johnson-Laird and Wason (1977) have vividly illustrated people's superiority in dealing with concrete descriptions of real-world affairs, as opposed to abstract descriptions (see note). Whereas abstract descriptions tend to be experienced as logical puzzles, concrete descriptions of real-world affairs seem to tie in with people's general abilities to deal with their world and to be experienced as much more straightforward. Thus, the coherence and concrete, real-world feel of the scenarios may be distinct advantages, which made the CSA engineers better able to grasp CSA work and reason about the suitability of different design ideas.

7 Toward implications for conceptual-design tools

The CSA engineers recorded and managed the requirements and scenarios by means of a standard text-processing system. The shift from text processing to CASE tool did not occur until a fairly comprehensive version of the business model had been developed, and this shift involved that everything was re-keyed into the CASE tool. Apparently, the CASE tool bypasses requirements engineering and only supports business modelling by providing facilities for recording such models. According to Lubars, Potts, and Richter (1993), it is a general shortcoming of CASE tools that they do not address the transition from informal to more formal representations. The duration and complexity of the conceptual design of the CSA system suggest a need for tools that support conceptual design as it unfolds. This section points to three potential implications of the CSA project as regards support for the *process* of conceptual design. It should be borne in mind that though these potential implications seem to accord with previous studies (Section 2) they derive from a single case.

7.1 COMMON GROUND

The CASE tool used in the CSA project focuses chiefly on stringing the software-engineering process together from conceptual design to code generation. Based on the CSA project this study argues that during conceptual design the focus should rather be on stringing the requirements together in a coherent description of the users' work. This is primarily a matter of supporting software engineers in reaching, in their minds, a shared and fairly detailed understanding of the users' work. Software engineers need this understanding to be able to reason about the work domain and the role of the various activities and artefacts that are involved in performing the work. Without such an understanding, tools capable of holding models and other representations of the users' work will be of little use (see also Schmidt & Bannon, 1992). In the CSA project, the difference between shared tools and a common ground was apparent in relation to the requirements specification and the business model, which both focused on enumerating large amounts of detail but without much in the way of a scaffolding for relating the details to each other. Neither of these two design artefacts provided the support necessary for the CSA engineers to create a common ground. In contrast the scenarios succeeded in providing reasonably coherent descriptions of the users' tasks in a way that was apprehended by the entire group of CSA engineers and not owned by – or relevant to – any one subgroup in particular. Specifically, the property that the scenarios focused on the users' work, rather than the system under construction, made them neutral to the occasionally heated debates between the CSA engineers responsible for the design and technical implementation of the system and the CSA engineers responsible for its utility and organisational implementation.

7.2 EVOLVING DESCRIPTIONS

The notion of evolving descriptions is a generalisation of how the scenarios supported the CSA engineers by allowing for a gradual refinement of coherent task descriptions while at the same time preserving a stable and easy-to-understand structuring of the users' work into tasks. During conceptual design such descriptions will evolve along a number of dimensions – such as from naturalistic toward more schematised, from tasks toward subtasks, from tasks toward goals, from prescribed procedures toward actual activities, and from normal cases toward inclusion of exceptions. Initially, tasks, or the entire work domain, may be described by means of, for example, recordings of interviews with users or other people knowledgeable about the domain. It is, on the one hand, important that such first-hand but often partial descriptions are acknowledged as conceptual-design data and stored in a systematic way. On the other hand, software engineers must also be provided with facilities for leaving a trace that indicates how these original data, or snippets thereof, are evolved into refined design descriptions. Such facilities could simply support the creation of ad-hoc links and annotations but they could also define a select set of dimensions along which the descriptions should evolve. Work domains can, for example, be analysed in terms of goals, scenarios, and system functions. Kaindl (2000) proposes a design process in which an initial and incomplete set of goals, scenarios, and system functions is used to refine and complement these goals, scenarios, and system functions. By analysing the hitherto identified scenarios it may become visible that some scenarios lack supporting system functions, that others cannot be linked to any identified goals, and that still others entail conflicts among goals. Similarly, analyses of the hitherto identified goals and system functions may lead to modifications of the scenarios. The CSA engineers were less systematic in their use of scenarios but they, too, used scenarios as a pivotal element in evolving a reasonably complete and comprehensible description of the work domain.

7.3 OPPORTUNISTIC USE

In the CSA project, scenarios were a major design artefact until they were discontinued. Similarly, Weidenhaupt et al. (1998) found that scenarios were rarely maintained throughout the surveyed projects and could, therefore, neither be used in the later stages of the projects, nor as system documentation. This is not simply a matter of poor scenario management or inadequate tool support for SBD. Software-engineering projects have tight deadlines and will inevitably go through a series of changes that necessitate modifications of the scenarios. So far, the few studies of how practising software engineers use scenarios in real-world projects indicate that systematic use of scenarios throughout the system lifecycle requires an unrealistic commitment to and diligence in scenario management given the realities of most projects. Instead, scenarios are used at selected points in the system lifecycle when software engineers perceive that scenarios may have a direct impact on the progress of the project. During the in-between periods where scenarios are not contributing directly to the software engineers' current activities the maintenance and management of the scenarios is likely to be postponed or downgraded in favour of activities that yield more immediate gains. Such opportunistic use of design artefacts is not exclusive to scenarios; it is also evident in relation to, for example, software-engineering methods (Bansler & Bødker, 1993) and design documentation (Button & Sharrock, 1996). To the practitioner this probably means that more can be gained from scenarios if it is decided to invest more resources in them. To the researcher it means that in spite of the seeming advantages of using scenarios systematically throughout the system lifecycle, SBD tools and approaches should accommodate opportunistic use as that is probably how they are chiefly going to be used.

8 Conclusion

During conceptual design software engineers develop a series of design artefacts as they go through requirements engineering, business modelling, and functional specification. While scenarios can be used for multiple purposes throughout the software-engineering lifecycle they have received most attention as a design artefact for grounding conceptual design in the users' work. SBD has gained acceptance in research as well as practice but there is very little empirical material available on how practising software engineers have applied scenarios in real-world projects, and to what effect. This longitudinal field study has analysed the use and role of scenarios in the complete redesign of a large information system.

The distinguishing features of the scenarios, relative to the requirements specification, the business model, and the UI prototype, were that they were task-based and descriptive. By being *task-based* the scenarios strung individual events and activities together in purposeful sequences and, thereby, provided an intermediate level of description that was both an instantiation of overall work objectives and a fairly persistent context for the gradual elaboration of subtasks. In contrast the three other design artefacts were mostly at the subtask level and left it to the reader to derive the dependencies and other relations among individual requirements, events, and activities. By being *descriptive* the scenarios preserved a real-world feel of the contents and flow of the users' work. In contrast the requirements specification and the business model were extensive lists that enumerated large amounts of separate details, and the UI prototype demonstrated the new design rather than described the users' work. This way the scenarios made the users' work recognisable to the software engineers as a complex but organised human activity.

The scenarios attained a unifying role as mediator among both the design artefacts and the software engineers. This evidences that the scenarios, temporarily, succeeded in describing the users' tasks in a way that was considered relevant and useful by all the software engineers across their different responsibilities within the project. In contrast the business model was only used by the software engineers who worked with the CASE tool and they, in turn, did not use the UI prototype. The scenarios were not used for communication with the user representatives because the software engineers felt that would amount to telling the users something they already knew. Instead the scenarios were a major input to the design of the UI prototype, and successive versions of the UI prototype, which showed how the software engineers envisaged the new system and use situation, were discussed with the user representatives. The creation and management of the scenarios were, however, time-consuming and only a few of the software engineers understood the users' work well enough to produce the more fine-grained parts of the scenarios. These few software engineers were also the driving forces of several other project activities. To make progress on these other activities the project manager discontinued the scenarios although they were valuable to the conceptual design, could have made useful design documentation, and could – potentially – have provided a common point of reference throughout the system lifecycle.

In terms of implications for the design of tools that support software engineers this study suggests a need for support during the early and rather open-ended activities that constitute conceptual design. Along with the few previous empirical studies of SBD this study of a redevelopment project suggests several hypotheses for further research. First, software engineers seemingly need to establish a common ground that strings requirements and other pieces of information together in a coherent understanding of the users' work. Second, software engineers seem to need evolving descriptions, which are capable of gradual refinement while at the same time preserving a stable overall structure, similar to how the scenarios structured the users' work into a fairly stable set of tasks that were gradually specified in more and more detail. Third, it seems as if scenarios are rarely used and maintained systematically throughout the system lifecycle. Whereas this calls for better tools and procedures for scenario management it also means that SBD tools and approaches should accommodate opportunistic use as that is probably how most software engineers will use scenarios. Finally, the software engineers in the studied project valued concrete, coherent descriptions of the users' present way of accomplishing their work over the risk of missing some re-conceptions of the users' work.

Acknowledgements

This work has been supported by the Danish National Research Foundation through its funding of the Centre for Human-Machine Interaction. For comments on an earlier version of this paper, I wish to thank Erik Frøkjær, Niels Jacobsen, the anonymous reviewers, and my colleagues at Risø National Laboratory. Special thanks are due to the members of the CSA project group who have put up with my presence in spite of their busy schedule.

Note

Johnson-Laird and Wason (1977) gave a number of subjects one of two tasks. The exact formulation of the tasks has been subject to much experimentation. The formulations given here are representative of the abstract condition (Task 1) and the concrete condition (Task 2):

1. You are presented with a stack of cards with either the letter A or the letter E on one side and either the digit 4 or the digit 7 on the other side. The cards are stacked with either side up, at random, and your task is to go through the cards and turn over only those cards necessary to establish whether the cards satisfy the rule: If the letter side of the card is an A, then the digit side must be a 4.
2. You are a cashier at a supermarket and have the checks received that day stacked before you, some face up and some face down. Your task is to go through the checks and turn over only those checks necessary to establish whether they satisfy the rule that checkout people are to accept checks for more than $50 only if approved on the back by the manager.

Although the two tasks are logically identical, the subjects were much less successful in establishing that they had to turn over cards showing *A*s and *7*s only than in establishing that they had to turn over checks for more than $50 and those with no approval on their backs.

References

BANSLER, J. P. & BØDKER, K. (1993). A reappraisal of Structured Analysis: Design in an organizational context. *ACM Transactions on Information Systems*, **11**, 2, 165-193.

BUTTON, G. & SHARROCK, W. (1996). Project work: The organisation of collaborative design and development in software engineering. *Computer Supported Cooperative Work (CSCW)*, **5**, 4, 369-386.

BØDKER, S. & CHRISTIANSEN, E. (1997). Scenarios as springboards in CSCW design. In G. C. BOWKER, S. L. STAR, W. TURNER & L. GASSER, Eds. *Social Science, Technical Systems, and Cooperative Work: Beyond the Great Divide*, pp. 217-233. Mahwah, NJ: Lawrence Erlbaum.

CARROLL, J. M., Ed. (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development*. New York: Wiley.

CARROLL, J. M. (1997). Scenario-based design. In M. HELANDER, T. K. LANDAUER & P. PRABHU, Eds. *Handbook of Human-Computer Interaction. Second, Completely Revised Edition*, pp. 383-406. Amsterdam: Elsevier.

CARROLL, J. M. (2000). *Making Use: Scenario-Based Design of Human-Computer Interactions*. Cambridge, MA: MIT Press.

CARROLL, J. M., KELLOGG, W. A. & ROSSON, M. B. (1991). The task-artifact cycle. In J. M. CARROLL, Ed. *Designing Interaction: Psychology at the Human-Computer Interface*, pp. 74-102. Cambridge: Cambridge University Press.

CARROLL, J. M. & ROSSON, M. B. (1992). Getting around the task-artifact cycle: How to make claims and design by scenario. *ACM Transactions on Information Systems*, **10**, 2, 181-212.

CARROLL, J. M., ROSSON, M. B., CHIN, G. & KOENEMANN, J. (1998). Requirements development in scenario-based design. *IEEE Transactions on Software Engineering*, **24**, 12, 1156-1170.

CONSTANTINE, L. L. & LOCKWOOD, L. A. D. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Reading, MA: Addison-Wesley.

CURTIS, B., KRASNER, H. & ISCOE, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, **31**, 11, 1268-1287.

DZIDA, W. & FREITAG, R. (1998). Making use of scenarios for validating analysis and design. *IEEE Transactions on Software Engineering*, **24**, 12, 1182-1196.

GOUGH, P. A., FODEMSKI, F. T., HIGGINS, S. A. & RAY, S. J. (1995). Scenarios – An industrial case study and hypermedia enhancements. In *Proceedings of the Second International Symposium on Requirements Engineering (RE'95)*, pp. 10-17. Los Alamitos, CA: IEEE Computer Society Press.

HAUMER, P., POHL, K. & WEIDENHAUPT, K. (1998). Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering*, **24**, 12, 1036-1054.

HERTZUM, M. (2000). People as carriers of experience and sources of commitment: Information seeking in a software design project. *New Review of Information Behaviour Research*, **1**, 135-149.

HERTZUM, M. (2002). Small-scale classification schemes: A field study of requirements engineering. Submitted for publication.

HOOPER, J. W. & HSIA, P. (1982). Scenario-based prototyping for requirements identification. *ACM SIGSOFT Software Engineering Notes*, **7**, 5, 88-93.

JACOBSON, I. (1995). The use-case construct in object-oriented software engineering. In J. M. CARROLL, Ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, pp. 309-336. New York: Wiley.

JACOBSON, I., BOOCH, G. & RUMBAUGH, J. (1999). *The Unified Software Development Process*. Reading, MA: Addison-Wesley.

JARKE, M., Ed. (1998a). Interdisciplinary uses of scenarios [Special issue]. *Requirements Engineering*, **3**, 3&4.

JARKE, M. (1998b). Guest editorial: Interdisciplinary uses of scenarios. *Requirements Engineering*, **3**, 3&4, 153-154.

JARKE, M., BUI, X. T. & CARROLL, J. M. (1998). Scenario management: An interdisciplinary approach. *Requirements Engineering*, **3**, 3&4, 155-173.

JARKE, M. & KURKI-SUONIO, R., Eds. (1998). Scenario management [Special issue]. *IEEE Transactions on Software Engineering*, **24**, 12.

JOHNSON-LAIRD, P. N. & WASON, P. C. (1977). A theoretical analysis of insight into a reasoning task. In P. N. JOHNSON-LAIRD & P. C. WASON, Eds. *Thinking: Readings in Cognitive Science*, pp. 143-157. Cambridge: Cambridge University Press.

KAINDL, H. (2000). A design process based on a model combining scenarios with goals and functions. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **30**, 5, 537-551.

KAMINSKY, S. K. (1992). Test early, test often: A formative usability kit for writers. In *Proceedings of the International Conference on Computer Documentation (SIGDOC'92)*, pp. 47-55. New York: ACM Press.

KARAT, J. & BENNETT, J. L. (1991). Using scenarios in design meetings – A case study example. In J. KARAT, Ed. *Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design*, pp. 63-94. San Diego, CA: Academic Press.

KAZMAN, R., ABOWD, G., BASS, L. & CLEMENTS, P. (1996). Scenario-based analysis of software architecture. *IEEE Software*, **13**, 6, 47-55.

KYNG, M. (1995). Creating contexts for design. In J. M. CARROLL, Ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, pp. 85-108. New York: Wiley.

LUBARS, M., POTTS, C. & RICHTER, C. (1993). A review of the state of the practice in requirements engineering. In *Proceedings of the International Symposium on Requirements Engineering (RE'93)*, pp. 2-14. Los Alamitos, CA: IEEE Computer Society Press.

NAUR, P. (1995). Coherent description as the core of scholarship and science. In P. NAUR, *Knowing and the Mystique of Logic and Rules*, pp. 317-350. Dordrecht: Kluwer.

NIELSEN, J. (1995). Scenarios in discount usability engineering. In J. M. CARROLL, Ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, pp. 59-84. New York: Wiley.

POTTS, C. & CATLEDGE, L. (1996). Collaborative conceptual design: A large software project case study. *Computer Supported Cooperative Work (CSCW)*, **5**, 4, 415-445.

RASMUSSEN, J., PEJTERSEN, A. M. & GOODSTEIN, L. P. (1994). *Cognitive Systems Engineering*. New York: Wiley.

SCHMIDT, K. & BANNON, L. (1992). Taking CSCW seriously - Supporting articulation work. *Computer Supported Cooperative Work (CSCW)*, **1**, 1&2, 7-40.

SMITH, W., DOWELL, J. & ORTEGA-LAFUENTE, M. A. (1999). Designing paper disasters: An authoring environment for developing training exercises in integrated emergency management. *Cognition, Technology & Work*, **1**, 2, 119-131.

STAR, S. L. & GRIESEMER, J. R. (1989). Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, **19**, 3, 387-420.

WEIDENHAUPT, K., POHL, K., JARKE, M. & HAUMER, P. (1998). Scenarios in system development: Current practice. *IEEE Software*, **15**, 2, 34-45.

Table 1. The roles of scenarios

Domain	Scenarios are used as a means to
Strategic management	• Recognise unexpected changes • Protect against judgement errors by flushing out invalid mindsets or assumptions • Use the most plausible mindsets and assumptions as the basis for development • Monitor fallback scenarios for possible modification of development strategy
Human-computer interaction	• Focus design efforts on use • Suspend commitment but support concrete progress • Provide a task-oriented design decomposition that can be used from many perspectives • Codify design knowledge as a 'middle-level abstraction' • Encourage and support participatory design
Software engineering	• Make abstract models concrete • Reach partial agreement and consistency of understanding • Provide a decomposition mechanism for managing complex projects • Provide a linkage mechanism between development phases • Support object models by functioning as design aids and boundary conditions

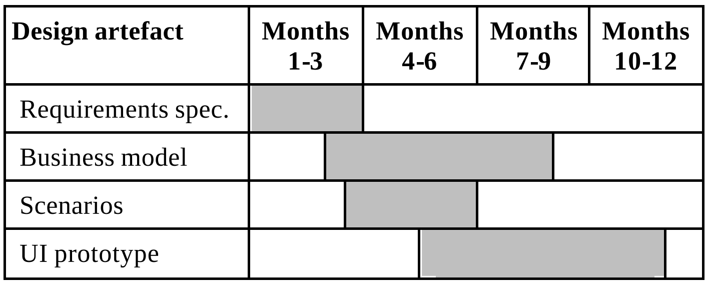

Figure 1. The four main design artefacts produced during the first twelve months of the CSA project

	Event	**Business process**	**Elementary process**
6	PE sends request	Process received request	1. Record the receipt of a request for a request-demanding payment 2. Release this request-demanding payment for processing
7	PE supplies information about the child's income	Process information about child's income	1. Record the supplied information about the child's income - Amount - Assessment: temporary or permanent 2. Evaluate the rule set regarding the max limit of children's income. If the income exceeds the max limit then: PE-Event-Child's-income-exceeds-max-limit
8	PE-Event-Child's-income-exceeds-max-limit		1. Discontinue advance payments 2. Inform PE about the ruling to discontinue advance payments 3. Inform PP's municipality that the child's income is of relevance: PP-Event-Child's-income-exceeds-max-limit

Figure 2. Sample scenario steps (PE is an abbreviation for the **p**erson who is **e**ntitled to CSA; PP is an abbreviation for the **p**erson who is obliged to **p**ay CSA)

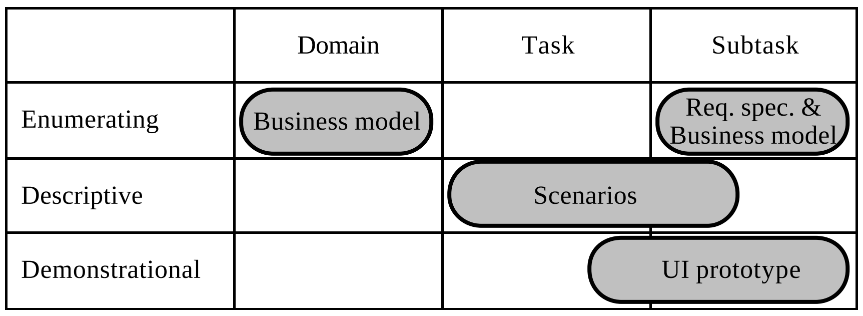

Figure 3. The scope of the four main design artefacts